UNIVERSITY OF CALIFORNIA,
IRVINE

Foot-mounted Pedestrian Inertial Navigation Systems for Self-contained Tracking

DISSERTATION

submitted in partial satisfaction of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

in Mechanical and Aerospace Engineering

by

Chi-Shih Jao

Dissertation Committee:
Professor Andrei M. Shkel, Chair
Professor Solmaz S. Kia
Professor Sharad Mehrotra

2023

DEDICATION

To the brave and selfless firefighters and first responders who risk their lives every day in dangerous situations to keep our communities safe, you are the true heroes of our society.

TABLE OF CONTENTS

LIST OF ABBREVIATIONS

Page

5G	Fifth Generation	4
AC	Alternate Current	23
AcRW	Acceleration Random Walk	13
AMV	Acceleration-Moving Variance	48
ANN	Artificial Neural Network	311
AoA	Angle of Arrival	27
APUAVD	International Conference Actual Problems of Unmanned Aerial Vehicles Developments	320
ARE	Angular Rate Energy	21
ARW	Angular Random Walk	13
ASPIN	Autonomous Systems Perception, Intelligence, & Navigation	219
BPNN	Back-Propagation Neural Network	21
CAD	Computer-Aided Design	151
CCDC	Chinese Control And Decision Conference	336
CEP	Circular Error Probable	48
CMOS	Complementary Metal-Oxide Semiconductor	352
CNN	Convolutional Neural Network	22
CoM	Center of Mass	41
CONECCT	International Conference on Electronics, Computing and Communication Technologies	332
COTS	Commercial-Off-The-Shelf	5
CPU	Central Processing Unit	221
CVPR	Computer Vision and Pattern Recognition Conference	324
DCM	Direction Cosine Matrix	10
DIO	Deep-Inertial Odometry	6
DNN	Deep Neural Network	206
DoA	Direction-of-Arrival	215
DoF	Degree of Freedom	xxxii
dps	degrees per second	86
DVS	Dynamic Vision Sensor	92
ECC	European Control Conference	333
EFIR	Extended Finite Impulse Response	29
EKF	Extended Kalman Filter	xxxiii
EMG	ElectroMyoGraphy	22
ENC	European Navigation Conference	323
FIBA	Foot-Instability-Based Adaptive	116
FOV	Field Of View	95
FPS	Frame Per Second	352
FSR	Full-Scale Range	xxxii
FUSION	2018 21st International Conference on Information Fusion	332
GigE	Gigabit Ethernet	151
GLRT	General Likelihood Ratio Test	21

ToA	Time of Arrival	27
ToF	Time of Fly	29
UART	Universal Asynchronous Receiver-Transmitter	217
UbiComp	Ubiquitous Computing	331
uFINS	Ultimate Foot-mounted Inertial Navigation System	11
UKF	Unscented Kalman Filter	29
UPINLBS	Ubiquitous Positioning, Indoor Navigation, and Location Based Service	320
USRP	Universal Software Radio Peripheral	219
UWB	UltraWide Band	xxi
VDOP	Vertical Dilution Of Precision	228
VRW	Velocity Random Walk	13
WCSP	Wireless Communications and Signal Processing	337
WPNC	Workshop on Positioning, Navigation and Communication	325
ZARU	Zero Angular Rate Update	24
ZUPT	Zero velocity UPdaTe	xii

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

ACKNOWLEDGMENTS

First, I would like to thank my Ph.D. advisor, Professor Andrei M. Shkel, for serving as my dissertation chair, bringing me into the field of navigation and indoor positioning, and giving me an opportunity to devote myself to solving challenges in this domain. His support, guidance, and advice have made this Ph.D. journey a pleasant and unforgettable experience. I would also like to thank my dissertation committee members, Professor Solmaz Kia and Professor Sharad Mehrotra, for their time reviewing this thesis and providing insightful comments and suggestions.

I am grateful to my colleagues and friends at UCI Microsystems Laboratory for their guidance, collaboration, and support. More specifically, I would like to thank Dr. Sina Askari and Dr. Yusheng Wang for mentoring me when I first joined the lab and rigorously training me on the fundamentals of statistical estimation and system integration, which later became the building blocks for me to realize many research ideas; I would like to thank Dr. Danmeng Wang for her invaluable suggestions on inertial sensor characterization and delightful collaboration; I would like to thank Dr. Mohammad H. Asadian, Dr. Radwan Mohammed Noor, Yu-Wei Lin, Dr. Daryosh Vatanparvar, Doreen Hii, Austin Parrish, Eudald Sangenis, and Lois Meira for the help in various aspects in the lab and mental supports.

I would also like to thank Professor Zak Kassas, Professor Solmaz Kia, and their students, Ali Abdullah, Dr. Jianan Zhu, Changwei Chen, and Minwon Seo, for their advice and collaboration in the uNavChip project, which was not only a great learning experience but also enlightened me to look at the problem of indoor positioning from different perspectives. I would like to thank Joe Grasso for his help and suggestion on performance evaluations of navigation systems. I would also like to thank Professor Emre Neftci and his student Kenneth M. Stewart for their collaboration and advice regarding the research on dynamic vision sensors. I would like to thank Professor Alexandra Voloshina and her student Paula Simo for their assistance in developing pedestrian navigation simulations.

I also would like to thank funding sources provided by U.S. Department of Commerce, National Institute of Standards and Technology (NIST) under Contract No. 70NANB17H192, 70NANB21H154, and 70NANB22H073.

Finally, I would like to express my gratitude to my parents, my brother, my wife, and my friends for their unconditional encouragement, support, and love.

VITA

Chi-Shih Jao

EDUCATION

Doctor of Philosophy in Mechanical and Aerospace Engineering **2023**
University of California, Irvine *Irvine, CA*

Master of Science in Mechanical and Aerospace Engineering **2022**
University of California, Irvine *Irvine, CA*

Master of Science in Electrical Engineering **2018**
The Pennsylvania State University, University Park *State College, PA*

Bachelor of Science in Electrical Engineering **2015**
National Tsing Hua University *Hsinchu, Taiwan*

RESEARCH EXPERIENCE

Graduate Research Assistant **2018–2023**
University of California, Irvine *Irvine, CA*

Graduate Research Assistant **2016–2018**
The Pennsylvania State University, University Park *State College, PA*

Undergraduate Research Assistant **2014–2015**
National Tsing Hua University *Hsinchu, Taiwan*

TEACHING EXPERIENCE

Teaching Assistant **2023**
University of California, Irvine *Irvine, CA*

JOURNAL PUBLICATIONS

Chi-Shih Jao, Danmeng Wang, Changwei Chen, Eudald Sangenis, Joe Grasso, Solmaz S. Kia, and Andrei M. Shkel, "UWB-Foot-SLAM2: Using Multi-Sensor Enhancement for Foot-mounted Pedestrian INS And Simultaneously Localized UWB Beacons", *IEEE Journal of Indoor and Seamless Positioning and Navigation*, 2023. (*In Preparation*)

Chi-Shih Jao, Ali A Abdallah, Changwei Chen, Min-Won Seo, Solmaz S Kia, Zaher M Kassas, Andrei M. Shkel, "PINDOC: Pedestrian indoor navigation system integrating deterministic, opportunistic, and cooperative functionalities", *IEEE Sensors Journal*, vol. 22, no. 14, pp. 14424–14435, 2022.

Austin R Parrish, **Chi-Shih Jao**, Andrei M. Shkel, "Stance phase detection for ZUPT-aided INS using knee-mounted IMU in crawling scenarios", *IEEE Sensors Letters*, vol. 6, no. 5, pp. 1–4, 2022.

Changwei Chen, **Chi-Shih Jao**, Andrei M. Shkel, Solmaz S Kia, "UWB sensor placement for foot-to-foot ranging in dual-foot-mounted ZUPT-aided INS", *IEEE Sensors Letters*, vol. 6, no. 2, pp. 1–4, 2022.

Chi-Shih Jao, Andrei M. Shkel, "A reconstruction filter for saturated accelerometer signals due to insufficient FSR in foot-mounted inertial navigation system", *IEEE Sensors Journal*, vol. 22, no. 1, pp. 695–706, 2021.

Ali A Abdallah, **Chi-Shih Jao**, Zaher M Kassas, Andrei M. Shkel, "A pedestrian indoor navigation system using deep-learning-aided cellular signals and ZUPT-aided foot-mounted IMUs", *IEEE Sensors Journal*, vol. 22, no. 6, pp. 5188–5198, 2021.

Chi-Shih Jao, Andrei M. Shkel, "ZUPT-aided INS bypassing stance phase detection by using foot-instability-based adaptive covariance", *IEEE Sensors Journal*, vol. 21, no. 21, pp. 24338-24348, 2021.

Yusheng Wang, **Chi-Shih Jao**, Andrei M. Shkel, "Scenario-dependent ZUPT-aided pedestrian inertial navigation with sensor fusion", *Journal of Gyroscopy and Navigation*, vol. 12, no. 1, pp. 1–16, 2021.

CONFERENCE PUBLICATIONS

Chi-Shih Jao, Danmeng Wang, Joe Grasso, and Andrei M. Shkel, "UWB-Foot-SLAM: Bounding Position Error of Foot-mounted Pedestrian INS with Simultaneously Localized UWB Beacons", *IEEE/ION Position, Location and Navigation Symposium (PLANS)*, Monterey, CA, Apr. 24-27, 2023.

Chi-Shih Jao, Danmeng Wang, and Andrei M. Shkel, "Prio-IMU: Prioritizable IMU Array for Enhancing Foot-Mounted Inertial Navigation Accuracy", *IEEE International Symposium on Inertial Sensors and Systems (INERTIAL)*, Kauai, Hawaii, Mar. 28-31, 2023.

Chi-Shih Jao, Eudald Sangenis, Paula Simo, Alexandra Voloshina, Andrei M. Shkel, "An Inverted Pendulum Model of Walking for Predicting Navigation Uncertainty of Pedestrian in Case of Foot-mounted Inertial Sensors", *IEEE International Symposium on Inertial Sensors and Systems (INERTIAL)*, Kauai, Hawaii, Mar. 28-31, 2023.

Austin R. Parrish, **Chi-Shih Jao**, Danmeng Wang, and Andrei M. Shkel, "Study of IMU Mounting Position for ZUPT-Aided INS in the Case of Firefighter Crawling", *IEEE International Symposium on Inertial Sensors and Systems (INERTIAL)*, Kauai, Hawaii, Mar. 28-31, 2023.

Austin R. Parrish, **Chi-Shih Jao**, Danmeng Wang, and Andrei M. Shkel, ""Sugar-Cube": Pedestrian Hardware Platform That Fits in the Sole of a Shoe", *IEEE International Symposium on Inertial Sensors and Systems (INERTIAL)*, Kauai, Hawaii, Mar. 28-31, 2023.

Eudald Sangenis, **Chi-Shih Jao**, Andrei M. Shkel, "SVM-based Motion Classification Using Foot-mounted IMU for ZUPT-aided INS", *IEEE Sensors*, Dalla, TX, USA, Oct 30-2, 2022.

Chi-Shih Jao, Ali A. Abdallah, Changwei Chen, Minwon Seo, Solmaz S. Kia, Zaher M. Kassas, Andrei M. Shkel, "Sub-meter accurate pedestrian indoor navigation system with dual ZUPT-aided INS, machine learning-aided LTE, and UWB signals", *the 35th International Technical Meeting of the Satellite Division of The Institute of Navigation (ION GNSS+ 2022)*, Denver, CO, USA, Sep 19-23, 2022.

Chi-Shih Jao, Danmeng Wang, Austin R Parrish, Andrei M. Shkel, "A neural network approach to mitigate thermal-induced errors in ZUPT-aided INS", *IEEE International Symposium on Inertial Sensors and Systems (INERTIAL)*, Avignon, France, May 8-11, 2022.

Chi-Shih Jao, Austin Parrish, and Andrei M. Shkel. ""Sugar-Cube" PLT: A Real-time Pedestrian Localization Testbed Utilizing Foot-mounted IMU/Barometer/Ultrasonic Sensors." *IEEE Sensors*, Virtual Conference, Oct. 31-4, 2021.

Chi-Shih Jao, Yusheng Wang, and Andrei M. Shkel, "A zero velocity detector for foot-mounted inertial navigation systems aided by downward-facing range sensor", *IEEE Sensors*, Virtual Conference, Oct. 25-28, 2020.

Chi-Shih Jao, Yusheng Wang, Yu-Wei Lin, Andrei M. Shkel, "A hybrid barometric/ultrasonic altimeter for aiding ZUPT-based inertial pedestrian navigation systems", *the 33rd International Technical Meeting of the Satellite Division of The Institute of Navigation (ION GNSS+ 2022)*, Virtual Conference, Sep 21-25, 2020.

Chi-Shih Jao, Kenneth Stewart, Jörg Conradt, Emre Neftci, and Andrei M. Shkel, "Zero Velocity Detector for Foot-mounted Inertial Navigation System Assisted by a Dynamic Vision Sensor", *DGON Inertial Sensors and Systems (ISS)*, Virtual Conference, Sep. 15-16, 2020.

Chi-Shih Jao, Yusheng Wang, Sina Askari, Andrei M. Shkel, "A closed-form analytical estimation of vertical displacement error in pedestrian navigation", *IEEE/ION Position, Loca-*

tion and Navigation Symposium (PLANS), Virtual Conference, Apr 20-23, 2020.

Yusheng Wang, Yu-Wei Lin, Sina Askari, **Chi-Shih Jao**, Andrei M. Shkel, "Compensation of systematic errors in ZUPT-aided pedestrian inertial navigation", *IEEE/ION Position, Location and Navigation Symposium (PLANS)*, Virtual Conference, Apr 20-23, 2020.

Chi-Shih Jao, Yusheng Wang, Andrei M. Shkel, "Pedestrian inertial navigation system augmented by vision-based foot-to-foot relative position measurements", *IEEE/ION Position, Location and Navigation Symposium (PLANS)*, Virtual Conference, Apr 20-23, 2020.

Sina Askari, **Chi-Shih Jao**, Yusheng Wang, Andrei M. Shkel, "Learning-based calibration decision system for bio-inertial motion application", *IEEE Sensors*, Montreal, Canada, Oct. 27-30, 2019.

Sina Askari, **Chi-Shih Jao**, Yusheng Wang, Andrei M. Shkel, "A laboratory testbed for self-contained navigation", *IEEE International Symposium on Inertial Sensors and Systems (INERTIAL)*, Naples, FL, USA, Apr 1-5, 2019.

Yusheng Wang, Sina Askari, **Chi-Shih Jao**, Andrei M. Shkel, "Directional ranging for enhanced performance of aided pedestrian inertial navigation", *IEEE International Symposium on Inertial Sensors and Systems (INERTIAL)*, Naples, FL, USA, Apr 1-5, 2019.

ABSTRACT OF THE DISSERTATION

Foot-mounted Pedestrian Inertial Navigation Systems for Self-contained Tracking

By

Chi-Shih Jao

Doctor of Philosophy in Mechanical and Aerospace Engineering

University of California, Irvine, 2023

Professor Andrei M. Shkel, Chair

This Ph.D. dissertation presented the research on development of pedestrian inertial navigation systems that track a person in a self-contained and infrastructure-free manner. The investigated approach utilized Zero-velocity UPdaTe (ZUPT)-aided Inertial Navigation Systems (INS) based on foot-mounted Inertial Measurement Units (IMUs). Prior implementations of the ZUPT-aided INS suffer from error sources originating from motion sensors, algorithm, and estimation. This dissertation attempted to address the identified error sources and proposed new algorithmic and system-level approaches. The main contributions of this thesis include:

- approaches, on both algorithm and system levels, to address issues of insufficient inertial sensors' Full-Scale Range (FSR) and bandwidth when mounting on the foot. On the algorithm level, a reconstruction filter was developed, predicting accelerometers' signals exceeding measurable ranges. On the system level, an IMU array with a prioritization mechanism was developed, enabling measurements of acceleration as large as 200 [g]. The two approaches were experimentally verified to improve the positioning accuracy of a ZUPT-aided INS in cases of foot-mounted sensors experiencing large shocks.
- a neural network-based approach to predict and compensate 12 different thermal-induced errors, consisting of bias and noise variations in a 6-Degree of Freedom (DoF)

IMU, based on temperatures and temperature rates. It was experimentally verified that the developed temperature compensation approach reduced navigation error of ZUPT-aided INS by more than 15×, as compared to a traditional ZUPT-aided INS, in a scenario where ambient temperature varied with a range of 30°C.

- improved traditional IMU-based stance phase detection by fusing measurements of a foot-mounted IMU and a downward-facing ultrasonic sensor. The ultrasonic sensor was demonstrated with preferred properties that its measurements had a unique value only when the foot is in contact with the ground and the value does not vary in different pedestrian activities. It was demonstrated, with experiments involving walking and running, that the accuracy of the developed stance phase detector outperformed the traditional IMU-based detector and improved navigation accuracy of the ZUPT-aided INS by more than 2×.

- an adaptive mechanism to vary the covariance of the zero-velocity measurements based on log-likelihood ratio metrics derived from measurements of a foot-mounted IMU. It was demonstrated that the adaptive covariance could bypass the necessity of using a stance phase detector in a ZUPT-aided INS. Results collected from indoor navigation experiments showed that the navigation accuracy of the ZUPT-aided INS using the developed adaptive covariance mechanism improved horizontal and vertical position accuracy by 36% and 64%, respectively, as compared to the case of conventional ZUPT-aided INS using a stance phase detector.

- a vision-based foot-to-foot positioning augmentation to increase the observability of the Extended Kalman Filter (EKF) on the yaw angle state in a dual foot-mounted IMU framework. This approach utilized a camera mounted on one foot to capture images of a feature pattern mounted on the other foot, and relative positions between the two feet were derived from the images. Numerical simulation and experimental results showed that the ZUPT-aided INS improved navigation accuracy by over 90%

and 85%, respectively, as compared to the traditional ZUPT-aided INS.

- developed and demonstrated, for the first time, a hybrid barometric/ultrasonic altimeter that considered a downward-facing ultrasonic sensor was mounted on foot to improve a ZUPT/Altimeter-aided INS framework, minimizing sensitivity of altitude measurements to variations in ambient temperature and air pressure. Experimental results showed that, in the case of abrupt temperature changes, the vertical displacement accuracy of ZUPT-aided INS augmented with the developed hybrid altimeter had a 96% and a 97% improvement, as compared to a standalone ZUPT-aided INS and a ZUPT-aided INS augmented with a barometer, respectively.

- a Simultaneously Localization And Mapping (SLAM) framework based on foot-mounted IMUs and environmental-deployed UWB beacons, referred to as UWB-Foot-SLAM, where beacons used did not need to be pre-deployed and pre-surveyed but were distributed in an environment during a navigation task and provided position compensation to bound position error growth. Additionally, this thesis developed the UWB-Foot-SLAM2 algorithm that augmented the original UWB-Foot-SLAM with self-contained aiding techniques. Experimental results showed that the developed UWB-Foot-SLAM framework mapped unknown beacons with displacement errors less than 0.5 [m]. In an indoor navigation experiment involving a pedestrian traveling around 3.5 [km] for an hour in a three-floor 50 [m] $\times$15 [m] $\times$15 [m] building on terrains of flat planes, ramps, stairs, and elevators, the 3D loop-closure error of the UWB-Foot-SLAM2 algorithm was 0.62 [m]

Chapter 1

Introduction

This chapter introduces the research presented in this thesis; provides the background on pedestrian navigation systems; formulates the problems in a traditional implementation of ZUPT-aided INS using foot-mounted IMUs; reviews state-of-the-art approaches; and presents an overview and an outline of this thesis.

1.1 Motivation

1.1.1 Pedestrian Localization in Extreme Scenarios

An accurate and reliable positioning solution is critical for personnel safety in numerous applications, such as emergency response and military operations [165]. Firefighters, first responders, and war fighters can get disoriented very easily while operating under complex building types and harsh environmental conditions. Losing track of current positions has been one of the factors causing approximately 80 to 100 firefighters to be lost or injured in the line of duty each year in the U.S., according to the National Institute for Occupational

Safety and Health (NIOSH). Having position information in extreme circumstances can not only increase situational awareness and allow for effective communication for these public safety personnel but also enable incident commanders to quickly plan search, escape, and rescue strategies, increasing personal accountability [179].

In order to navigate in an unknown environment and safely return to exits in emergency situations, firefighters are currently trained to practice simple but robust techniques, such as following a fire hose or a dedicated rope that connects them to a point outside of dangerous areas, leaving flashlights in doorways to locate rooms' exits, or keeping their left or right hand in contact with the wall [111]. However, these approaches do not report instantaneous numerical positioning information to firefighters or incident commanders, making it difficult in some cases to perform immediate responses to the fast-changing firefighting scenarios, which can cause delays in life-saving missions. Developing localization systems for firefights and first responders is in high demand [51].

1.1.2 Navigation System Requirements for Worst-case Scenarios

Development of localization systems for firefighters, however, is a very challenging problem because a solution dedicated to this purpose needs to meet several strict requirements [249], including:

1. 1-meter accuracy for several minutes to hours: complex building interiors can consist of narrow hallways and multiple rooms;

2. environmental-independent: in addition to the need to navigate between indoor and outdoor environments, the emergency responders can encounter extreme conditions where visibility is poor due to smoke, mist, airborne particles, or low light intensity. Figure 1.1 shows an example of an extreme environment of structural firefighting sce-

Figure 1.1: A photo of a severe structural fire that occurred on December 3 1999 in Worcester, Massachusetts [52].

nario;

3. infrastructure-free: an assumption of being able to pre-deploy or access navigational infrastructures in the surrounding environment might not be feasible;

4. fast deployment: life-saving operations may require responses within a few seconds;

5. consistent availability: in harsh environments, accidents can happen any time and anywhere;

6. small-form-factor: large and heavy devices can become extra burdens restricting locomotion of firefighters in emergencies, who already carry many pieces of equipment, such as helmets, gas masks, oxygen tanks, thermal cameras, and Person Protective Equipment (PPE), that collectively can be as heavy as 70 lbs;

7. low-cost: fire departments have a limited budget to maintain and update utilities and equipment.

These requirements not only eliminate the possibility of using Global Navigation Satellite Systems (GNSS), which has degraded localization performance or fails in indoor environments or urban canyons but also exclude many existing indoor navigation technologies [129, 133]. These technologies include vision-based systems [58, 74, 139, 190, 140] that perform SLAM using images captured by monocular [139], stereo [140, 122], RGB-D infrared [190], thermal imaging [110], or event-based cameras [200]; Light Detection And Ranging (LiDAR) [78]; Radio Frequency (RF) signal-based positioning systems that trilaterates range measurements or fingerprints Receiver Signal Strength (RSS) using Bluetooth [34, 265], Wi-Fi [38], UWB [61], Radio Frequency IDentification (RFID) [232], or cellular signals including Long-Term Evolution (LTE) [5], Fifth Generation (5G) [6], Low Earth Orbit (LEO) satellite [17]. These technologies may be used opportunistically in firefighting scenarios. However, it can be dangerous to assume that sensors and underlying assumptions

behind the operations of these technologies are always valid [47]. Under these circumstances, pedestrian INS based on inertial sensing technology are the reasonable option, as the systems operate in a self-contained manner, require no installation time, and produce consistently available measurements [9].

1.2 Background

1.2.1 A Brief History of Pedestrian Inertial Navigation Systems

An inertial navigation approach can be traced back to the mid−20th century when the system first became operational [195] and was used in military operations for navigation of aircraft, missiles, and ships. Although INS was demonstrated with effective navigation performance in these applications, inertial sensors of the 20th century were large and expensive [196]. As a result, it was not practical to adopt this type of navigation solution for personal navigation applications.

Thanks to the successful development of Micro-Electro-Mechanical-System (MEMS) technology in the past two decades, low-cost IMUs with small form factors have become broadly available. The miniaturization of IMUs has allowed for attaching the sensors to a person and performing strapdown INS [198]. However, due to noise and stochastic time-varying biases in IMU measurements, navigation errors build up in each dead reckoning step. This property renders the system to have errors in position, velocity, and orientation estimates accumulating very quickly when a miniaturized IMU is used because the sensor had degraded noise performance, as compared to its macro-scale counterpart. Table 1.1 presents noise characteristics of different grades of IMUs. In the case of a Commercial-Off-The-Shelf (COTS) consumer-grade IMU, position errors of the INS can exceed one meter within just a few seconds [69]. Therefore, standalone INS does not have sufficient accuracy for firefight local-

ization.

Table 1.1: Bias characteristics of different grades of IMUs [212].

IMU grade	**Bias Instability**		**Typical Application**
	Accelerometer [mg]	Gyroscope [°/h]	
Consumer	>50	>100	Consumer electronics
Industrial	1∼50	10∼100	Automotive industry
Tactical	0.5∼1	0.1∼10	Short-term navigation
Navigation	<0.05	0.01	Aeronautics navigation

1.2.2 Why Foot-mounted IMUs?

Pedestrian INS has advantages over a standalone INS for the ability to exploit, whether directly or as an enhancement technique, local bio-mechanical information at parts of a human body where IMUs are mounted or attached [233]. Previous studies have investigated different implementations of pedestrian INS using IMUs attached to a helmet [22, 40], placed in a pocket [138, 104, 8], and mounted on a foot [48, 67]. The systems using helmet- or pocket-mounted IMUs perform Step and Headings Systems (SHSs) or Deep-Inertial Odometry (DIO), which calculate a series of vectors including information regarding step length and step heading for each gait cycle and sum up the vectors to track 2D position when steps are detected [75]. Figure 1.2 illustrates the differences in solutions produced by SHSs and INS-base approaches. In the SHSs, step lengths are estimated based on an assumption that when a pedestrian walks naturally, there is a high correlation between walking pace and step length. However, this assumption might not be valid in firefighting scenarios, where firefighters perform various activities, causing inaccuracies in step estimation and increasing navigation errors. In DIO, position tracking is formulated as a learning problem. This approach trains deep neural networks that learn pedestrian locations from raw IMU data [33]. In the corresponding training processes, a very large dataset that consists of position measurements involving extensive maneuvers is collected, and the measurements are usually

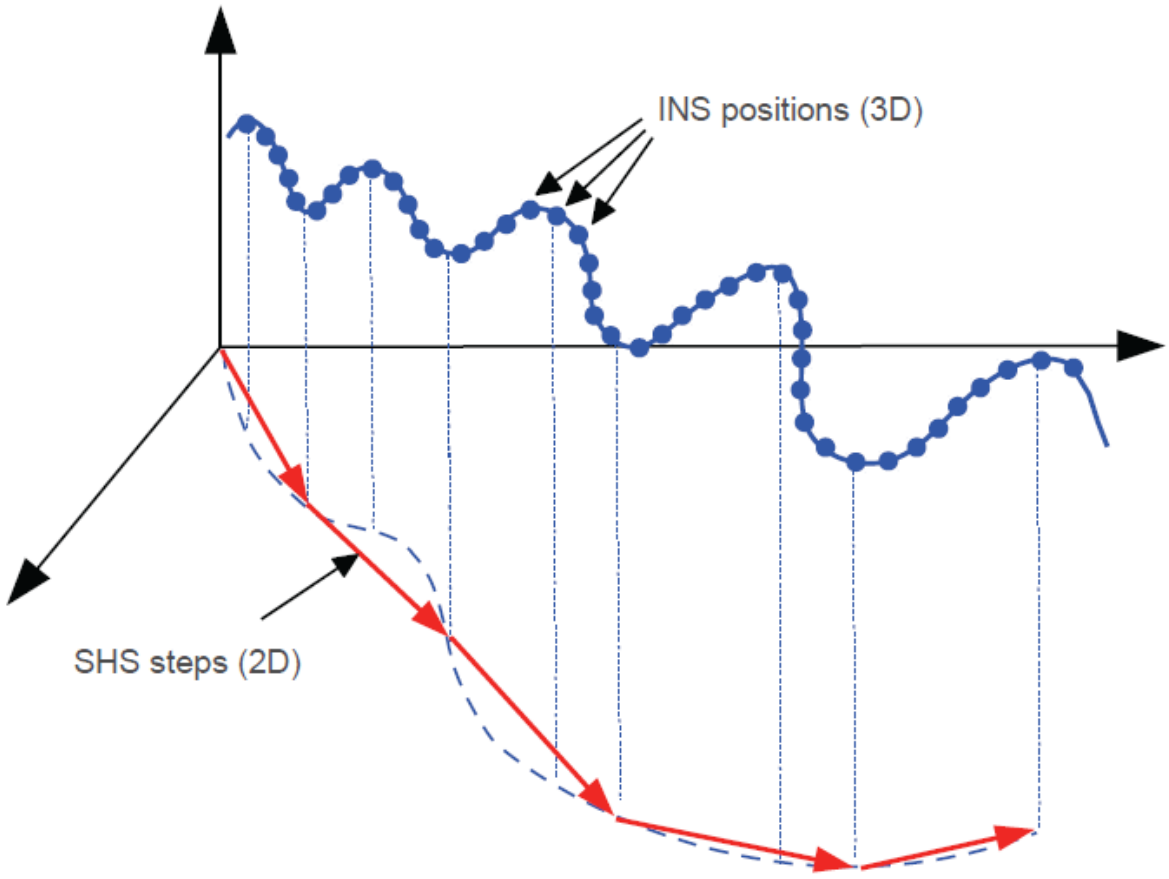


Figure 1.2: INS and SHS. An INS computes the full trajectory of a unit in 3D, represented with solid line with position dots, whilst an SHS deals only with gross step vectors in 2D, marked with arrow sequence [75].

obtained from other non-inertial sensing modalities, such as motion cameras [215]. Among these IMU placements, the systems using foot-mounted IMUs attract great attention for firefighter localization due to their unique ability can significantly enhance a strapdown INS with a ZUPT algorithm.

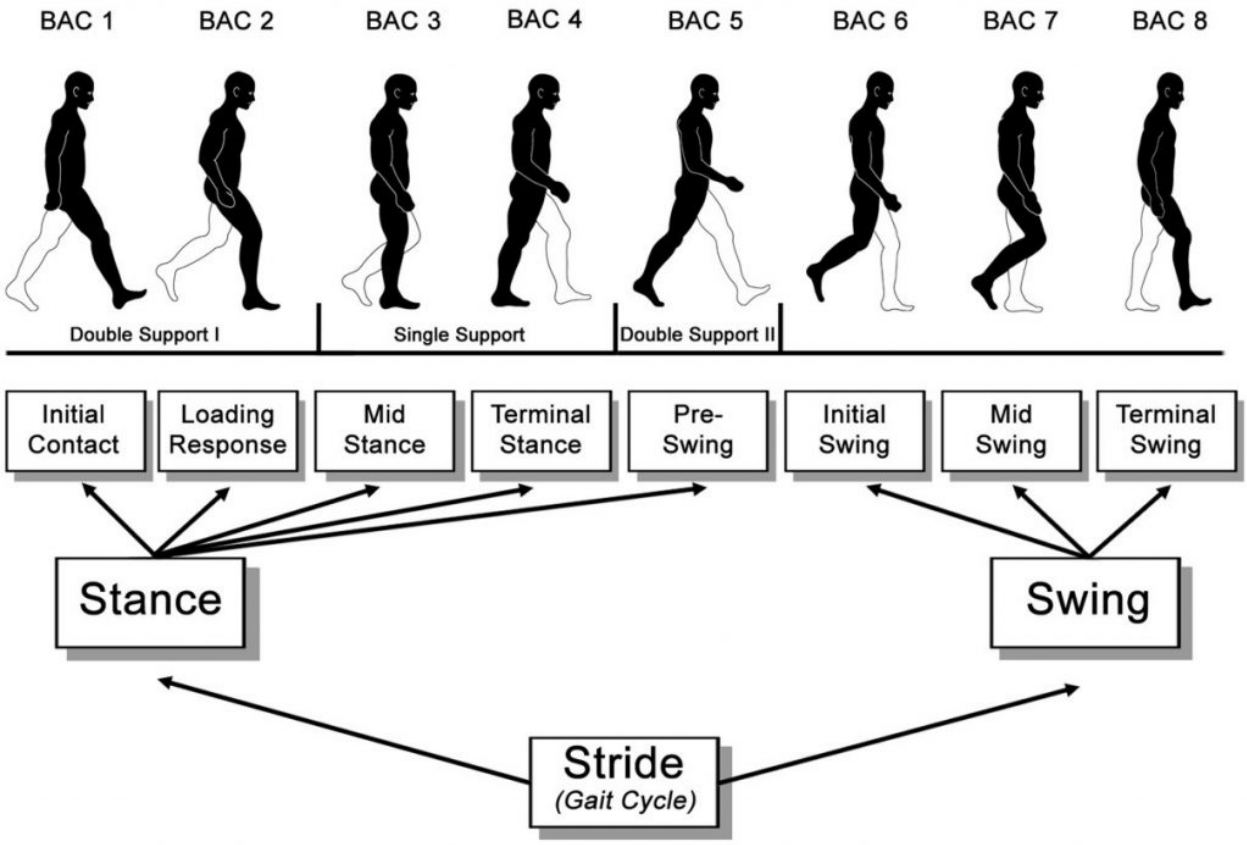


Figure 1.3: Illustration of a human gait cycle [81].

1.2.3 Zero-velocity Update Algorithm

ZUPT algorithm is based on observations of patterns in a human gait cycle, which is divided into the stance phase and the swing phase, as shown in Figure 1.3. The algorithm assumes that during the stance phase, velocities of a human foot are very close to zero and periodically resets velocity estimates to zero during this period [95]. In the early 2000s, the ZUPT-aided INS using a foot-mounted IMU was implemented by simply resetting the velocity

estimates produced by an INS to zero when a stance phase is detected [189]. Although this implementation improved navigation accuracy, it did not have a reliable navigation accuracy due to being very sensitive to the performance of stance phase detection, which has been a challenging problem even until now. In 2005, researchers implemented the ZUPT-aided INS in an EKF framework, where the propagation step is the INS, and the update step feedbacks pseudo-zero-velocity measurements to compensate for residual velocities of the INS when a stance phase is detected [57]. The implementation with EKF has a significant advantage over the one simply resetting the velocity to zero because it was mathematically proven that the EKF has not only bounded velocity errors but also orientation errors along the roll and pitch axes. As compared to a standalone INS, the ZUPT-aided INS significantly improved the navigation accuracy and duration and has been experimentally demonstrated to have a navigation error of less than 1% of traveling distances with a consumer-grade IMU, which allowed for maintaining a navigation error of less than 1 m for a few minutes [141].

1.2.4 Traditional ZUPT-aided INS

The ZUPT-aided INS is implemented in EKF. This subsection presents the details of strapdown INS, EKF, and stance phase detector.

Strapdown Inertial Navigation Systems

The strapdown INS performs localization by dead reckoning based on IMU measurements. Readouts of COTS IMUs are usually expressed as linear accelerations and angular velocity in the body frame (b-frame), which is an orthogonal axis set aligned with the roll, pitch, and yaw axes of the sensor. The strapdown INS can be derived in different reference coordinate systems, including the inertial frame (i-frame), which has its origin at the center of the Earth and axes which are non-rotating with respect to the fixed stars; the Earth frame (e-frame),

which has its origin a the center of the Earth axes that are fixed with respect to the Earth; and the navigation frame (n-frame), which is a local geographic frame having its origin at the location of the navigation system and axes aligned with the direction of north, east, and down (vertical). Pedestrian navigation focuses on localizing people in the local n-frame. The n-frame mechanization is presented as follows:

$$\mathbf{a}^n = \mathbf{C}_b^n \mathbf{f}^b - [2\boldsymbol{\omega}_{ie}^n + \boldsymbol{\omega}_{en}^n] \times \mathbf{v} + \mathbf{g}_l^n \tag{1.1}$$

$$\dot{\mathbf{C}}_b^n = \mathbf{C}_b^n \boldsymbol{\Omega}_{nb}^b \tag{1.2}$$

$$\boldsymbol{\omega}_{nb}^b = \boldsymbol{\omega}_{ib}^b - \mathbf{C}_n^b [\boldsymbol{\omega}_{ie}^n + \boldsymbol{\omega}_{en}^n] \tag{1.3}$$

$$\mathbf{g}_l^n = \mathbf{g} - \boldsymbol{\omega}_{ie}^n \times [\boldsymbol{\omega}_{ie}^n \times \mathbf{p}] \tag{1.4}$$

Here, $\mathbf{a}^n$ is the linear acceleration in the n-frame generated by navigation motion, $\mathbf{f}^b$ is the 3-axis accelerometer readouts, $\boldsymbol{\omega}_{ib}^b$ is the 3-axis gyroscope readouts, $\boldsymbol{\Omega}_{nb}^b$ is skew-symmetric matrix corresponding to $\boldsymbol{\omega}_{ib}^b$, $\boldsymbol{\omega}_{ie}^n$ is the Earth rotation rate expressed in the n-frame, $\boldsymbol{\omega}_{en}^n$ is the transport rate of n-frame with respect to the e-frame, $\dot{\mathbf{v}}$ is the acceleration generated by motion of navigation in the n-frame, $\mathbf{v}$ is the velocity estimates in the n-frame, $\mathbf{p}$ is the position estimates in the n-frame, $\mathbf{C}_b^n$ is the Direction Cosine Matrix (DCM) for transformation from b-frame to n-frame, and $\mathbf{g}$ is the unit gravity vector expressed in the n-frame.

To implement the n-frame mechanisation in a computer, (1.1)-(1.4) need to be discretized. In this section, the discretization uses 1st-order Taylor Series Expansion, presented as follows:

$$\mathbf{a}(k+1) = \mathbf{C}_b^n(k+1)\mathbf{f}^b(k) - [2\boldsymbol{\omega}_{ie}^n(k) + \boldsymbol{\omega}_{en}^n(k)] \times \mathbf{v}(k) + \mathbf{g}_l^n(k+1)$$

$$\mathbf{C}_b^n(k+1) = \mathbf{C}_b^n(k)[\boldsymbol{\Omega}_{nb}^b(k+1) + \mathbf{I}_{3\times 3}]$$

$$\boldsymbol{\omega}_{nb}^b(k+1) = \boldsymbol{\omega}_{ib}^b(k) - \mathbf{C}_n^b(k)[\boldsymbol{\omega}_{ie}^n(k) + \boldsymbol{\omega}_{en}^n(k)]$$

$$\mathbf{g}_l^n(k+1) = \mathbf{g} - \boldsymbol{\omega}_{ie}^n(k) \times [\boldsymbol{\omega}_{ie}^n(k) \times \mathbf{p}(k)]$$

Velocity $\mathbf{v}$ and position $\mathbf{p}$ in the n-frame are calculated as follows:

$$\mathbf{v}(k+1) = \mathbf{a}(k+1)\Delta t + \mathbf{v}(k)$$
$$\mathbf{p}(k+1) = \mathbf{v}(k+1)\Delta t + \mathbf{p}(k).$$

Stance Phase Detection

The stance phase detection in the proposed Ultimate Foot-mounted Inertial Navigation System (uFINS) will be achieved with a combination of IMU and ultrasonic measurements. The detector determines a stance phase if

$$T(z_n) = \frac{1}{N}\sum_{k\in\Omega_n}(\frac{1}{\sigma_\alpha^2} \parallel y_k^\alpha - g\frac{\bar{y}_k^\alpha}{\parallel \bar{y}_k^\alpha \parallel} \parallel^2 + \frac{1}{\sigma_\omega^2} \parallel y_k^\omega \parallel^2 + \frac{1}{\sigma_h^2} \parallel y_k^h - h \parallel^2) < \gamma,$$

where $\Omega_n = l \in \mathbb{N}, n \leq l \leq N-1$ is a collection of the IMU measurement indexes at time n with a window of length N, $z_n = \{[y_k^{\alpha\top}, y_k^{\omega\top}]\}_{k=N}^{k=N-1}$ is a sequence of the IMU measurements in the window, $y_k^{\omega\top}$ are the gyroscope measurements at k, σ_a^2 is the noise variance of the accelerometer, σ_ω^2 is the noise variance of the gyroscope, σ_h^2 is the noise variance of the ultrasonic sensor, h is the height of the ultrasonic sensor with respect to the ground, and γ are user-defined thresholds.

Extended Kalman Filter

The EKF state $\mathbf{x}$ is a 15×1 vector, described as follows:

$$\mathbf{x} = [\mathbf{q}, \mathbf{v}, \mathbf{p}, \mathbf{b}_a, \mathbf{b}_g],$$

where $\mathbf{q}$ the attitudes of the IMU mounted expressed in navigation frame, respectively. $\mathbf{b}_a$ and $\mathbf{b}_g$ are the bias of the accelerometers and gyroscopes.

In each implementation iteration, the EKF first performs a prediction step, followed by an update step. In this subsection, $\mathbf{x}_k$ and $\hat{\mathbf{x}}_k$ denote estimated state in the update step and the prediction step at time k, and $\mathbf{P}_k$ and $\hat{\mathbf{P}}_k$ denote estimated state in the update step and the prediction step at time k. The prediction step is described as follows:

$$\hat{\mathbf{x}}_{k+1} = \mathrm{INS}(\mathbf{x}_k)$$
$$\hat{\mathbf{P}}_{k+1} = \mathbf{F}_k \mathbf{P}_k \mathbf{F}_k^\top + \mathbf{Q}_k.$$

The update step is described as follows:

$$\mathbf{x}_{k+1} = \hat{\mathbf{x}}_{k+1} + \hat{\mathbf{P}}_{k+1}\mathbf{H}_k^\top [\mathbf{H}_k \hat{\mathbf{P}}_{k+1} \mathbf{H}_k^\top + \mathbf{R}_k]^{-1} [\mathbf{z}_k - \mathbf{H}_k \hat{\mathbf{x}}_{k+1}]$$
$$\mathbf{P}_{k+1} = [\mathbf{I} - \hat{\mathbf{P}}_{k+1}\mathbf{H}_k^\top [\mathbf{H}_k \hat{\mathbf{P}}_{k+1} \mathbf{H}_k^\top + \mathbf{R}_k]^{-1} \mathbf{H}_k] \hat{\mathbf{P}}_{k+1}.$$

The matrices $\mathbf{F}_k$, $\mathbf{Q}_k$, $\mathbf{H}_k$, $\mathbf{R}_k$ are expressed as follows:

$$\mathbf{F}_k = \exp(\mathbf{A}_k \Delta t + \mathbf{I}), \text{where } \mathbf{A}_k = \begin{bmatrix} \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & -\mathbf{C}(\mathbf{q}) & \mathbf{0}_{3\times3} \\ \overrightarrow{f}^n \times & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{C}(\mathbf{q}) \\ \mathbf{0}_{3\times3} & \mathbf{I}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \end{bmatrix},$$

$$\mathbf{Q}_k = \begin{bmatrix} \boldsymbol{\sigma}_{\mathrm{ARW}} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \boldsymbol{\sigma}_{\mathrm{VRW}} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \boldsymbol{\sigma}_{\mathrm{RRW}} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \boldsymbol{\sigma}_{\mathrm{AcRW}} \end{bmatrix},$$

$$\mathbf{H}_k = \begin{bmatrix} \mathbf{0}_{3\times3} & \mathbf{I}_{3\times3} & \mathbf{0}_{3\times9} \end{bmatrix}.$$

where $\overrightarrow{f}^n \times$ is the skew-symmetric cross-product-operator of the accelerometer outputs of

the IMU, expressed in the navigation frame, $\boldsymbol{\sigma}_{\text{ARW}}$ is Angular Random Walk (ARW) of the gyroscopes, $\boldsymbol{\sigma}_{\text{VRW}}$ is the Velocity Random Walk (VRW) of the accelerometers, $\boldsymbol{\sigma}_{\text{RRW}}$ is Rate Random Walk (RRW) of the gyroscope, and $\boldsymbol{\sigma}_{\text{AcRW}}$ is the Acceleration Random Walk (AcRW) of the accelerometers. $\mathbf{C}(\mathbf{q})$ is the DCM corresponding to the quaternion $\mathbf{q}$.

1.3 Problem Statement

Although ZUPT-aided INS can achieve a localization error on the order of 1% of the total trajectory length, this level of accuracy might not be sufficient for firefighters and first responders as they may need to travel for hundreds or thousands of meters in GNSS-degraded environments and maintain the accuracy of their location on the level of 1 meter. To extend the usage of the ZUPT-aided INS, improvements are needed on both the sensor-level and the algorithm-level. For MEMS-based IMUs, sensor level development includes innovative design of micro-mechanical structures and corresponding control circuits that lead to better noise characteristics in terms of ARW, VRW, RRW, and AcRW, and larger sensor FSR and bandwidth. On the algorithmic and system level, it has been identified that accuracy and reliability of the ZUPT-aided INS are affected by multiple error sources, including inconsistent stance phase detection accuracy, weak observability of yaw angle estimates, EKF systematic errors, ambient temperature variations, insufficient IMU FSR and bandwidth, and unbounded position error growth [205].

This thesis focuses on the issues of the ZUPT-aided INS on algorithmic and system level and aims to develop approaches to address each of the error sources. These error sources can be grouped into three categories: on motion sensors, on algorithm assumptions, and on estimation filters. The three categories are discussed in detail in the following paragraphs.

1.3.1 Challenges on Foot-mounted Sensors

Ambient Temperature Variations

A majority of MEMS IMUs are silicon-based devices, which have performance sensitive to ambient operating temperatures. Variations in ambient temperatures significantly impact the fundamental characteristics of these inertial sensors as well as their control circuits [16]. This impact leads to bias drifts and increased noise Standard Deviation (SD) in IMU measurements. In such cases, the accuracy and reliability of a traditional ZUPT-aided INS are degraded because the unmodeled bias drifts and the increased noise SD would cause large errors in the unobservable yaw angle state, affect detection accuracy of a traditional zero-velocity detector involved in the ZUPT algorithm, and generate filter inconsistencies in the EKF. The degraded performance of the ZUPT-aided INS may occur in firefighting scenarios, as it is conceivable that temperatures around firefighters can change as much as $20 - 30°\text{C}$ in a short time span [79].

Insufficient IMU FSR and Bandwidth

Insufficient sensor FSR has been reported as an error source in foot-mounted INS [205]. Although mounting IMUs on footwear brings the benefits of using the ZUPT algorithm, foot-mounted sensors experience significantly larger forces than the sensors mounted on other parts of a body. Forces as large as 40 g can occur during the heel-strike phase of a gait cycle and saturate measurements obtained from many COTS high-performance IMUs, which have an excellent noise performance but often limited sensor FSR and were often used in previous research work regarding foot-mounted IMUs. Table 1.2 summarizes FSR and -3 dB bandwidth of IMUs used in the 82 selected publications related to foot-mounted INS. Figure 1.4 presents of accelerometer and gyroscope FSR and bandwidth of 42 different IMUs used

in the selected 82 publications related to foot-mounted INS. It could be observed in Figure 1.4 that many of the chosen IMUs might not have sufficient sensor FSR and bandwidth to fully reconstruct the violent inertial forces experienced by sensors during walking and running. Although there are COTS IMUs with accelerometer's FSR larger than 200 g, which might be sufficient for foot-mounted INS, these sensors are not preferable for inertial navigation because they usually have an order-of-magnitude worse noise performance. With high-performance IMUs, the saturation indicates that foot-mounted IMU could produce incorrect readings of actual accelerations during the take-off phase and the heel-strike phase, which leads to degraded navigation accuracy and needs to be taken into account in the ZUPT-aided INS [44]. The trade-off between sensor's noise performance, FSR, bandwidth, and Size, Weight, Power, and Cost (SWaP+C) remained to be unresolved challenges in development of sensors for pedestrian inertial navigation.

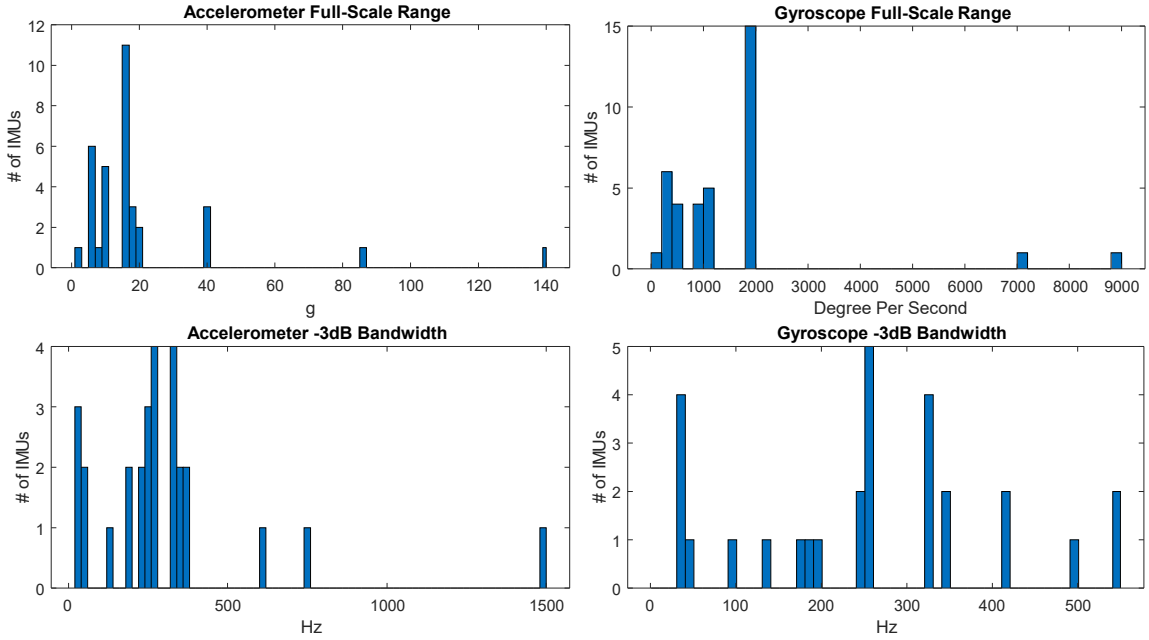


Figure 1.4: Histograms of accelerometer and gyroscope FSR and bandwidth of 42 different IMUs used in the selected 82 publications related to foot-mounted INS.

Table 1.2: FSR and −3 dB bandwidth of inertial sensors used in selected 82 publications related to foot-mounted IMUs.

			Gyro		Accel		
Year	**Brand**	**Model**	FSR (°/s)	BW (Hz)	FSR (g)	BW (Hz)	**Publication**
2003	Analog Devices	ADXL	NA	NA	10	NA	[189]
2009		ADXRS300	300	40	NA	NA	[254]
2010		ADIS16355	300	350	10	350	[181, 186, 65, 267, 145]
2011		ADXL345	NA	NA	16	NA	[80, 261]
2012		ADIS16367	1200	330	18	330	[144, 251]
2013		ADIS16354AMLZ	300	350	1.7	350	[160]
2015		ADIS16488	480	330	18	330	[240]
2016		ADIS16480	480	330	10	330	[148]
2019		ADIS16485	480	330	5	330	[213]
2022		ADIS16470	2000	550	40	600	[99]
2022		ADIS16497−3	2000	550	40	750	[1]
2007	BAE	SILMU02	9000	135	140	130	[149]
2006	Cloud Cap	Crista	300	100	10	50	[66, 67, 21]
2003	Honeywell	GG1308	1000	NA	NA	NA	[189]
2011		HG1930	7200	NA	85	NA	[117]
2017	Inertial Element	MIMU22BTP	2000	256	16	260	[203]
2005	Intersense	InertiaCube3	1200	NA	NA	NA	[57]
2011		NavChip	2000	NA	16	NA	[44, 124, 267]
2011	Invensense	ITG−3200	2000	NA	NA	NA	[255, 261]
2011		IMU−3000	2000	NA	NA	NA	[208, 80]
2014		MPU−9150	2000	256	16	260	[141, 146, 221, 128]
2017		MPU−9250	2000	250	16	260	[194, 41, 130, 131]
2020		MPU−6050	2000	256	16	260	[258, 11]
2022		ICM−20948	2000	197	16	246	[37]
2009	Kionix	KXM52	NA	NA	6	1500	[254]
2009	Memsense	nIMU	1200	50	NA	NA	[24]
2011	MicroStrain	3DM−GX2	1200	NA	10	NA	[206, 23]
2016		3DM−GX4−45	900	250	16	225	[156]
2021		3DM−CV5−25	1000	500	40	225	[120]
2020	Starneto	IMU	120	NA	6	NA	[231]
2011	STMicro-electronics	LIS3LV02	NA	NA	6	40	[255]
2018	VectorNav	VN−100	2000	260	16	256	[202]
2018		VN−200	2000	260	16	256	[223, 216, 222, 218, 214]
2015	x-io Technologies	x-IMU	2000	NA	8	NA	[208, 123, 220]
2007	Xsens	MTx	300	40	5	30	[152, 256, 164, 101, 102, 168]
2009		MTi	300	40	5	30	[95, 96, 176, 158, 175, 176, 252, 45]
2010		MTx−28A53G25	1200	40	18	30	[15]
2017		MTi−100	450	415	20	375	[257, 201, 151]
2016		MTi−G710	1000	415	20	375	[210, 197, 125]
2018		MTw	2000	184	16	184	[105]
2020		MTi−3−8A7G6−DK	2000	180	16	180	[204]

1.3.2 Challenges on Algorithm Assumptions

Inconsistent Stance Phase Detection Accuracy

Navigation errors of the ZUPT-aided INS highly depend on the accuracy of stance phase detection. A misdetection is defined as the case where the detector fails to determine a stance phase when a foot-mounted IMU is stationary. Misdetection can leave velocity estimates not being corrected for an extended period of time, propagating polynomially to accumulated position errors. A false alarm is referred to as the case where the foot is moving while the detector indicates a stance phase. False alarms can lead to resetting velocity to zero while a pedestrian's foot is traveling very fast, violating the assumption of the ZUPT algorithm and greatly degrading the navigation accuracy [181]. In a traditional implementation of the ZUPT-aided INS, stance phase detection is based purely on IMU measurements. However, this mechanism does not have a high detection accuracy because the measurement patterns of the stance phases and the swing phases are very different when a pedestrian is performing different activities.

EKF Systematic Errors

In ideal scenarios, the innovation sequences of the EKF are zero-mean Gaussian sequences. However, the innovation sequence of the ZUPT-aided INS has been reported to have between-step correlations and non-zero means [143]. The non-zero mean phenomenon is caused by the violation of the assumption that the velocity of a foot in the stance phase is completely stationary because the completely stationary scenario almost never happens in practice when a pedestrian is performing daily activities, such as walking, running, climbing stairs, and crawling. The violation of the assumption misleads the ZUPT algorithm to over-confidently correct velocity states frequently with the measurement covariance matrix that has low-value

entrees. The modeling errors accumulate at each step, decreasing the navigation accuracy of the ZUPT-aided INS [157].

1.3.3 Challenges on Estimation Filter

Weak Observability of Yaw Angle Estimates

While it has been proven that the ZUPT-aided INS implemented in the EKF had bounded errors for velocity states along the three axes of translation and orientation states along the roll and pitch axes, yaw angle errors grow unboundedly as the state are not observable in the EKF. The unbounded yaw angle errors lead to accumulated horizontal position errors, limiting the usage of the foot-mounted INS [96].

Unbounded Position Error Growth

In addition to the yaw angle state being weakly observable, position states of the ZUPT-aided INS along the three axes of translation also have errors that grow unboundedly. The horizontal position errors are coupled with yaw angle errors. The vertical position errors are enhanced due to that COTS IMU FSR and bandwidth are not insufficient to measure high shocks generated by foot-striking the ground. Once error sources are introduced to the system, accuracy of estimated positions would significantly decrease and the errors would propagate to subsequent position estimates, lowering the reliability of the ZUPT-aided INS.

To realize an accurate and reliable pedestrian navigation system based on ZUPT-aided INS, the discussed six error sources are needed to be addressed. Although tremendous research efforts and progress were shown in the literature, which will be discussed in Chapter 1.4, that was devoted to independently mitigating the impacts of the six errors on navigation performance, the problems caused by these error sources in ZUPT-aided INS remains unsolved.

This thesis intends to develop algorithmic advancements that address the problems.

1.4 Literature Review

This chapter of the dissertation discusses previous work on approaches to improve the ZUPT-aided INS using foot-mounted IMUs. The approaches are grouped into solutions that solve the different aspects of the system described in Chapter 1.3. Each group is presented as a section in this chapter.

1.4.1 Enhancements On Motion Sensor

Compensation for Insufficient FSR and Bandwidth

In order to address the problem of insufficient inertial sensor FSR, previous works have attempted different approaches. In [143] and [214], shock-absorbing pads were attached between an IMU and a shoe. The pad could reduce the magnitude of forces experienced by the IMU to a certain extent, but the forces could still be larger than accelerometer's FSR of many COTS IMUs. In [101], the research group modified the ZUPT-aided INS algorithm by applying zero position change during heel strike phases and adjusting estimation error covariance matrices. Their algorithm was designed based on the assumption that displacements of an IMU mounted on the heel side of a shoe are minimal during heel strike phases. Based on our observation in pedestrian navigation experiments, we considered that the assumption might not be realistic in some scenarios, such as running, where the foot-mounted IMUs can be very unstable during the heel-strike phases. In [100], an additional IMU mounted on the calf of a human was used to assist a foot-mounted IMU. The usage of the calf-mounted IMU was shown to improve navigation accuracy but increased the hardware

complexity of a pedestrian navigation system.

To mitigate errors induced by insufficient bandwidth of inertial sensors in ZUPT-aided INS, the errors can be modeled as a function of both sensor bandwidth and experienced velocities [120], and the error model was used in the EKF of the ZUPT-aided INS.

Temperature Compensation

MEMS-based IMUs are well-understood to have measurements that are affected by temperature-dependent errors [46]. Approaches reported in the literature for mitigating temperature-induced errors can be categorized into temperature control mechanisms and temperature compensation approaches. Temperature control mechanisms integrate an additional temperature controller co-located with the IMU to stabilize the ambient temperatures [247]. Temperature compensation approaches approximate a relation between sensor thermal-induced biases and thermal-related variables with a mathematical model [147]. The mathematical model is then used to predict the biases, and the predicted biases are removed from raw sensor measurements. Among these two groups of approaches, temperature control mechanisms have been demonstrated to more effectively minimize temperature-induced errors in MEMS-IMUs, but come with a trade-off of increasing the size and cost of an integrated sensor system.

This section focuses on reviewing literature regarding temperature compensation methods. MEMS-IMUs as on silicon-based devices were analyzed to have a linear relationship between sensor thermal-induced biases and thermal-related variables [42]. Since temperature variations also impact control circuits and may cause distortion on corresponding Printed Circuit Boards (PCBs), the relationship was demonstrated with nonlinearity [237] in experiments where thermal variations were gentle. To account for the nonlinearity, previous works investigated different mathematical models, including different orders of poly-

nomials [147, 192, 245] and configurations of Back-Propagation Neural Networks (BPNNs) [237, 209, 241, 53, 54, 56, 55, 16, 28, 228]. However, in experiments where temperature variations were significant, the temperature-bias relation was demonstrated with a hysteresis effect [159]. The observations of hysteresis effects indicated that sensor biases might be a nonlinear function of more than one input of temperature and led researchers to investigate different input features of the mathematical models. Therefore, in addition to temperature readings, temperature rate, temperature gradient, or IMU readings were also considered as inputs to the mathematical model that aims to predict thermal-induced errors [192, 241, 54, 245, 16, 228]. Although these methods have demonstrated to effectively mitigate the errors in IMU readings, it is still unclear to what extent can these approaches improve position accuracy of foot-mounted INS when operating in temperature-varying environments.

1.4.2 Enhancements On Algorithm Assumption

Robust Stance Phase Detection

In the early development of the ZUPT-aided INS algorithm, researchers have started to use a foot-mounted IMU for stance phase detection [66, 149]. In 2010, the detection problem was formalized as a binary hypothesis testing derived in a General Likelihood Ratio Test (GLRT) framework [181]. In this framework, the two hypotheses correspond to the cases of the stance phase and the swing phase in a gait cycle, respectively. Using this approach, four detectors were derived, which are the Stance Hypothesis Optimal dEtection (SHOE), acceleration-Moving Variance (MV), acceleration-MAGnitude (MAG), and Angular Rate Energy (ARE). In these mechanisms, the detection is achieved by comparing a pre-defined threshold with a metrics called statistics, which can be expressed in terms of the ratio of likelihoods computed from sensor measurements acquired during the two hypotheses cases. The statistics of these

detectors directly relates to the stability of the IMU, and the detection is based on the fact that the foot is more stable in the stance phase than in the swing phase. These detectors compare the statistics with a pre-defined threshold and detect a stance phase if the statistics are lower than the threshold.

Using a fixed threshold for determining the gait phases for these detectors is, however, challenging because the stability of a person's foot during the stances varies significantly and depends on the type of activities. A detection threshold that is optimal for normal walking might be too low for running [223], and vice versa, resulting in misdetection or false alarms.

To improve the detection accuracy, many techniques have been developed to enhance the detectors derived in the GLRT framework. These techniques include using adaptive thresholds based on 1) additional assumptions of pedestrian gait models, such as the maximum shock of a foot-mounted IMU within a gait is different while walking and running [223] and the probability of occurrence of two consecutive stance phases during a short period is low [206], magnitudes of normalized accelerometer signals are different for various motions [121], a combination of accelerometers and gyroscopes is beneficial for walking and standalone gyroscope measurements is better for running [124], and sliding windows [194], 2) activities classified using data-driven approaches, such as Support Vector Machine (SVM) algorithm [203, 152], Hidden Markov Model (HMM) [191], fuzzy logic reasoning [210], Long Short-Term Memory (LSTM) [202, 41], and Convolutional Neural Network (CNN) [39], 3) additional information extracted from the foot-mounted IMU, such as a chest-mounted accelerometer [256], the temporal variance of accelerometers [197, 127], and 4) sensor fusion with additional non-inertial sensing modalities, which include magnetometers [148], ElectroMyoGraphy (EMG) [211], shoe-embedded pressure sensors [130, 131], and downward-facing RF range sensors[260, 255].

Non Zero Velocity Update

To mitigate the impact of the observation that the velocity of the foot is, although very close, not zero, previous approaches have considered applying non-zero-velocity updates during the detected stance phase. The non-zero-velocity measurements can be derived from foot kinematics [234, 102] or obtained from additional sensors, such as an array of magnetometers [182], Alternate Current (AC) electromagnetic sensors [222, 218], or cameras [15, 157].

1.4.3 Enhancements On Estimation Filter

Magnetometer Compass

Based on measurements collected with calibrated 3-axis magnetometer, the heading angle can be estimated. Multiple previous research works have reported utilizing a magnetometer as a compass to provide additional measurements of heading angle and bound yaw angle error growth [257, 125, 208, 23]. However, modern indoor environments are filled with conductive materials and metals, which generate magnetic field disturbance, including hard- and soft-iron effects, and significantly degrade heading estimation accuracy. To reduce hard-iron and soft-iron interference in magnetometers, physically rotating the sensors at every operating location is needed in magnetometer calibration approaches [235]. These approaches are not realistic in a real-time pedestrian navigation system. Without calibration, the estimated heading angle can have errors of up to 100° in indoor environments [7], which are much larger than the errors generated by integrating gyroscope measurements in a short- to mid-term navigation mission.

Zero Angular Rate Update

Zero Angular Rate Update (ZARU) algorithm feedbacks pseudo measurements of zero angular rate update when a stance phase is detected [96]. This algorithm allows for correcting EKF states of gyroscope biases, reducing heading angle drift. However, it has been reported that to effectively use the ZARU algorithm, the measurements of zero angular rate update should be used only when the foot-mounted IMU is completely stationary [143]. Otherwise, the actual measurements of angular rate generated by subtle motions of the foot may be incorrectly treated as gyroscope biases, leading to significantly degraded yaw angle accuracy.

Multi-IMU Platform

In the ZUPT-aided INS, ARW is an important gyroscope measurement characteristic that affects how fast the yaw angle state drifts [216]. Reducing the ARW can directly improve yaw angle estimation accuracy. To improve the ARW with COTS IMUs, a Multi-IMU (MIMU) platform, which acquires measurements from multiple IMUs simultaneously via parallel communication protocols, can be used [183, 142, 184]. The MIMU platform allows for averaging out independent stochastic errors in low-cost IMUs.

Heuristic Heading Compensation

Heuristic Heading Compensation (HHR) is an approach to enhance the ZUPT-aided INS with an additional assumption that when pedestrians walk in indoor environments, the trajectories consist of segments of straight lines [24]. The HHR algorithm detects if a person is walking a straight line and corrects the yaw angle state by feeding back measurements of zero yaw angle changes between two or multiple consecutive steps during the stance phase [96]. A variation of the HHR approach was to integrate a side-facing laser range sensor with

a foot-mounted IMU [158]. The laser range sensor measures the distance to walls. Assuming that walls are vertical flat planes, this approach simultaneously maps position of walls and updates position and heading angle states of the sensor based on the range measurements.

Foot-SLAM

Implementing a SLAM algorithm based on foot-mounted INS, which is referred to as the Foot-SLAM algorithm, has also been demonstrated to reduce errors in the yaw angle state, leading to more reliability in long-term position stability. In Foot-SLAM algorithm, the ZUPT-aided INS was implemented in a particle filter framework with probabilistic transition maps, based on the probability of the pedestrian crossing transitions in a regular 2D grid of adjacent hexagons of a predefined radius [171, 14, 163, 164, 63]. The navigation performance of this approach could be enhanced by considering cases of indoor moving platforms, such as elevators and escalators [105], using an adaptive threshold in the built-in zero-velocity detection to account for non-walking cases [204], combining the occupancy maps obtained from multiple agents [173, 162, 161], incorporating prior knowledge of the building layout as prior maps [106], and integrating a few known locations [103].

Foot-to-foot Augmentation

An enhancement mechanism to augment ZUPT for pedestrian INS is to employ a dual foot-mounted IMU system, which has shown to significantly improved navigation results compared to a single foot-mounted IMU and a standalone ZUPT algorithm. A system using dual IMU benefits from additional measurements derived from the relative motions between the two shoes of a pedestrian, which information will be stacked with the pseudo-measurements from ZUPT in a Kalman filter-based algorithm [213, 99]. Several different measurements derived from the relative shoe motion were reported. One branch of measurement methods

imposes statistical constraints on the maximum or minimum distance between the two shoes [186, 65, 146, 37]. These methods formulate the constraint conditions based on a single hypothetical foot motion model, which in general is different for each individual.

Another branch uses actual ranging sensors to acquire real-world foot-to-foot distance measurements. Brand et al. proposed a personal navigation system aided by foot-to-foot ranging measurement, which is the relative distance between the two shoes [26]. Their system was later tested by Laverne et al. with foot-to-foot range measurements obtained from two pairs of shoe-mounted SOund Navigation And Rangings (SONARs) [117]. Although this system has been demonstrated with excellent navigation results, their measurement model had to be linearized for use of the EKF; their model assumed the mismatch between position estimations from IMUs and distance measurements from SONARs to be small, which might not always be the case. In addition to relative distance measurements, the system proposed by Bancroft et al. also used relative positions between the two shoes as updates in the EKF [21]. Wang et al. utilized beam pattern characteristics of some COTS ultrasonic transmitters to provide foot-to-foot yaw angle compensation [213]. The system proposed by Placer et al. [160] adopted a shoe-mounted marker. They used the marker as a landmark and updated the position of the system based on the landmark when the corresponding images showed that the marker was stationary.

Magnetic-field SLAM

In indoor environments, the presence of ferromagnetic elements generate magnetic interference, which makes it very challenging to directly enhance ZUPT-aided INS with heading angles estimated from calibrated magnetometer measurements. However, magnetometers can be used to implement a SLAM algorithm based on a magnetic interference map.

Assuming magnetic field anomalies in an indoor environment are nearly static over time and

have sufficient spatial variability, localization solutions can be obtained based on distinct patterns of a magnetic fingerprint [76] with a pre-surveyed magnetic anomaly map. This approach was demonstrated to track position along the three axes of mobile robots operating in indoor environments and generate a magnetic-field fluctuation map [199, 188]. This approach was combined with foot-mounted IMU in pedestrian navigation. Previous work reported a method called MagSLAM, which produces 2D localization solution by first using foot-mounted INS to map ambient magnetic field strength, and then the map was used to improve long-term positioning stability [172]. The MagSLAM was later extended to in the case of 3D positioning and experimentally proven effective with a foot-mounted IMU, including 3-axis accelerometer, 3-axis gyroscope, and 3-axis magnetometer [201, 151].

Aiding with Signals of Opportunity

To improve localization accuracy of the foot-mounted INS for long-term navigation tasks, solutions using opportunistic exteroceptive external aiding signals have been explored. These signals include Wi-Fi, Bluetooth, RFID, UWB, and cellular signals, such as LTE and 5G.

Wi-Fi Although standalone Wi-Fi positioning is not an infrastructure-free solution, the signals are presented in many modern buildings. Localization based on Wi-Fi signals can be implemented in many different approaches, including Time of Arrival (ToA), Angle of Arrival (AoA), or hybrid of ToA and AoA, Round-Trip Time (RTT), standalone RSS, and RSS fingerprinting [246, 180]. Among these approaches, RSS-based techniques, which requires a site-surveying step, can be operational with only one anchor signal transmitter without the requirement of re-programming COTS Wi-Fi routers. These properties allow for a larger operational area and ease of deployment, making it a more attractive option for indoor localization compared to the other implementations using Wi-Fi signals.

Previous research has investigated approaches that combined the RSS fingerprinting techniques with foot-mounted IMUs. These two systems worked in a complementary manner. In [71, 73, 72, 13, 50, 35, 30], foot-mounted INS is used to both perform localization as well as constructing a map of Wi-Fi RSS. These approaches have been demonstrated to increase positioning accuracy for long-term navigation, as compared to a standalone ZUPT-aided INS. Bruno et al. enhanced the Foot-SLAM algorithm with Wi-Fi RSS map [27].

Bluetooth Indoor positioning solutions based on Bluetooth protocols require pre-installation of Bluetooth beacons. When these beacons are available in a navigation environment, localization can be performed by using corresponding range measurements or RSS fingerprints. In [266], researcher combined range and RSS measurements of Bluetooth with pedestrian INS using IMUs embedded in a hand-held smartphone and performed positioning in a SLAM framework realized with graph-based optimization.

Although many previous research works have reported the foot-mounted pedestrian navigation platform including both IMU and Bluetooth module, the Bluetooth modules were mostly used for transmitting sensor measurements [19, 20] or real-time navigation solutions [144]. Gu et al. utilized Bluetooth signals to initialize and calibrate foot-mounted INS based on iBeacons pre-deployed in a navigation environment [70].

RFID RFID systems include tags and readers. It was first demonstrated that distance measurements deduced from RSS path-loss models of known RFID tags with a standalone INS in both loosely- and tightly-coupled EKF framework could improve navigation accuracy, as compared to a standalone INS [168, 254, 170]. To further increase the navigation accuracy of RFID/INS system, foot-mounted IMU was used, allowing for implementation of a ZUPT algorithm [175, 80, 176].

UWB UWB communication protocols are mainly used to enhance pedestrian inertial navigation systems in two approaches: beacon-based approaches using distance measurements and cooperative localization.

COTS UWB modules, such as Decawave DWM1000, can be programmed as a receiver or a beacon. Distance measurements between a UWB receiver and a UWB beacon can be obtained using RTT, which does not require clock synchronization between each UWB node, allowing a flexible implementation, as compared to Time of Fly (ToF), AoA, or Time Difference of Arrival (TDoA). When UWB beacons, which are usually pre-installed in a indoor environment before a navigation task, are connected to UWBs mounted on a pedestrian's body, the corresponding range measurements can be used to enhance navigation solutions of foot-mounted IMU systems. These approaches can be implemented in different filter framework, including Kalman Filter (KF) [267], EKF [251, 123], Particle Filter (PF) [156, 174], joint state PF [252, 258, 219], quaternion KF [231], iterated EKF [221, 11], Unscented Kalman Filter (UKF) [36], predictive adaptive KF [243], federated Extended Finite Impulse Response (EFIR) Filter [244], and graph-based optimization [220]. The experiments conducted in these research mounted UWB on different parts of the body, such as head [219, 221, 123], backpack [268, 244, 243], chest [252, 36], and foot [231, 258, 267, 11].

In cooperative localization, a group of communicating agents uses inter-agent relative range measurements as feedback to improve the localization accuracy of their local navigation filters [155]. Cooperative localization is often considered an infrastructure-free enhancement technique in pedestrian navigation and can be implemented based on inter-agent range measurements obtained using different mechanisms, including computer vision or wireless RF signals [43]. Among these implementations, UWB-based range measurements have attracted significant attention in indoor navigation because the sensor has high time resolution, wide bandwidth, and capability to work under Non-Line-Of-Sight (NLOS) conditions [264]. Using enhancement with UWB-based inter-agent range measurements, ZUPT-aided INS has been

experimentally shown to increase position accuracy [150, 128, 264, 145, 261].

Cellular Signals Cellular signals have been used in an opportunistic localization framework. In this framework, cellular towers are treated as beacons, and positioning can be achieved by trilaterating pseudorange measurements based on cellular signals [108]. Cellular signals possess several desirable characteristics for indoor localization: abundance, geometric diversity, high bandwidth, high carrier-to-noise ratio (C/N_0) in indoor conditions, and the fact that some of their downlink signals are free to use [4, 3]. These localization approaches have been demonstrated to provide absolute position compensations to standalone INS [109, 132, 135]. In [1], a system integrating ZUPT-aided INS and LTE pseudorange measurements in both loosely- and tightly-coupled manners was developed.

Height Compensation

For first responders, the precision of height estimation is a crucial performance metrics. For example, an accumulated three-meters of error in the vertical direction is equivalent to an error in identification of the floor in a building [166]. The navigation errors coming from the ZUPT-based INS accumulate unboundly in the vertical direction [143]. Thus, additional sensing modalities can be effectively used along with the ZUPT-based INS to enhance the navigation accuracy, in-plane and out-of-plane. Barometric altimeters (barometers) are popular devices for this purpose as they provide independent and direct measurements of position along the vertical direction. Barometric data for INS has been shown to improve the overall navigation results in various types of integrated INS [167, 98, 187, 253, 177, 240]. However, because these devices detect changes in air pressure, their measurements are easily affected by weather changes during data acquisition, elevated pressure due to temperature increase in the event of fire, and other environmental effects [118]. Although this issue can be addressed by using one or multiple reference barometers at known locations [107, 169, 193, 154, 238],

this type of solution is not suitable for infrastructure-free navigation. Moreover, barometers are easily subjected to air pressure perturbations in local environments. For example, opening a door or a window near the sensor would lead to an incorrect estimation of height [107], and those errors could be further amplified in chaotic air pressure environments [165, 207].

An alternative sensor that can be employed to obtain information about the relative vertical position is an ultrasonic sensor [242, 59, 112, 166]. Downward-facing ultrasonic sensors can measure the distance between the sensor and the ground. In [166], for example, an ultrasonic sensor was mounted on the bottom of a backpack to detect a relative height distance. This information was fused with height measurements collected from a barometer and used in the KF framework. Such arrangement has been demonstrated to effectively handle the pressure shock during navigation and allowed for estimating the barometer bias during the transition between indoor and outdoor environments. However, the KF discussed in [166] depends highly on barometric measurements, and those can be inaccurate due to, for example, the room temperature controlled by air-conditioning or other chaotic variations under extreme operational cases. In pedestrian navigation, rich information can be extracted from the dynamics of a human walk. The mechanism introduced in [166], if used to aid ZUPT-based INS, can provide a bound for displacement error growth in the vertical direction but cannot capture subtle foot motions, which could benefit the overall navigation results of a pedestrian INS. In [45], downward-facing ultrasonic sensors were mounted on the shoe to extract measurements of the height of a shoe relative to the ground. Although the method showed an improvement of navigation accuracy in experiments of walking on a flat surface, as compared to standalone ZUPT-aided INS, it did not account for other common indoor terrains, such as stairs, ramps, and elevators.

1.5 Thesis Overview

This Ph.D. thesis develops algorithmic techniques for enhancing the current state-of-the-art implementation of ZUPT-aided INS. In this thesis, we developed eight approaches, each of which is designed specifically to address one of the three error sources: foot-mounted sensors, algorithm assumptions, and estimation filter, discussed previously in Section 1.3. The approaches have been experimentally verified to improve the navigation accuracy of pedestrian navigation systems using ZUPT-aided INS and were envisioned to be combined together to achieve an infrastructure-free pedestrian positioning solution. Such a solution has a potential for long-term accuracy and reliability not subjective to surrounding environments undergoing pedestrian activities.

The rest of this thesis is organized as follows. Chapter 2 and Chapter 3 present approaches to mitigate issues on foot-mounted inertial sensors, including insufficient FSR and bandwidth as well as thermal-induced errors. Chapter 4 and Chapter 5 focus on the problems on algorithm assumptions and provide multi-sensor-aided stance phase detectors and an adaptive mechanism to automatically adjust measurement covariance of zero-velocity measurements. Chapter 6, Chapter 7, and Chapter 8 addresses the problems on estimation filters from three different aspects of the yaw angles, vertical position, and 3D positions, respectively. Chapter 9 presents a SLAM framework based on UWB and foot-mounted IMUs. Finally, Chapter 10 concludes the thesis with highlights of my contributions and suggestions for future research directions.

Chapter 2

On Motion Sensor – Overcoming Insufficient Sensor FSR and Bandwidth

2.1 Introduction

There are increased demands on inertial sensor's Full-Scale Range (FSR) and bandwidth when the sensor is mounted on the foot. The foot-mounted sensors experience large shocks that could saturate many COTS IMUs, degrading navigation performance of pedestrian navigation systems using such sensors. This chapter focuses on addressing the problem of sensor's insufficient FSR and bandwidth. The rest of this chapter is organized as follows. Section 2.2 experimentally investigates the minimum FSR of a foot-mounted IMU to capture signals without saturation in different pedestrian activities. Section 2.3 utilizes an analytical pedestrian inertial navigation simulation model, based on an inverted pendulum rigid body walking model, allowing for confirmation that insufficient sensor's FSR and bandwidth indeed lead to a significant increase in navigation errors. Section 2.4 develops an algorithmic approach to reconstruct saturated accelerometer signals. Section 2.5 presents a prioritizable

IMU array, a system-level approach integrating multiple different IMUs to boost the FSR and bandwidth. Section 2.6 summarizes this chapter with future research outlooks.

Section 2.3 of this chapter is a collaborative effort. The rigid body walking model presented in Section 2.3.1 was developed by UCI Professor Alexander Voloshina. The author of this thesis, Chi-Shih Jao, supervised by UCI professor Andrei Shkel, contributed to implementing reverse inertial navigation mechanization presented in Section 2.3.2, as well as the approach and results presented in Section 2.3.3, generating foot-mounted IMU signals that considered deterministic noises, stochastic noises, and sensor limitation.

2.2 Experimental Investigation of FSR Requirements

This section discusses properties of IMU measurements at different mounting positions, sources that lead to incorrect readings, and profiles of saturated IMU measurements.

An IMU consists of a triaxial accelerometer and a triaxial gyroscope, measuring specific forces that are proportional to linear accelerations and angular velocities experienced by the inertial sensors. During pedestrian navigation, IMUs deployed on different locations of a human body experienced different forces, which generated distinct patterns of IMU measurements.

To understand characteristics and limitations of the IMU measurements, a series of pedestrian navigation experiments of different pedestrian activities with multiple IMU mounting positions was conducted. The experiments were conducted in the Engineering Gateway Building at the University of California, Irvine. Figure 2.1 shows an experimental setup and experimental environment for this study. In the experiments, four IMUs, including one Invensense SmartBug, two VectorNav VN−200, and one Analog Device ADIS16497−3, were mounted firmly, with tapes, on the head, chest, left pocket, and left shoe of a pedestrian,

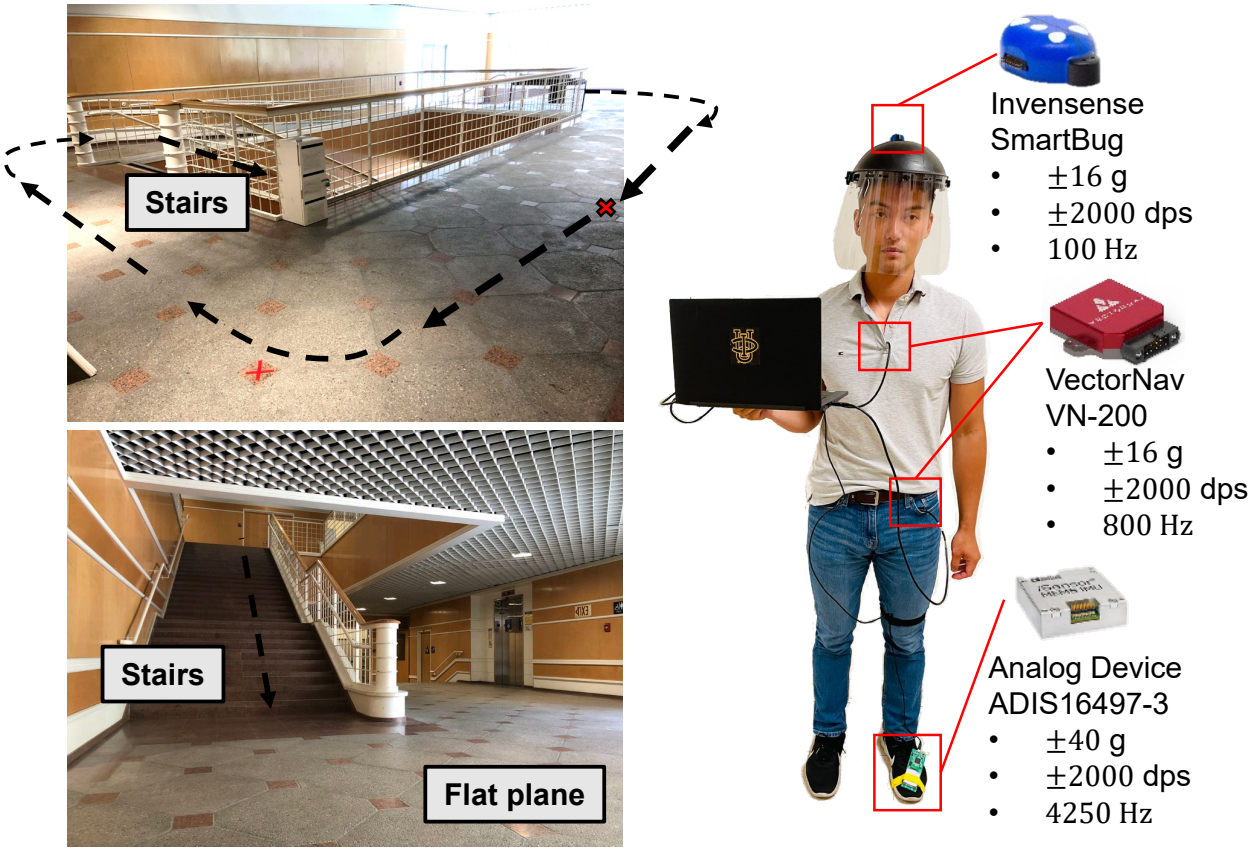


Figure 2.1: Experimental setup and experiment scenario discussed in Section 2.2. The Smartbug IMU (TDK/InvenSense) was attached with a double-sided tape on a face shield worn by the person. One VN−200 (VectorNav) was attached with a double-sided tape on chest of a person. The other VN−200 was placed inside the left pocket. The ADIS16497−3 (Analog Devices) was mounted with tape on the toe side of the left shoe.

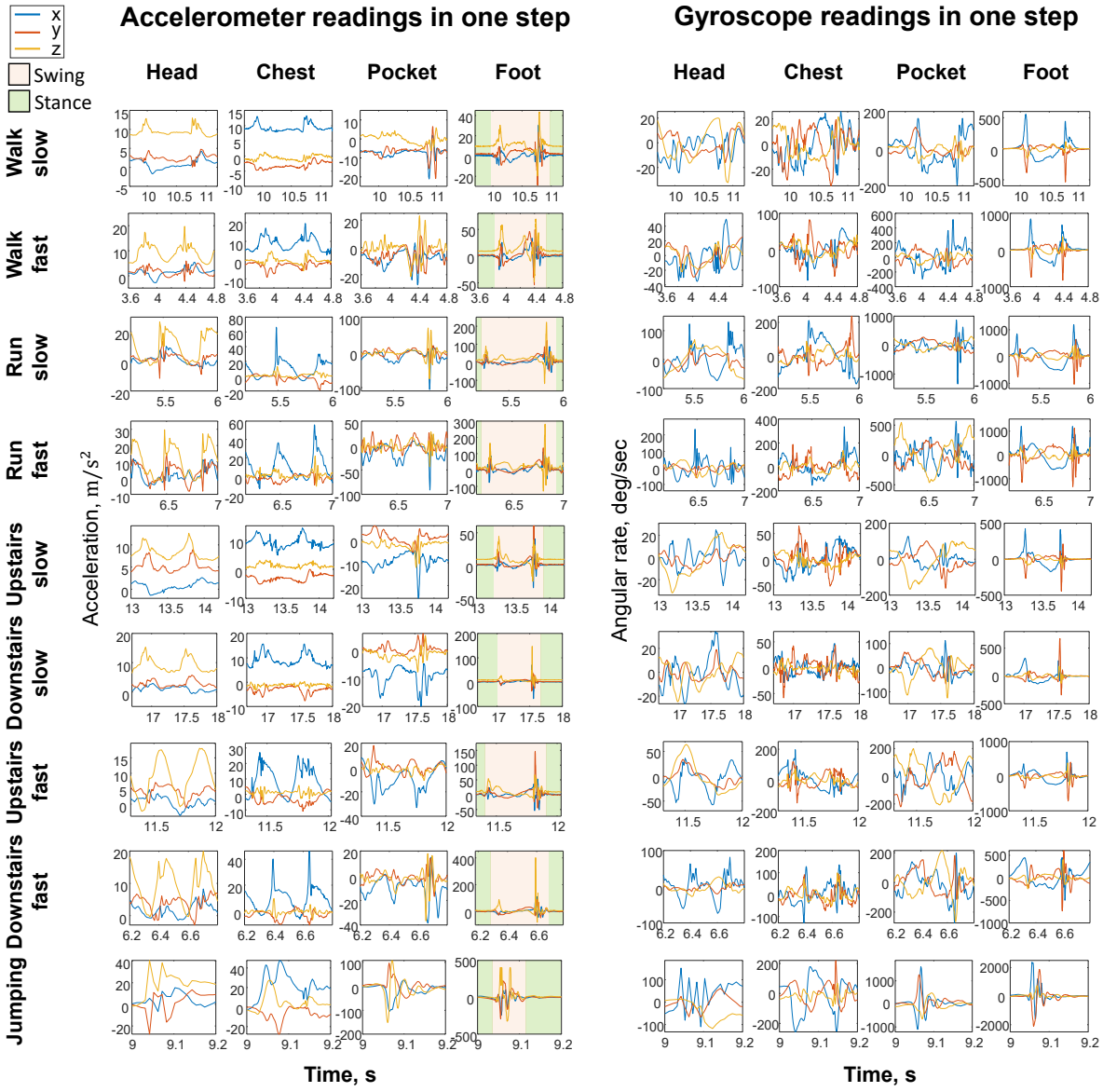


Figure 2.2: IMU readings at different mounting positions of a human body while performing everyday pedestrian activities. Plots in the same column show IMU readings collected with the same sensor, and plots in the same row correspond to measurements obtained within one complete gait cycle while performing the same activity.

respectively. With this setup, the pedestrian performed nine different activities, which are listed on the y-axis label of Figure 2.2. While performing the activities, the pedestrian used a metronome as a reference to the tempo of each step, not as a measurement itself. In each activity listed in Figure 2.2, a slow movement corresponds to a tempo of approximately 60 steps per second, and a fast movement means 120 steps per second. Duration of each experiment was around 60 seconds.

Figure 2.2 demonstrates the experimental results in the series of experiments. In Figure 2.2, plots in the same column display IMU readings collected with the corresponding sensor as described in Figure 2.1, and plots in the same row reveal the measurements obtained within one complete gait cycle while performing the same activity. In this study, one complete gait cycle contains two stance phases and one swing phase. The stance phase is the period when a foot is stationary on the ground, and the swing phase is when the foot is traveling in the air. In this series of experiments, the largest angular velocities were always measured in the step in which the largest acceleration occurred. The maximum magnitudes of accelerometer's readings and gyroscope's readings collected in each experiment with the different IMUs are documented in Table 2.1.

The following observation can be made about results presented in Figure 2.2.

1. In most activities, except for jumping, the z-axis accelerometer's measurements collected by the four different IMUs exhibit two peaks. In the cases of head- and chest-mounted IMUs, the peaks were generated by the heel-strike phases of two consecutive strides taken by two different feet. In the cases of pocket- and foot-mounted IMUs, the peaks were produced during the toe-off phase and the heel-strike phase of the same foot in the gait cycle.

2. Patterns of accelerometers' and gyroscopes' measurements collected by the four IMUs appeared different during the stance and swing phases. These differences can be utilized

for step detection, as described in [22] with head-mounted IMUs, [8] with pocket-mounted IMUs, and [218] with foot-mounted IMUs.

3. Foot-mounted IMUs demonstrated a distinct characteristic, as compared to other mounting positions. For example, during the stance phases, the acceleroemters' readings were nearly identical to unit gravity and the gyroscopes' readings were very close to zero measurements. This distinct characteristic allows for robust stance phase detection. Furthermore, with the assumption that the foot has minimal motion when the foot is on the ground, the characteristic enables applying a ZUPT algorithm to the strapdown inertial navigation systems.

4. The magnitudes of linear accelerations and angular velocities experienced by the foot-mounted IMU were significantly larger than the IMUs mounted at other positions. In this series of experiments, nominal gyroscope's FSR of the four different sensors had similar values of approximately ± 2000 degree per second (dps), which was verified by an inertial characterization equipment centrifuge Ideal Aerosmith Model 1571 at UCI Microsystems Lab[222]. It can be observed in Table 2.1, that the ± 2000 dps gyroscope's FSR was not large enough for the case of jumping. Note in Table 2.1, maximum accelerometers' and gyroscopes' readings of the Analog Device IMU were larger than their nominal FSR. This is because for the Analog Device's IMU, FSR is the largest amount of acceleration the sensor can measure accurately. A measurement larger than the FSR may not reflect the actual force the sensor experienced.

5. Nominal accelerometers' FSR of the four IMUs were different, which were ± 16 g for the Invensnese SmartBug and Vector VN$-$200 and ± 40 g for the Analog Device ADIS16497$-$3. The nominal accelerometers' FSRs were confirmed by inertial characterization in UCI Microsystems Lab with shaker APS Dynamics Model APS500 [222]. In Table 2.1, it can be observed that the accelerometers' FSR of the SmartBug and the VN$-$200 were not sufficient to fully reconstruct forces experienced by the foot-

mounted IMU in some activities, such as running slow, running fast, going downstairs fast, and jumping. In the cases of going downstairs fast, and jumping, even a ±40 g accelerometer's FSR was not high enough.

Table 2.1: Maximum accelerometer's and gyroscope's readings in different activities.

Mounting position	**Head (SmartBug)**		**Chest (VectorNav)**		**Pocket (VectorNav)**		**Foot (Analog Device)**	
	Accel (g)	Gyro (dps)	Accel (g)	Gyro (dps)	Accel (g)	Gyro (dps)	Accel (g)	Gyro (dps)
Walk slow	1.4	51.7	1.4	58.6	2.3	214.6	4.4	586.8
Walk fast	2.1	71.5	2.1	126.2	3.8	576	14.6	1083.9
Run slow	3.8	252.6	7.3	342.3	11.3	1342.9	22.5	1182.4
Run fast	3.5	312.8	5.8	366.1	11.3	1922.6	40.9*	2157.3*
Upstairs slow	1.4	67.3	1.5	74.6	3.4	188.4	8	563.6
Downstairs slow	2.1	59.9	2.5	95.9	3.5	216.1	15	796.4
Upstairs fast	2.1	126.7	5.1	322	5	479	29.4	1047
Downstairs fast	3.3	195.5	8.3	245.3	5.3	553.5	40.8*	1008.6
Jumping	4.1	152.1	6.2	402	15.3	1617.5	40.9*	2379.2*

* The measurements exceed nominal sensor FSR.

Based on these IMU readings, it can be emphasized that although foot-mounted IMU has the advantage of being able to use the ZUPT-aided INS algorithm, the requirements of sensor's FSR for foot-mounted systems are higher, and many COTS high-performance IMUs might not have characteristics that meet the requirements of the application. This section focuses on the issue of insufficient accelerometer's FSR in foot-mounted pedestrian navigation. In the next section, we use an analytical pedestrian simulation model to confirm that insufficient sensor FSR causes a significant error growth.

2.3 Confirmation Using Pedestrian Simulation Model

This section presents an analytical model with reduced complexity based on an inverted pendulum shown in Figure 2.3. This model is used to generate foot position for the case of foot-mounted INS-based navigation. The developed model accounts for high acceleration shocks during the heel-strike and toe-off phases, which enable simulating the effects of the limited sensor FSR and bandwidth on navigation accuracy. Additionally, six different IMU

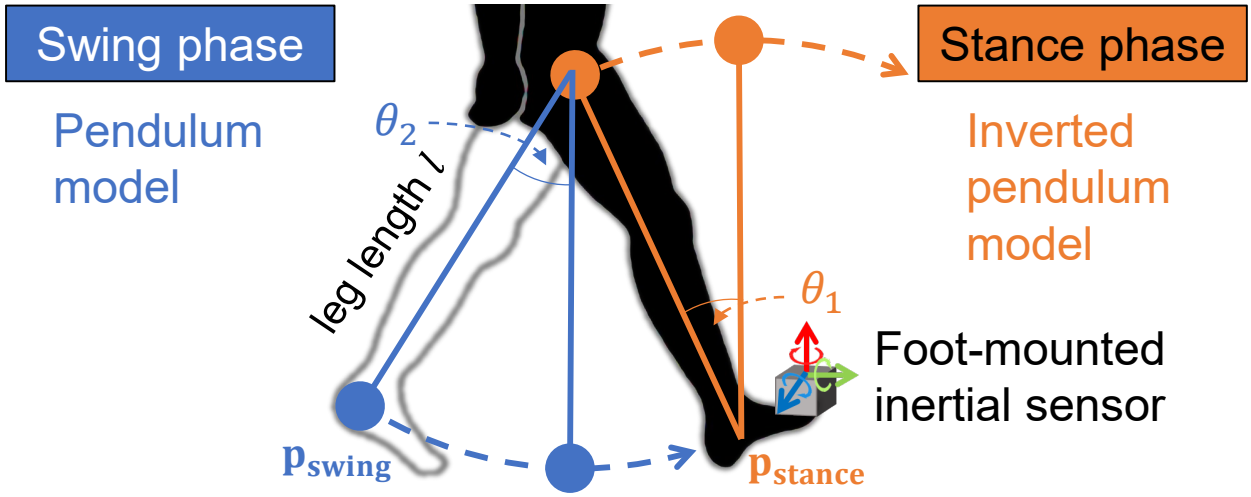


Figure 2.3: Modeling of walking dynamics using an inverted pendulum in the stance phase and regular pendulum in the swing phase.

noise sources, including white noise, bias instability, random walk, scale factor inconsistencies, misalignment, and turn-on bias were added to the simulated IMU signals.

2.3.1 An Analytical Walking Model Based on an Inverted Pendulum

We used a walking model, referred to as the rigid body walker [62, 114, 10], and investigated whether such a model is capable of predicting with sufficient accuracy the trajectory of the foot during walking. Figure 2.3 shows the configuration of the model, which consists of two rigid legs of length l of negligible mass connected by a frictionless hinge joint. The motion of the model is constrained to two dimensions. In this model, only one foot is on the ground at any given moment, during which time the system behaves like an inverted pendulum. The angle dynamics of the stance leg with respect to gravity, denoted as θ_1, are expressed as

$$\ddot{\theta_1} = \frac{g}{l}\sin(\theta_1),$$

where g is the gravity constant. The motion of the swing leg is dictated by the torsional hip spring, with a spring constant, k_{hip}, acting between the stance and swing legs. This is

analogous to hip flexor and extensor muscle activity during healthy human walking. The angle of the swing leg, θ_2, with respect to gravity, is calculated as

$$\ddot{\theta_2} = \frac{k_{\text{hip}}}{m_f l^2}(\theta_1 - \theta_2) + \frac{g}{l}(\sin(\theta_1 - \theta_2)\cos(\theta_1)) - \dot{\theta_1}^2 \sin(\theta_1 - \theta_2),$$

where m_f is the mass of the foot. With $m_f \ll m$, the swing leg does not affect stance leg dynamics [113]. The model is simulated to be analogous to human walking at a speed of 1.00 [m/s] and a step length of 0.662 [m]. To obtain generalized results, all model parameters were non-dimensionalized with respect to body mass (m), leg length (l, 0.87 m for typical human) and gravity (g). As a result, the simulated model gait had an average speed, $\bar{v}$, of $\bar{v}$=0.342 $\sqrt{lg}$.

Initial conditions of the model are set such that model dynamics repeat with each step. In other words, the model is on a limit cycle and all system states are identical for each step. The roles of the swing and stance leg switch when the swing leg contacts the ground. At this point, the model undergoes a perfectly inelastic collision, which redirects the center of mass velocity to be tangential to the new stance leg. Energy is lost as a result of this collision and must be replaced in order to maintain steady-state walking. To compensate for the energy loss, an impulse is applied along the stance leg immediately prior to foot contact of the swing leg. In other words, the impulse is perpendicular to the Center of Mass (CoM) direction of motion before collision and is analogous to the push-off work done by the lower leg muscles during human walking [113]. That is, push-off is *perpendicular* to the direction of motion of the CoM *before* collision. This redirects the CoM dynamics prior to collision, adding energy to the system and compensating for energy lost during foot contact [113].

The whole system provides a better understanding of the energy expenditure during human gait. The pendulum dynamics experience energy loss due to the collisions of the model after every step. To compensate for the loss, the model is powered by impulsive push-off P during

stance leg and hip work k acting between the legs. The algorithm applies the toe-off impulse before losing energy due to the collision of the heel-strike and later this value is used to set the initial condition for the upcoming steps. The hip torque is produced by a spring of stiffness k [114, 115]. The energy gained when the foot leaves the ground is compensated by a loss of kinetic energy. Thus, the metabolic effort is performed during the changeover between feet due to the work of the collision. An optimization algorithm provides the most optimal set of initial conditions (P and k) to create a limit cycle for the model, where each step is identical to the previous one. The algorithm performs diverse iterations to find the values that provide the minimum errors compared to the desired speed and step length defined by the user.

The model we are using simulates human walking at a velocity of 1m/s and at a step length of 0.662m, which is dictated by the optimized hip stiffness. To obtain generalized results, the outcome variables are non-dimensionalized with respect to body mass M, leg length (l, nominally 0.87 for typical human), and gravity g. The numerical integration began every time after heel-strike followed by toe-off. After a new heel-strike, the integration was interrupted to define a new stance leg. The simulated two-step gait relied on a combination of push-off and hip work for power.

The pelvis positions of the rigid body walker, $\mathbf{p}_{\text{pelvis}}$, during each step can be computed as

$$\mathbf{p}_{\text{pelvis}} = l \begin{bmatrix} -(\sin(\theta_1) - \sin(\theta_{1,0})) \\ 0 \\ \cos(\theta_1) \end{bmatrix} + \mathbf{p}_{\text{pelvis,0}},$$

where $\mathbf{p}_{\text{pelvis,0}}$ is the position of the pelvis at the end of the previous step and $\theta_{1,0}$ is the angle of the stance leg at the beginning of each step. Positions of the stance leg, $\mathbf{p}_{\text{stance}}$, and the

swing leg, $\mathbf{p}_{\text{swing}}$, are computed as

$$\mathbf{p}_{\text{stance}} = \mathbf{p}_{\text{pelvis}} + l \begin{bmatrix} \sin(\theta_1) \\ 0 \\ -\cos(\theta_1) \end{bmatrix}, \mathbf{p}_{\text{swing}} = \mathbf{p}_{\text{pelvis}} + l \begin{bmatrix} \sin(\theta_2) \\ 0 \\ -\cos(\theta_2) \end{bmatrix}.$$

In our simulation, positions of a foot in the navigation frame, denoted as $\mathbf{p}^n$, are obtained by alternating between positions of the stance leg and the swing leg. Roll angle ϕ and yaw angle ψ of the foot remain zero throughout the entire simulation, and the pitch angle θ alternates between θ_1 and θ_2.

2.3.2 Synthesizing Noise-Free IMU Readings

This study follows the strapdown INS algorithm, discussed for example in [198], and transforms the ground truth position $\mathbf{p}^n$ and orientations along roll ϕ, pitch θ, and yaw ψ angles generated from the rigid body walker to noise-free IMU signals. Velocities in the navigation frame $\mathbf{v}^n$ are computed from the position $\mathbf{p}^n$, and accelerations in the navigation frame $\mathbf{a}^n$are obtained from the velocity. Angular rates of the body frame with respect to the navigation frame $\boldsymbol{\omega}_{nb}^b$ are calculated from orientations as

$$\boldsymbol{\omega}_{nb}^b = \begin{bmatrix} \dot{\phi} \\ 0 \\ 0 \end{bmatrix} + \mathbf{C}_3 \begin{bmatrix} 0 \\ \dot{\theta} \\ 0 \end{bmatrix} + \mathbf{C}_3\mathbf{C}_2 \begin{bmatrix} 0 \\ 0 \\ \dot{\psi} \end{bmatrix},$$

where $\mathbf{C}_3$ is a rotation matrix rotating ϕ degree along the roll direction and $\mathbf{C}_2$ is a rotation matrix rotating θ degree along the pitch direction. Local gravity is computed as

$$\mathbf{g}_l^n = \mathbf{g} - \boldsymbol{\omega}_{ie}^n \times (\boldsymbol{\omega}_{ie}^n \times \mathbf{p}^n),$$

where $\mathbf{g}$ is the gravity vector and $\boldsymbol{\omega}_{ie}^{n}$ is the Earth rate. Noise-free accelerometer readings $\mathbf{u}_a$ are obtained as

$$\mathbf{u}_a = \mathbf{C}_n^b(\mathbf{a}^n + (2\boldsymbol{\omega}_{ie}^n + \boldsymbol{\omega}_{en}^n) \times \mathbf{v}^n + \mathbf{g}_l^n), \tag{2.1}$$

where $\boldsymbol{\omega}_{en}^{n}$ is the transport rate of the navigation frame and $\mathbf{C}_n^b$ is the DCM representation of $\boldsymbol{\omega}_{nb}^{b}$. Noise-free gyroscope readings $\mathbf{u}_g$ are expressed as

$$\mathbf{u}_g = \boldsymbol{\omega}_{nb}^b + \mathbf{C}_n^b(\boldsymbol{\omega}_{ie}^n + \boldsymbol{\omega}_{en}^n). \tag{2.2}$$

2.3.3 IMU Noise Model

This study considers eight different error sources for each sensor of an IMU, consisting of three accelerometers and three gyroscopes. The error sources include stochastic components of white noise, bias instability, and random walk; deterministic components of scale factor inconsistency, misalignment, and turn-on bias; and sensor limitations of FSR and bandwidth. These error sources are added to the noise-free accelerometers and gyroscopes readings described in (2.1) and (2.2).

Stochastic Noise Model

Simulated accelerometer readings corrupted with the stochastic components, denoted as $\hat{\mathbf{u}}_a$, are expressed as

$$\hat{\mathbf{u}}_a = \mathbf{u}_a + \hat{\mathbf{n}}_N + \hat{\mathbf{n}}_B + \hat{\mathbf{n}}_K, \tag{2.3}$$

where $\hat{\mathbf{n}}_N$, $\hat{\mathbf{n}}_B$, and $\hat{\mathbf{n}}_K$ denote the white noise, bias instability, and random walk components, respectively. This study modeled these three noise components with approaches presented

in [49], having the following formulations:

$$\hat{\mathbf{n}}_N = \boldsymbol{\omega}_N, \dot{\hat{\mathbf{n}}}_B = -\mu_B \hat{\mathbf{n}}_B + \boldsymbol{\omega}_B, \dot{\hat{\mathbf{n}}}_K = \boldsymbol{\omega}_K,$$

where $\boldsymbol{\omega}_N$, $\boldsymbol{\omega}_N$, and $\boldsymbol{\omega}_K$ are zero-mean Gaussian noise with variances of σ_N^2, σ_B^2, and σ_K^2, respectively. The values of the variances are to be determined via an optimization process that fits the noise models to the Allan Variance plot of a real IMU. μ_B is the correlation time associated with bias instability, and the value of μ_B was set to 10 [s] in this study.

Deterministic Noise Model

Simulated accelerometer readings corrupted with both the stochastic and deterministic components, denoted as $\tilde{\mathbf{u}}_a$, are expressed as

$$\tilde{\mathbf{u}}_a = \mathbf{M}(\hat{\mathbf{u}}_a + \mathbf{b}_0), \tag{2.4}$$

where $\mathbf{M}$ is a misalignment matrix with diagonal entries being scale factor errors along the three axes and off-diagonal entries being cross-axis sensitivity, and $\mathbf{b}_0$ is the turn-on bias. In this study, values of $\mathbf{b}_0$ were determined from a zero-mean Gaussian distribution with a

Table 2.2: Parameter settings for different noise models.

Parameter	**Accel Value**	**Gyro Value**
White noise σ_N	0.0015 [m/s$^{3/2}$]	1.74e−4 [rad/s$^{1/2}$]
Bias instability σ_B	3.92e−4 [m/s^2]	4.84e−5 [rad/s]
Random walk σ_K	1.01e−4 [m/s$^{5/2}$]	1.41e−4 [rad/s$^{3/2}$]
Scale factor error	0.05%	0.05%
Cross axis sensitivity	0.02°	0.02°
Turn-on bias $\sigma_{\mathbf{b}_0}$	0.01 [g]	0.3 [deg/s]
Full-scale range α_{FSR}	16 [g]	2000 [deg/s]
Bandwidth f_{cutoff}	260 [Hz]	256 [Hz]

variance of $\sigma^2_{\mathbf{b}_0}$ at the beginning of each simulation.

Sensor Measurement Limitation

This study applies a 6th-order Butterworth low-pass filter on $\tilde{\mathbf{u}}_a$ with a cut-off frequency, denoted as f_{cutoff}, to simulate the limited bandwidth of an IMU. To simulate sensor with limited FSR, α_{FSR}, we require that $|\tilde{\mathbf{u}}_a| < \alpha_{\text{FSR}}$. Simulated IMU readings that include the stochastic and deterministic noise components and measurement limitation, denoted as $\bar{\mathbf{u}}_a$, is expressed as

$$\bar{\mathbf{u}}_a = \begin{cases} \alpha_{\text{FSR}} & \text{if lowpass}(\tilde{\mathbf{u}}_a, f_{\text{cutoff}}) > \alpha_{\text{FSR}} \\ -\alpha_{\text{FSR}} & \text{if lowpass}(\tilde{\mathbf{u}}_a, f_{\text{cutoff}}) < -\alpha_{\text{FSR}} \\ \text{lowpass}(\tilde{\mathbf{u}}_a, f_{\text{cutoff}}) & \text{otherwise} \end{cases} \tag{2.5}$$

The processes described in (2.3)-(2.5) with different noise parameter settings are used to obtain simulated gyroscope readings corrupted with the error sources, denoted as $\bar{\mathbf{u}}_g$. In this study, IMU readings were simulated based on noise characteristics of a VectorNav IMU VN−200. The parameters of stochastic noise components were determined experimentally with the Allan Variance test, and the deterministic component and sensor measurement limitation parameters were set nominally according to the sensor datasheet. Values of the parameters used in our developed approach are summarized in TABLE 2.2. The sampling rate of the simulated IMU was set to 800 [Hz].

Figure 2.4 shows profiles of IMU readings in two steps generated with the developed simulation and collected with a VN−200 IMU. In Figure 2.4, acceleration shocks could be observed during the heel-strike and toe-off phases, in both the simulated and experimented IMU readings. The acceleration shocks in the simulation were generated because the rigid

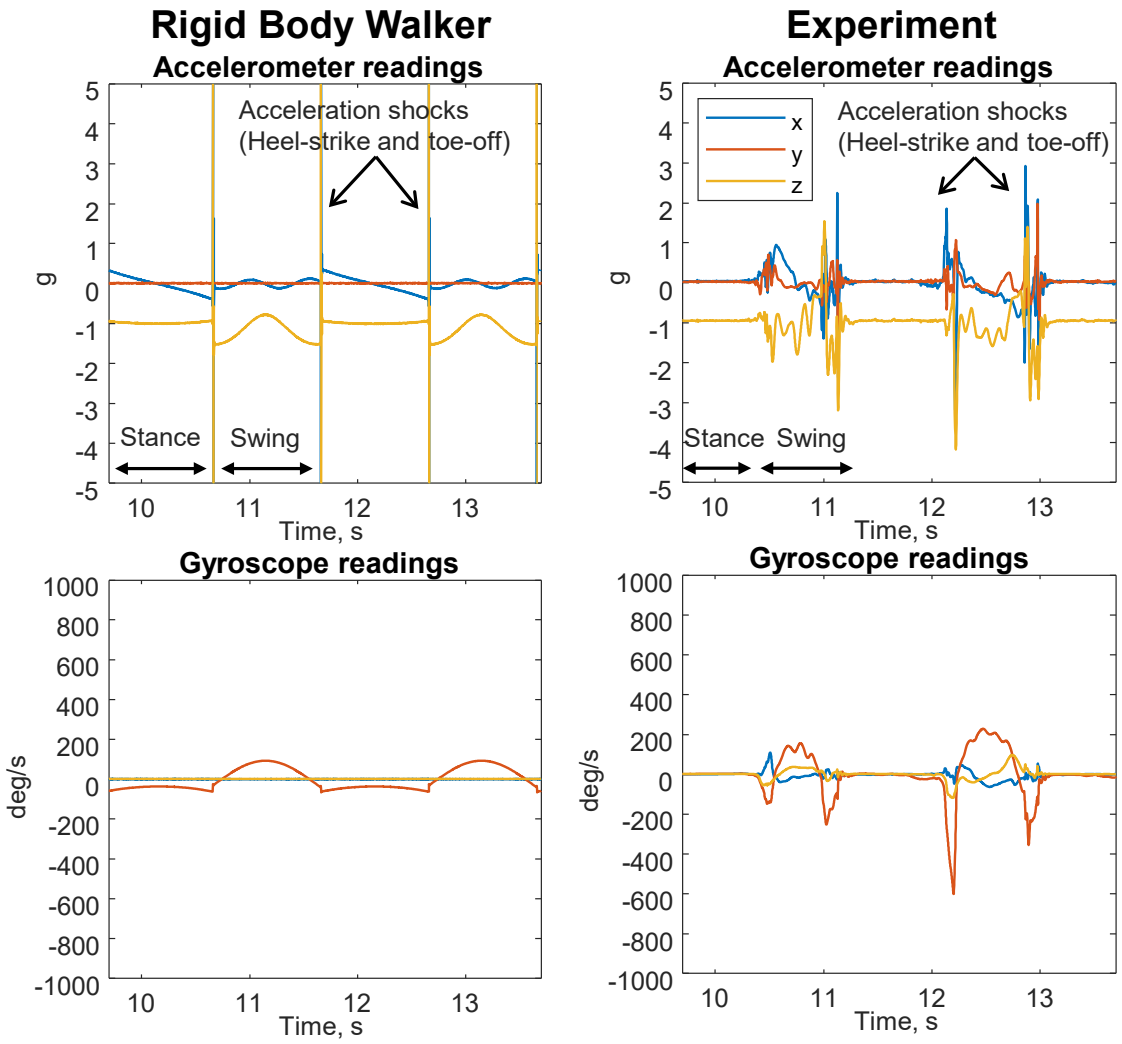


Figure 2.4: Example profiles of simulated and measured IMU readings in two steps, in the case of walking along a straight line. The left column represents modeled sensors' readings, and the right column represents experimental sensors' readings.

body walker model discussed in Section 2.3.1 considers events of the foot colliding with the ground, which causes a discontinuity in velocity measurements. The maximum acceleration shock generated in our model was around 200 [g].

2.3.4 Comparing Simulated and Experimental Results

This study compares the navigation accuracy of the ZUPT-aided INS using a series of 20 simulations and 20 experiments. In each run of the simulations, we used the model discussed in Section 2.3.1 with 28 steps, resulting in a straight-line trajectory of 42.86 [m]. In the experiments, an IMU VN−200 was mounted on the toe-side of a subject's boot, and the sampling rate of the sensor was set to 800 [Hz]. The subject walked a straight-line trajectory of 42.3 [m] with each step length of approximately 0.75 [m] and a pace of around 60 steps per minute. The ZUPT-aided INS was implemented in an EKF with the Acceleration-Moving Variance (AMV) detector [181]. The initial yaw angle of each navigation solution was assumed to be aligned with the North.

Figure 2.5 shows the navigation results of the simulations (left column) and experiments (right column). 3D Root Mean Square Error (RMSE), horizontal Circular Error Probable (CEP), and vertical ($\perp$) RMSE between the simulated and experimental results had a difference of 6%, 40%, and 7%, respectively. The vertical positions in the simulations and experiments drifted in the same direction. To the best of our knowledge, this is the first pedestrian navigation simulation model that captures with sufficient accuracy the vertical positioning drifts. It could be shown with our model that insufficient sensor FSR and bandwidth are the dominating sources of the drifts. This observation supports the hypothesis on the causes leading to the vertical positioning drifts in the foot-mounted INS that were discussed in previous research [205].

Two lessons could be learned by observing the results presented in Figure 2.5. First, the $\perp$

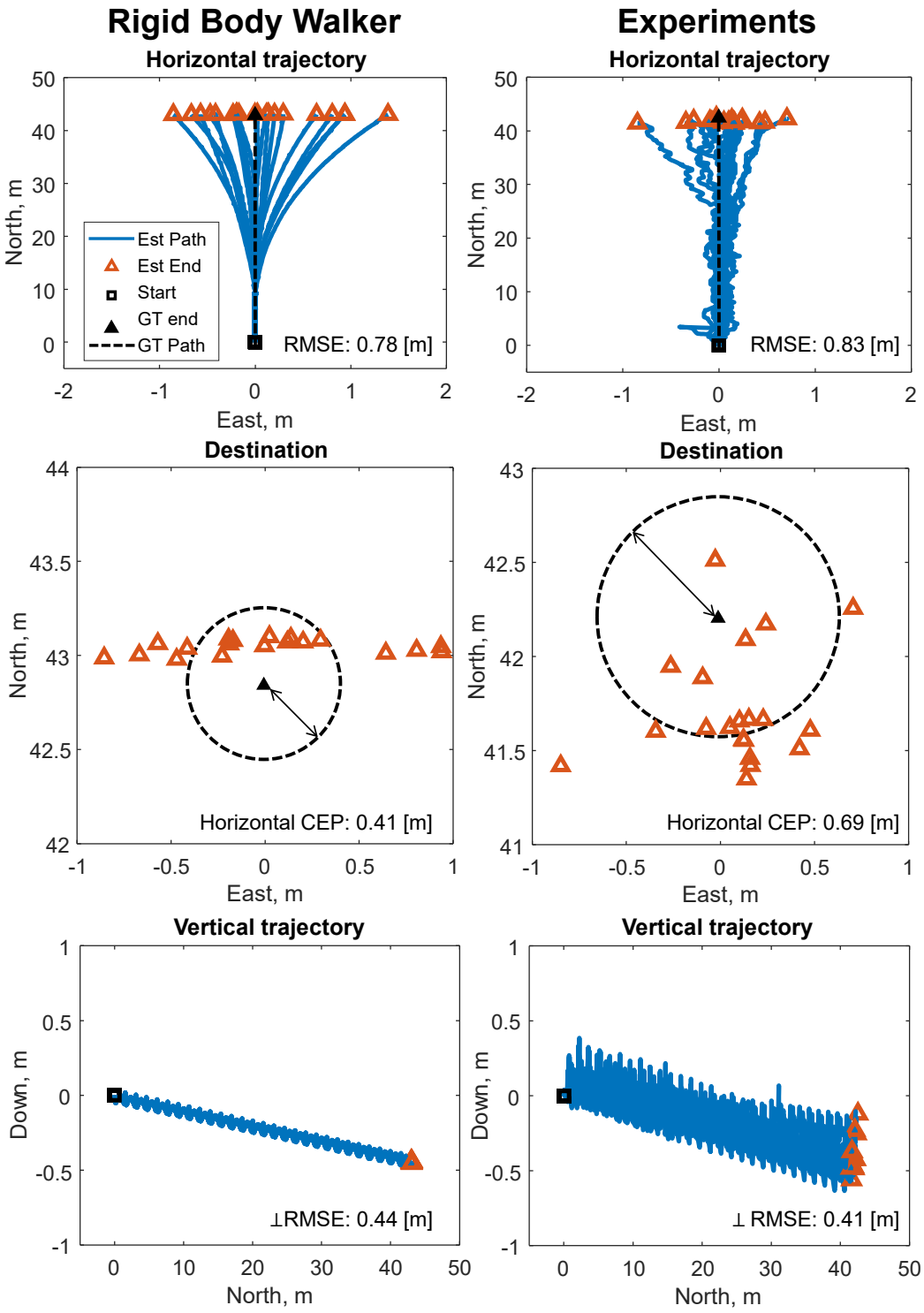


Figure 2.5: Comparison of navigation accuracy of the ZUPT-aided INS in the cases of rigid body walker simulation and experiments with VN−200 IMU. The left column represents modeling, and the right column represents experiments.

RMSE of the simulation results was larger than the experiments. This was due to the fact that the simulated accelerometer readings, as shown in Figure 2.4, even though capturing the trend, had significantly larger shocks along the z-axis than the experimental accelerometer readings. When limiting in the simulated IMU readings the accelerometer FSR to 16 [g], large accelerometer biases were introduced, resulting in a larger vertical positioning error. Second, the estimated trajectory length in the experiments had larger deviations than in the simulation. We believe the reason is that the simulated IMU signals had identical patterns in each step, while, in the experiments, the signals had slightly different patterns of the human subject's walking style. Thus, a fixed threshold in the stance phase detection could be optimal in the simulation but not in the experiments. These observations suggest that, in order to improve the accuracy of navigation uncertainty prediction, future research should augment the model with more sophisticated foot motions. One potential research direction is to combine the analytical rigid body walker with traditional approaches of generating foot motion using motion cameras or IMUs.

This section presented a simple, yet practical, model that is sufficient to predict with high accuracy the navigation uncertainty of a ZUPT-aided INS using a foot-mounted IMU. The experimental results showed that the developed simulation model had a 6% discrepancy in position RMSEs, as compared to experiments, but captured all main features of motion. Our model also accurately predicted the drift in the vertical direction, matching well the reported experiments. The simulation results show that insufficient FSR and bandwidth were factors causing the drifts along the vertical direction. The results discussed in this section were published in [85].

In the following two sections, we present two enhancement approaches for pedestrian navigation systems using foot-mounted IMUs to mitigate the impact of saturated IMU signals on positioning accuracy.

2.4 Algorithmic Reconstruction of Saturated Signals

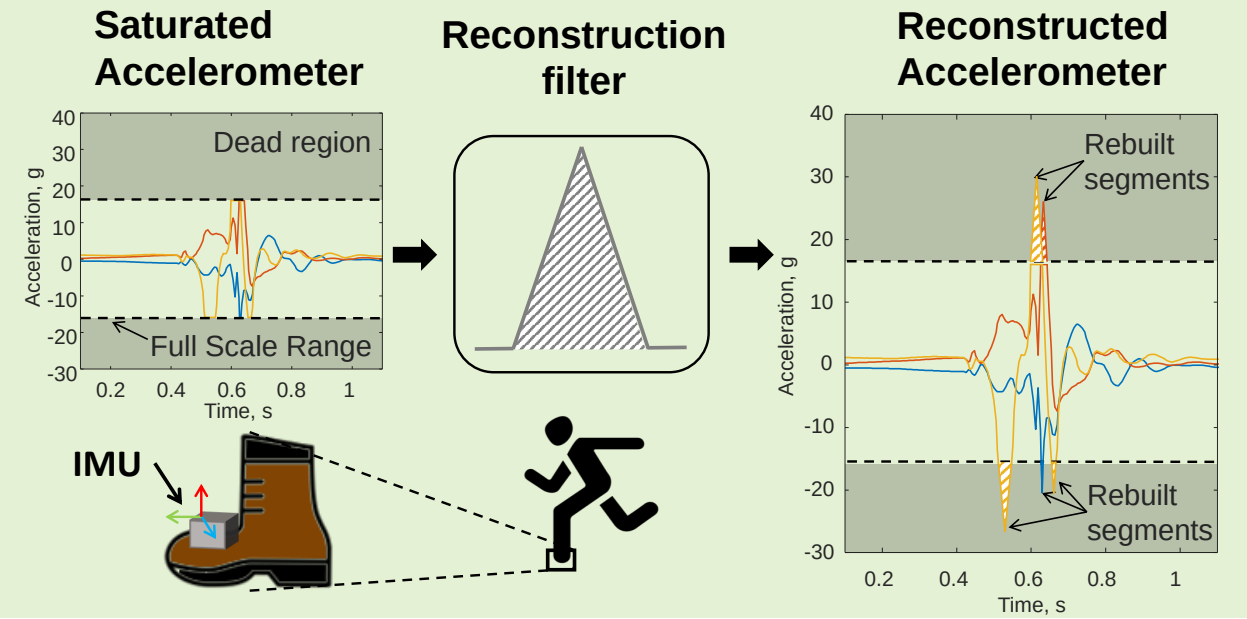


Figure 2.6: Concept of the developed reconstruction filter.

This section presents a reconstruction filter to pre-process accelerometer signals for a ZUPT-aided INS. The developed reconstruction filter aims to rebuild saturated accelerometer signals due to insufficient accelerometer's FSR. Reconstructing saturated accelerometer signals is considered equivalent to predicting immeasurable signals. Immeasurable signals are defined in this section as the acceleration outputs that can be correctly measured by accelerometers with very high FSR but are immeasurable if the FSR is low. The concept of the developed reconstruction filter is based on observations that profiles of immeasurable signals during the heel-striking phase are similar to a triangular function, which are parameterized with amplitude α and period λ. When a sequence of saturated accelerometer signals are detected, the developed reconstruction filter estimates the parameters α and λ, generates a triangle wave based on the estimated parameters, and superimposes the triangle wave to the saturated accelerometer signals. Reconstructed accelerometer's signals, which are outputted from the reconstruction filter, are inputted to the ZUPT-aided INS. Because noise performance of reconstructed signals and raw signals are different, the process noise settings

for accelerometer bias states in the EKF need to be modified. The developed reconstruction filter also predicts a noise variance for every reconstructed accelerometer's readings, and the predicted noise variances are used to vary the value of the process noise in the ZUPT-aided INS.

The rest of the section is organized as follows. Section 2.4.2 discusses derivation and implementation of the developed reconstruction filter. Experimental results are presented in Section 2.4.3 and concludes the section with a highlight of our main results.

2.4.1 Properties of Saturated Foot-mounted IMU Measurements

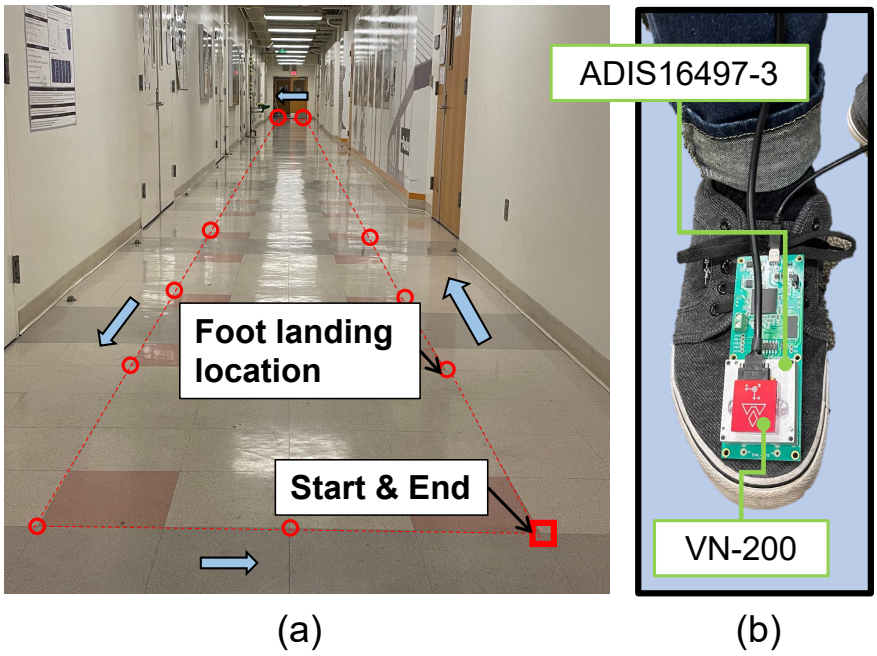


Figure 2.7: (a) Experimental setup of controlled indoor navigation experiments. The red circles indicate foot-landing locations, and the blue arrows illustrate traveling directions. (b) Experimental setup of foot-mounted IMUs. The VN−200 IMU (red) was mounted on top of the ADIS16497−3 IMU (silver). Both IMUs were firmly attached on the toe side of the foot.

As shown in Figure 2.2, foot-mounted inertial sensors are experiencing high linear accelerations and angular velocities while performing common pedestrian activities. When sensors are not designed for the required FSR, the pedestrian activities may lead to saturated IMU

measurements. To confirm that saturated measurements do not represent actual forces experienced by a foot-mounted IMU, two series of ten indoor walking experiments with two IMUs, including an Analog Device ADIS16497−3 and a VectorNav VN−200, were conducted in the Engineering Gateway Building at the University of California, Irvine. The experimental setup is shown in Figure 2.7(a). The two IMUs were mounted firmly next to each other on a pedestrian's right foot, as illustrated in Figure 2.7(b). The illustrated scenario will be used in the next section to show how shortcomings of FSR of sensors can be addressed algorithmically.

In the first series of experiments, the pedestrian began by standing still for 20 seconds. Then, the pedestrian walked, in a way that he felt most comfortable, for exactly 74 steps at a pace of around 70 steps per min in a close-loop rectangular trajectory, as shown with the red lines in Figure 2.7(a). During the walk, the pedestrian landed the foot on locations, marked with the red circles in Figure 2.7(a), at every step. The controlled foot landing landmarks were used for better control and evaluation of the experiments. The stride length of each step was 121.92 cm (4 feet) when traveling along the hallway and 60.96 cm (2 feet) when perpendicular to the direction. The trajectory length was 87.8 m, and the duration was around 120 seconds. The IMU measurements showed that, on average, a maximum acceleration shock of 2.3 g was generated during each step. In the second series of experiments, the pedestrian repeated the same trajectory but landed his foot hard on the floor during walking that produced a maximum shock in each step with a mean value of 23.6 g. The maximum shock usually occurred along the vertical version. It should be noted that this acceleration exceeded the accelerometer's FSR of the VN−200. Sampling rates of ADIS16497−3 and VN−200 were set to 850 Hz and 800 Hz, respectively. It is worth noting that gyroscopes' FSR, for both the ADIS16497−3 and the VN−200, was 2000 deg/sec, and the maximum angular rate measured in all the experiments was around 1600 deg/sec. These angular rates were within the FSR of IMUs used for the experiments but would exceed FSR of many consumer-grade COTS gyroscopes on the market.

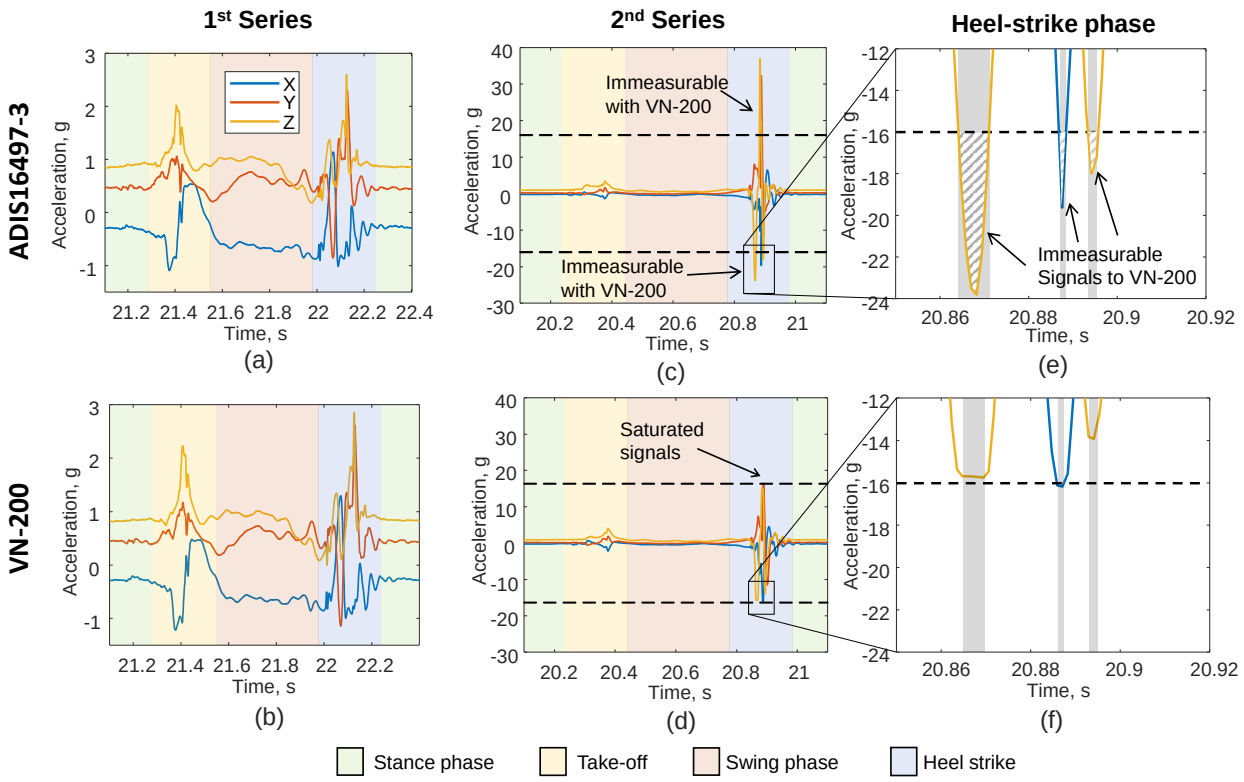


Figure 2.8: (a) Accelerometer readings of one gait cycle collected with an Analog Device ADIS16497−3 IMU in the first series of experiments discussed Section 2.4.1. (b) Accelerometer readings collected with a VectorNav VN−200 IMU during the same time period as (a). (c) Accelerometer readings of one gait cycle collected with the Analog Device IMU in the second series of experiments discussed Section 2.4.1. Accelerometer readings collected with the VectorNav IMU during the same time period as (c). (e) A zoomed-in view of (c), showing signals pattern in a heel-strike phase of a gait cycle. The areas marked with striped patterns indicate measurements of accelerations having magnitudes larger than 16 g. For the VN−200 IMU, these accelerometer's measurements could not be correctly measured and are called the immeasurable signals in this section. (f) A zoomed-in view of (d), showing a saturated signal pattern in the heel-strike phase.

Figure 2.8 shows an example of the accelerometer's readings of the VectorNav and the ADIS16497−3 collected during one gait cycle in the first and the second series of experiments. Three observations were made in Figure 2.8. First, the accelerometer readings in Figure 2.8(a) and (b) showed similar profiles and did not exceed the accelerometers' FSR of the two IMUs. Second, in Figure 2.8(c), the Analog Device IMU measured an acceleration as large as 37 g during the heel strike phase, while in Figure 2.8(d), the VectorNav measured a maximum acceleration of around 16 g. Since the accelerometer's FSR of VN−200 was 16 g, the VectorNav IMU was considered to have incorrect measurements during the second series of experiments. The highlighted area shown in Figure 2.8(e) is an example of an immeasurable signal of the VN−200 IMU. Third, in Figure 2.8(f), it could be seen that the third pick did not reach 16 g. However, in Figure 2.8(e), the Analog Device IMU clearly measured an acceleration larger than 16 g along the x-axis. In our opinion, the third pick in Figure 2.8(f) is an incorrect measurements, but it did not display a value as large as the accelerometer's FSR of the VN−200 because during the short period of time, measurements were not sampled by the inertial sensor when the maximum acceleration occurred.

These two series of experiments illustrated that in the case of insufficient FSR, large forces experienced by the foot-mounted IMU could lead to incorrect accelerometer's readings. The matching between FSR and profile of accelerometers is critical in ZUPT-aided INS. The next section of this section will introduce a reconstruction filter that aims to mitigate the negative effect that insufficient accelerometer's FSR brings to foot-mounted INS. The introduced approach allows to relax FSR requirements.

2.4.2 A Reconstruction Filter

This section aims to provide an approach to reconstruct, in real-time, accelerometer's measurements that are saturated due to insufficient FSR. This section discusses modeling the

saturated accelerometer's readings and presents the a reconstruction filter resolving the saturated signals.

Modeling of Saturated Accelerometer Signals

This section defines immeasurable signals of an accelerometer as the measurements of acceleration that have magnitude larger than the FSR of the sensor. Our developed reconstruction filter is designed to predicts immeasurable signals and superimposes the predicted signals to saturated measurements. To characterize the immeasurable signals, this section models accelerometer's measurements, $y_{a,\beta}(n)$, that are collected at time n with FSR of value β, as follows:

$$y_{a,\beta}(n) = y_F(n) + b(n) + \omega(n), \tag{2.6}$$

where

$$b(n) = b_a(n) + \Psi_\beta(n),\ \omega(n) \sim N(0, \sigma^2_{\text{VRW}}).$$

Here, $y_F(n)$ is true specific forces experienced by the accelerometers, $\omega(n)$ is a white Gaussian noise components have a standard deviation of σ_{VRW}, and $b(n)$ is an accelerometer bias. This section considered that the accelerometer bias consists of a stochastic component, $b_a(n)$, and a deterministic bias, $\Psi_\beta(n)$. The stochastic component is the inherent bias of accelerometers and is commonly described by in-run bias instability. The deterministic component $\Psi_\beta(n)$ is caused by using insufficient FSR of a value β to measure large forces. The deterministic bias $\Psi_\beta(n)$ has the same magnitude as an immeasurable signal but with opposite sign. It is worth noting that $\Psi_\beta(n)$ depends on an accelerometer's FSR. Figure 2.8(e) and (f) show examples of profiles of accelerometer's signals collected by the Analog Device IMU and the VectorNav IMU, respectively, during a heel-strike phase. In the case of the Analog Device IMU, the

accelerometer's FSR, $\beta = 40$ g, and $\Psi_{40}(n) = 0$ everywhere. In the case of the VectorNav IMU, $\beta = 16$ g, and $\Psi_{16}(n)$ is considered to have a profiles similar to the accelerometer signals shown in Figure 2.8(e) that have magnitude larger than 16 g.

It can be observed in Figure 2.8(e) that the immeasurable signals are formed by individual humps. Each of such humps is denoted as $\psi_{\beta,k}(n)$. Then, the mathematical expression of the bias $\Psi_\beta(n)$ can be described as follows:

$$\Psi_\beta(n) = \left\{ \begin{array}{ll} \psi_{\beta,k}(n), & t(n_{s,k}) \leq t(n) < t(n_{s,k}) + \tau_k \\ 0, & \text{elsewhere} \end{array} \right\}$$

Here, $t(n)$ is elapsed time at sample n and $n_{s,k}$ is the sample where a hump $\psi_{\beta,k}(n)$ is first detected.

Design of Reconstruction Filter

This section aims to provide an approach to mitigate navigation errors due to insufficient accelerometer's FSR by reconstructing the immeasurable signals. The reconstruction process is equivalent to predicting the deterministic bias component, $\Psi_\beta(n)$ and adding it to the saturated signals.

Measuring Immeasurable Signals In order to section the characteristics of each $\psi_{\beta,k}(n)$, it is needed to collect a series of $\psi_{\beta,k}(n)$ during pedestrian navigation experiments. However, acquiring the true $\psi_{\beta,k}(n)$ is challenging as it would require knowledge of the true stochastic accelerometer bias, $b_a(n)$, which is not accessible when an IMU is in motion. In this section, it was assumed that the ZUPT-aided INS could accurately estimate the stochastic bias. Based on the assumption, we approximated $\Psi_\beta(n)$ and denoted the approximated version as $\Psi'_\beta(n)$. Following this notation, the humps $\psi_{\beta,k}(n)$ are approximated with $\psi'_{\beta,k}(n)$. The

approximated version $\Psi^{'}_{\beta}(n)$ can be obtained by subtracting unsaturated accelerometer signals, collected using an IMU with a higher accelerometer's FSR, with the saturated one. For example, consider a case where one accelerometer has an FSR of value β_1 and another has a value β_2 with a condition that $\beta_1 > \max(y_{a,\beta_1}(n)) > \beta_2$. In such case, $\psi^{'}_{\beta_2,k}(n)$ is expressed as follows:

$$\psi^{'}_{\beta_2,k}(n) = y_{a,\beta_1}(n) - y_{a,\beta_2}(n) = y_{a,\beta_1}(n) - \text{sgn}(y_{a,\beta_2}(n))\beta_2, \text{ for } n_{s,k} \leq n < n_{s,k} + d_k, \quad (2.7)$$

where sgn() is the sign function. The second equation holds because it was assumed that during saturation periods, $y_{a,\beta_2} = \beta_2$ if $y_{a,\beta_2} > 0$ and $y_{a,\beta_2} = -\beta_2$ if $y_{a,\beta_2} < 0$.

In the experiments described in Section 2.4.1, $\beta_1 = 40$ and $\beta_2 = 16$, as the the ADIS16497−3 IMU and the VN−200 IMU had an accelerometer FSR of 40 g and 16 g, respectively. The ADIS16497−3 was used to collect a dataset of $\psi^{'}_{16,k}(n)$ for the VN−200 IMU. To do so, we artificially saturated, with a threshold of 16 g, the accelerometers' readings collected by the Analog Device IMU in the second series of experiments described in Section 2.4.1. In the experiments, there were 494 detected saturation events in total. Figure 2.9 demonstrates an example of profiles of approximated deterministic bias $\Psi'_{16}(n)$ for the VN−200 IMU. We could see in Figure 2.9 that there were three individual humps. The yellow humps correspond to the deterministic bias along the z-axis and the yellow hump marks the bias along the y-axis. In Figure 2.9, no x-axis deterministic bias was collected. Furthermore, by observing the collected profiles of $\psi^{'}_{16,k}(n)$, we discovered two properties of $\psi^{'}_{16,k}(n)$. First, profiles of $\psi^{'}_{16,k}(n)$ are not identical but have a similar triangular shape. Second, we found that areas under $\psi^{'}_{16,k}(n)$ is proportional to saturated period.

Characterizing Immeasurable Signals Based on the observation made in Figure 2.9, we characterized each $\psi_{\beta,k}(n)$ with height, h_k, width, τ_k, and area, A_k. τ_k is a period where a sequence of accelerometer signals are saturated and will be referred to as saturation period

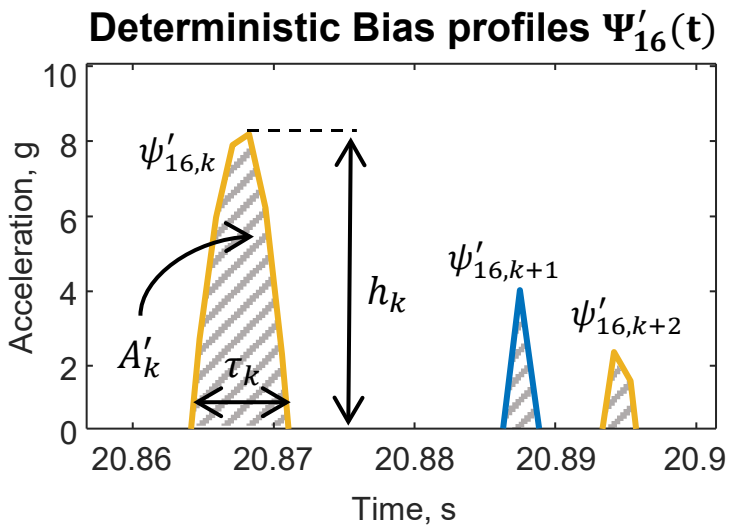


Figure 2.9: Three examples of deterministic bias profiles of an IMU having accelerometer's FSR of 16 g.

in the following texts. h_k and A_k are described as

$$h_k = \text{sgn}(y_a(m)) \max_{n_{s,k} \le x \le n_{s,k}+d_k} |\psi_{\beta,k}(x)|$$

$$A_k = \int_{t(n_{s,k})}^{t(n_{s,k})+\tau_k} |\psi_{\beta,k}(t)|\, dt = \sum_{m=n_{s,k}}^{n_{s,k}+d_k} |\psi_{\beta,k}(m)|\delta t$$

where $d_k = \frac{\tau_k}{\delta t}$ and δt is the sampling period of an accelerometer.

In practical situation where $\psi'_{16,k}(n)$ is used instead of $\psi_{\beta,k}(n)$, the saturation area A'_k can be measured as follows:

$$A'_k = \sum_{n=n_{s,k}}^{n_{s,k}+d_k} (|\psi'_{16,k}(n)|) = \sum_{n=n_{s,k}}^{n_{s,k}+d_k} (|y_{a,40}(n) - \text{sgn}(y_{a,40}(n))16|). \tag{2.8}$$

Figure 2.10 shows a relationship of A'_k and τ_k. The data points marked in blue in Figure 2.10 and their corresponding histogram indicated with the orange bar illustrate the statistics of saturation areas at different saturation periods based on the IMU measurements collected with the Analog Device IMU during the second series of experiments described in Section

2.4.1. It could be seen in Figure 2.10 that around 80% of the data had a saturation period of less than 6 ms. Moreover, the statistical mode of the collected data appears at 0.0035 ms.

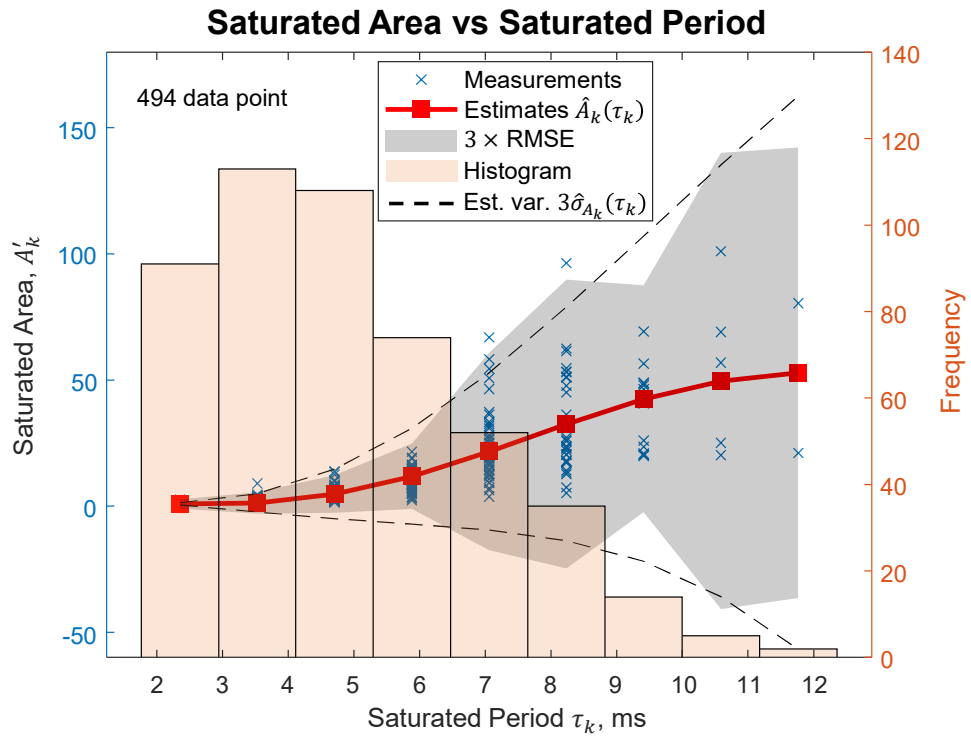


Figure 2.10: Relationship of saturation area $A_k^{'}$ and saturation period τ_k. The blue dots marked measurements of saturated area $A_k^{'}$ in the second series of experiments discussed in Section 2.4.1. The red curve represents saturated areas predicted by a GP regression discussed in Section 2.4.2. The grey shadow areas indicate the 3× RMSE of the prediction. The black dashed lines illustrate a 2^{nd}-order polynomial for curve-fitting the RMSE. The orange bars indicate measurement distribution.

Approximating Immeasurable Signals With a Triangular Function Although $\psi'_{\beta,k}(n)$ is measurable, the measurements cannot be achieved in a real-time manner if only a single IMU is used in a pedestrian navigation system. Therefore, estimation of $\psi'_{\beta,k}(n)$ is needed. Based on the observation in Figure 2.9 that the immeasurable signals have triangular shape, we hypothesized that the $\psi'_{\beta,k}(n)$ can be approximated with a scale version of triangle wave functions, $\Lambda_k(n, \alpha, \lambda)$, which is expressed as follows:

$$\Lambda_k(t, \alpha_k, \lambda_k) = 2\alpha_k|\frac{t}{\lambda_k} - \lfloor\frac{t}{\lambda_k} + \frac{1}{2}\rfloor|, \tag{2.9}$$

where α_k is the amplitude, λ_k is the period of the triangle wave, and $\lfloor\ \rfloor$ is the floor function. The area of the triangular wave, $\Delta_k(\alpha_k, \lambda_k)$ is calculated as

$$\Delta_k(\alpha_k, \lambda_k) = \frac{\alpha_k \lambda_k}{2} \tag{2.10}$$

The estimated deterministic bias, denoted as $\hat{\Psi}_{16}(n)$, is expressed as follows.

$$\hat{\Psi}_{16}(n) = \left\{ \begin{array}{ll} \hat{\psi}_{16,k}(n), & t(n_{s,k}) \leq t(n) < t(n_{s,k}) + \tau_k \\ 0, & \text{elsewhere} \end{array} \right\} \tag{2.11}$$

where

$$\hat{\psi}_{16,k}(n) = \text{sgn}(y_{a,16}(n))\Lambda(n, \alpha_k, \lambda_k)$$

Here, λ_k is available when operating in real-time and has the same value as τ_k. However, α_k, which determines the amplitudes of the triangle wave, cannot be measured directly from saturated accelerometer signals and needs to be estimated.

Estimating Immeasurable Signals Using Gaussian Process Regression In this section, the estimation of α_k aims to minimize the statistical expectation of difference in $A_k^{'}$ and $\Delta_k(\alpha_k, \lambda_k)$. For each measured $\psi_{16,k}^{'}(n)$, the corresponding τ_k was considered as a feature and the corresponding $A_k^{'}$ as a label. Then, the saturation areas were modeled as a Gaussian Process (GP). The modeled GP has a constant basis function and a squared exponential kernel. The GP was trained with the 494 measurements of saturation areas and their corresponding saturation periods. The trained GP is found to have a parameter for the basic function, $\beta = 24.5$, and an estimated noise, $\sigma = 8.25$. The trained GP was then used to predict a saturation area for each detected saturation period τ_k. The predicted saturation area is denoted as $\hat{A}_k(\tau_k)$.

The red curve presented in Figure 2.10 shows the predicted saturation area $\hat{A}_k(\tau_k)$. The grey shadow area represents the RMSE of the prediction evaluated at each saturation period. Based on the results shown in Figure 2.10, several observations were made. First, the RMSEs of the data corresponding to saturation period less than 6 ms, which takes up 80% of the entire dataset, are smaller than the RMSEs of the other 20% of the data. Second, in the cases of saturated periods higher than 6 ms, the RMSEs increased, indicating that the prediction had large uncertainties. The large uncertainties indicate that behaviors of the immeasurable signals in the cases of the higher saturation periods were less predictable. In this case, the developed reconstruction filter can have limited performance, and sensors with a larger accelerometer's FSR are needed.

The estimation of α_k, denoted as $\hat{h}_k(\tau_k)$, that minimized difference between $A_k^{'}$ in (2.8) and $\Delta_k(\alpha_k, \lambda_k)$ in (2.10) can be found as follows:

$$\hat{h}_k(\tau_k) = \frac{2\hat{A}_k(\tau_k)}{\tau_k}. \tag{2.12}$$

Using (2.12), the height of the triangular wave function is determined, which is used to estimate $\psi_{16,k}^{'}(n)$.

Besides estimation of α_k, we would also like to predict the uncertainty of estimated α_k because the RMSEs presented in Figure 2.10 shows that the uncertainty of $\hat{A}_k(\tau_k)$ is not constant at different values of τ_k. We used a 2^{nd}-order polynomial with a least-square cost function to fit the calculated RMSEs marked with the grey area in Figure 2.10. The fitted curve, denoted as, $\hat{\sigma}_{A_k}(\tau_k)$ was utilized to predict the uncertainty of $\hat{A}_k(\tau_k)$. The two black dashed curves shown in Figure 2.10 illustrate $\hat{A}_k(\tau_k) + 3\hat{\sigma}_{A_k}(\tau_k)$ and $\hat{A}_k(\tau_k) - 3\hat{\sigma}_{A_k}(\tau_k)$, respectively. The uncertainty of estimated α_k, denoted as $\hat{\sigma}_{\alpha_k}(\tau_k)$, can be expressed as follows:

$$\hat{\sigma}_{\alpha_k}(\tau_k) = \frac{2\hat{\sigma}_{A_k}(\tau_k)}{\tau_k}. \tag{2.13}$$

With $\hat{h}_k(\tau_k)$ in (2.12) and $\hat{\sigma}_{\alpha_k}(\tau_k)$ in (2.7), we determined that $\psi'_{16,k}(n)$ in (2.11) can be best approximated with a triangular wave described as follows:

$$\psi'_{16,k}(n) \approx \hat{\psi}_{16,k}(n) = \text{sgn}(y_{a,16}(n))\Lambda(n, \frac{2\hat{A}_k(\tau_k)}{\tau_k}, \tau_k) + \omega_\Lambda(n), \tag{2.14}$$

where

$$\omega_\Lambda(n) \sim N(0, \hat{\sigma}^2_{\alpha_k}(\tau_k)).$$

Reconstruction of Saturated Accelerometer Signals In this section, the estimated immeasurable signals $\hat{\Psi}_{16}(n)$ are used to reconstruct saturated accelerometer's readings $y_{a,16}(n)$ when saturated signals are detected. The detection of saturated signals can be achieved by comparing accelerometer's readings with a threshold, which has a value close to a nominal accelerometer's FSR. When a saturated signal is detected, the reconstructed accelerometer's readings, $\hat{y}_{a,16}(n)$, is expressed as follows:

$$\begin{aligned} \hat{y}_{a,16}(n) &= y_{a,16}(n) - \hat{\Psi}_{16}(n) \\ &= y_F(n) + b_a(n) + \Psi_{16}(n) - \hat{\Psi}_{16}(n) + \omega(n) \\ &= y_F(n) + b_a(n) + \hat{\omega}_{a,16}(n), \end{aligned} \tag{2.15}$$

where

$$\hat{\omega}_{a,16}(n) \sim N(0, \sigma^2_{\text{VRW}} + \hat{\sigma}^2_{\alpha_k}(\tau_k)).$$

Figure 2.11 exhibits examples of accelerometer signals reconstructed based on the artificially saturated accelerometer signals collected with the Analog Device IMU in the second series of experiments discussed in Section 2.4.1. To quantify the accuracy of reconstructed signals, the raw unsaturated accelerometer's signals were considered as the reference measurements.

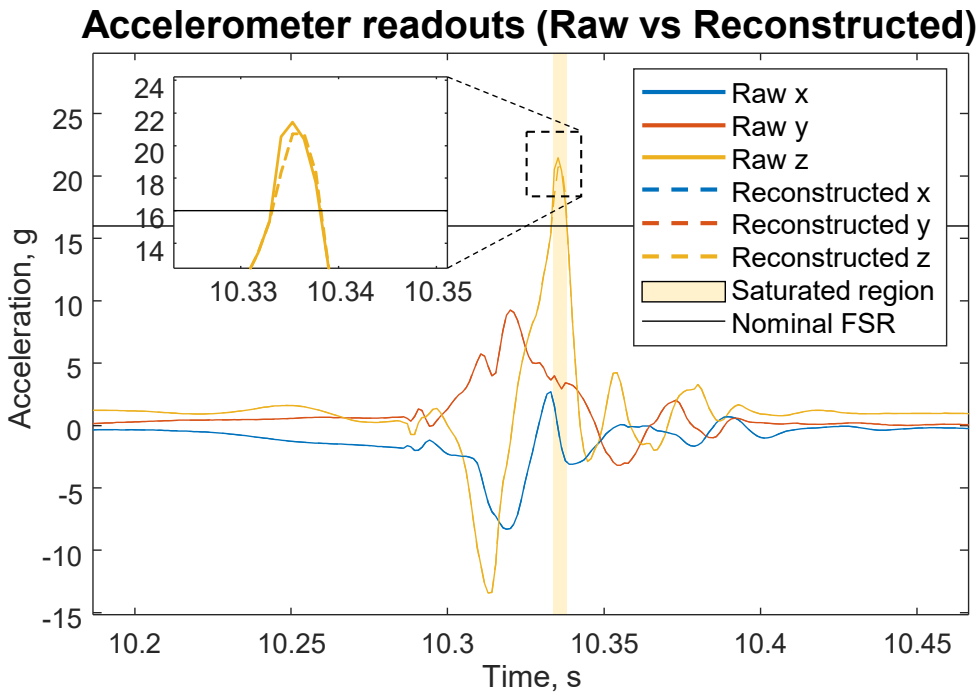


Figure 2.11: Examples of raw unsaturated accelerometer's measurements collected during the heel-strike phase and artificially saturated measurements that are reconstructed by the developed reconstruction filter.

Errors of the artificially saturated signals and the reconstructed signals with respect to the raw unsaturated measurements during the saturation period were calculated. Figure 2.12 shows the error distributions of the saturated accelerometer's signals and the reconstructed signals. Few things can be noted in Figure 2.12. First, the RMSE of the saturated signals was 4.63, while the RMSE of the reconstructed signals was decreased to 2.42. Second, the errors in the saturated signals had a bias of 2.97 g, while the bias was reduced to 0.507 g when the reconstruction filter was used. Third, the error distribution of the saturated signals was ill-fitted with a Gaussian distribution, indicating that inputting the saturated signals to the EKF of the ZUPT-aided INS would extensively violate the assumption of the EKF. After applying the reconstructed filter, it could be perceived that the shape distribution becomes closer to a Gaussian distribution. Based on the experimental result, we concluded that it was beneficial to apply our developed reconstruction filter in foot-mounted pedestrian inertial navigation to pre-process IMU readings, which were collected with an accelerometer

having an FSR of 16 g.

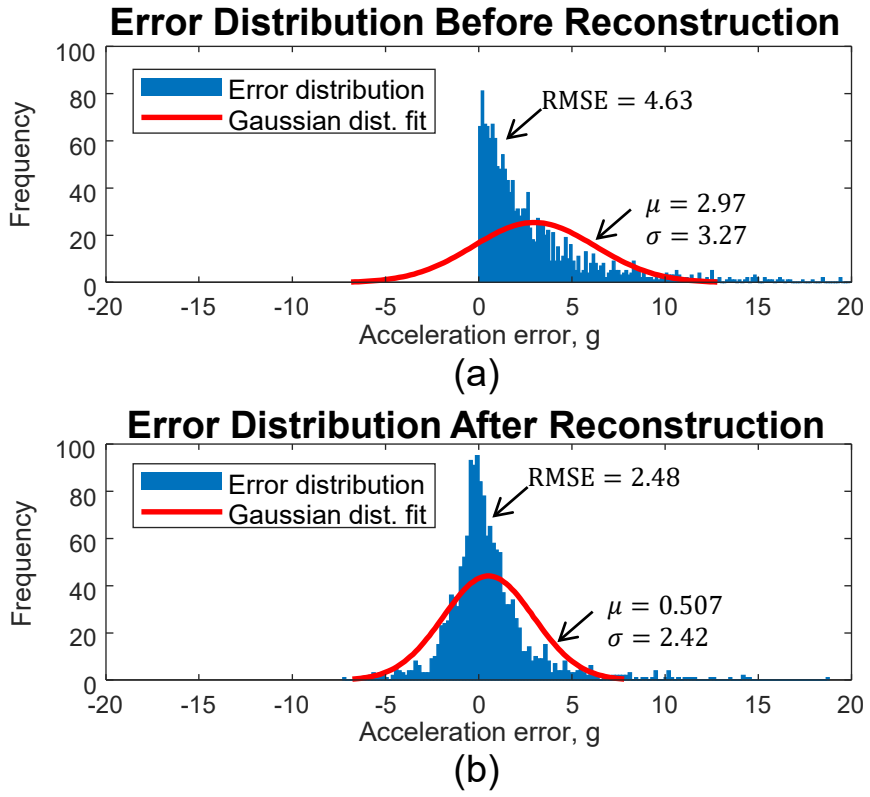


Figure 2.12: (a) Error distribution of artificially saturated accelerometer's readings discussed in Section 2.4.2. (b) Error distribution of the artificially saturated accelerometer's readings that were reconstructed by the developed reconstruction filter.

Use of the Reconstructing Filter With a ZUPT-aided INS

The navigation framework used to produce localization solutions is a ZUPT-aided INS augmented by the developed reconstruction filter, which is shown in Figure 2.13. In the developed framework, the ZUPT-aided INS is implemented with the SHOE detector using a constant threshold. Implementation of the ZUPT-aided INS was documented in detail in [224]. Before inputting measurements collected from a foot-mounted IMU into the ZUPT-aided INS, the developed framework first pre-process the IMU measurements with the developed reconstruction filter. The reconstruction filter outputs two pieces of information. The first piece is the pre-processed IMU measurements, and the second piece is the predicted

uncertainty for the pre-processed measurements. The pre-processed IMU measurements are inputted to the strapdown INS and the ZUPT algorithm. The predicted uncertainty is used to boost the process noise for accelerometer bias states in the EKF.

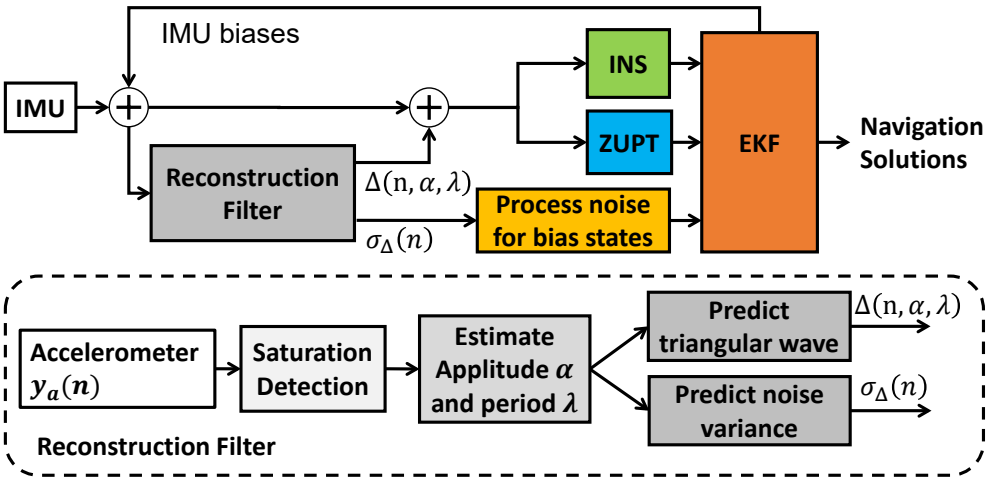


Figure 2.13: developed navigation framework.

2.4.3 Experimental Validation

We compared navigation performance of a traditional ZUPT-aided INS and the developed ZUPT-aided INS using the reconstruction filter based on the dataset collected by the two IMUs in the two series of experiments discussed in Section 2.4.1. In the experiments, the trajectories contained saturated sensor readings. Thresholds for stance phase detection in the ZUPT-aided INS were set to a similar value in both series of experiments, as the nominal walking paces were the same. The EKF noise parameters, including VRW σ_{VRW}, ARW σ_{ARW}, RRW σ_{RRW}, AcRW σ_{AcRW}, and measurement noise for zero-velocity measurements σ_{ZUPT} had values listed in Table 2.3. In each of the experiments, initial biases of gyroscopes were estimated by taking the average of gyroscope measurements collected in the first 20 seconds when the pedestrian was standing still on the ground. During the same period, initial biases of accelerometers were estimated by the ZUPT-aided INS. The estimated initial biases were removed from the IMU measurements collected in the rest of the timestamps. The initial

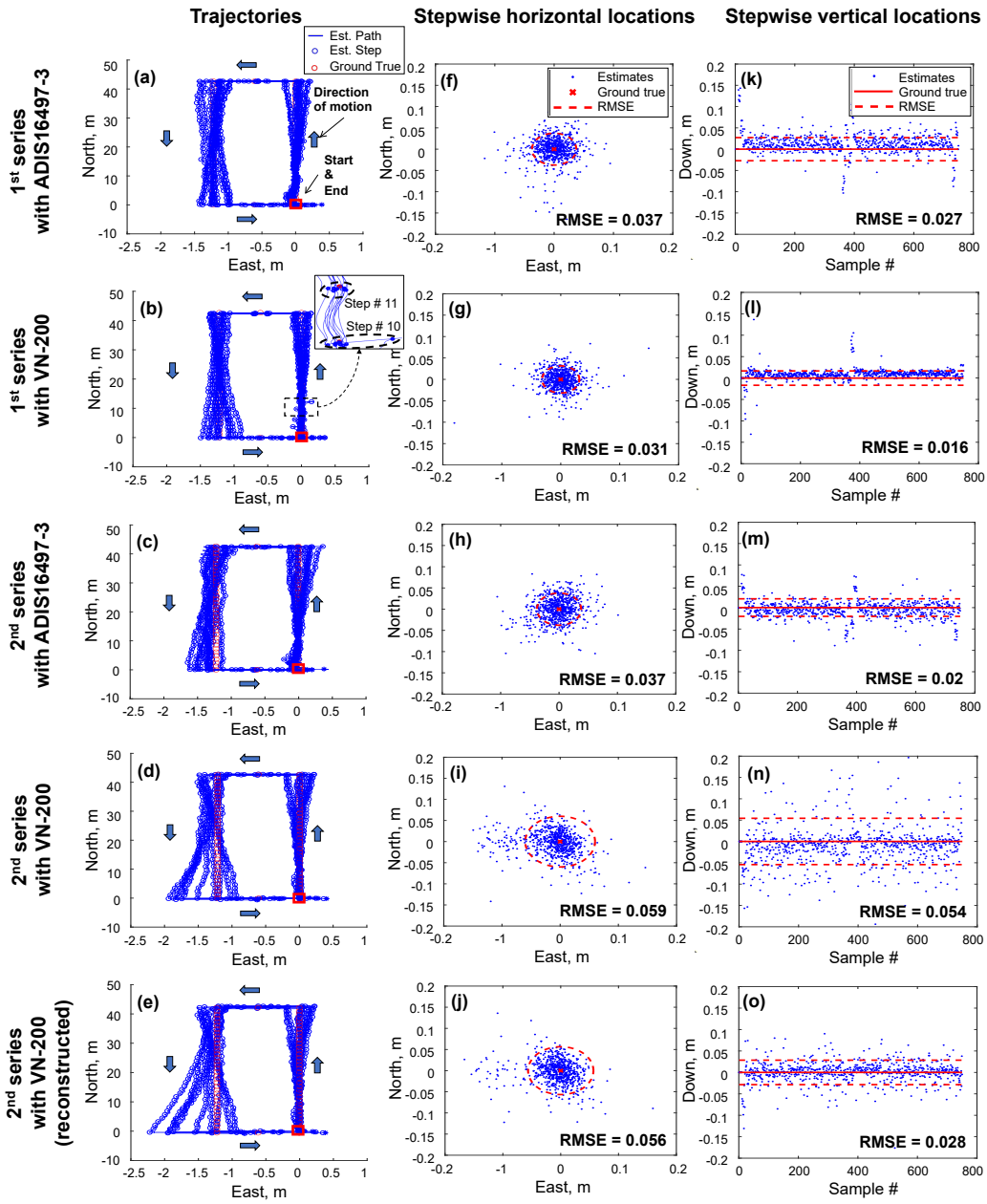


Figure 2.14: (a), (b), (c), and (d) display the ground truth and the trajectories estimated by the ZUPT-aided INS using measurements collected by ADIS16497−3 and VN−200 in the first series as well as ADIS16497−3 and VN−200 in the second series, respectively. (e) presents trajectories estimated by the ZUPT-aided INS using the developed reconstruction filter based on measurements collected by the VN−200 IMU in the second series of experiments. (f), (g), (h), and (i) present ground truth and the horizontal step-wise displacements obtained in the five approaches. The radiuses of the dashed red circles indicate the values of the corresponding horizontal step-wise Root Mean Square Errors (RMSEs). (k), (l), (m), (n), and (o) present ground truth and vertical displacement between two consecutive steps. The red dashed lines marked the vertical step-wise RMSEs. (f), (g), (h), (i), (j), (k), (l), (m), (n), and (o) contains exactly 740 blue points.

orientation of each experiment was determined by aligning the trajectory of the first ten steps with the north.

Table 2.3: EKF Parameter settings for the ZUPT-aided INS

EKF parameter	**Value**
σ_{ARW}	2.1597×10^{-5}
σ_{VRW}	4.8557×10^{-4}
σ_{RRW}	1.7141×10^{-6}
σ_{AcRW}	1.3873×10^{-6}
σ_{ZUPT}	0.02

In Figure 2.14(a), (b), (c), and (d), the blue marks show the ten trajectories estimated by the standalone ZUPT-aided INS with the two IMUs in the two series of experiments, respectively. Figure 2.14(e) presents trajectories estimated by the ZUPT-aided INS using the developed reconstruction filter based on measurements collected by the VN−200 IMU in the second series of experiments. The red circles in Figure 2.14(a), (b), (c), (d), and (e) indicate the true foot-landing location in each step. The ground truth information was measured by a ruler. In Figure 2.14(f), (g), (h), (i), and (j), each blue dot indicates a vector of the horizontal displacement between two consecutive steps subtracted by the ground true step-wise displacements. In Figure 2.14(k), (l), (m), (n), and (o), each blue dot marks the vertical displacement between two consecutive steps. Each plot of Figure 2.14(f), (g), (h), (i), (j), (k), (l), (m), (n), and (o) contains exactly 740 blue points. The RMSE of the step-wise displacements was computed. In Figure 2.14(f), (g), (h), (i), and (j), the radius of the dashed red circle is the step-wise RMSE value. In Figure 2.14(k), (l), (m), (n), and (o), two horizontal dashed lines represent the positive and the negative vertical step-wise RMSE. The horizontal and the vertical RMSEs are summarized in Table 2.4. In Table 2.4, the entries in the fourth row of the first two columns are filled as "not applicable" because no saturated measurements were detected in the first series of experiments.

A few phenomena can be noted in Figure 2.14.

Table 2.4: Step-wise displacement errors in terms of RMSE.

Deployed	**1st series**		**2nd series**	
sensor	Horizontal	Vertical	Horizontal	Vertical
ADIS16497−3	0.037	0.027	0.037	0.026
VN−200	0.038	0.016	0.059	0.054
VN−200 (Reconstructed)	N/A*	N/A*	0.056	0.028

*No saturated accelerometer's reading was detected.

1. It can be observed that the step-wise navigation errors using the two different IMUs in the first series of experiments were on a similar level. Although the Analog Device IMU and the VectorNav IMU have different noise performances, the difference was negligible in our opinion and would not lead to a different navigation accuracy in a pedestrian navigation task of around 120 seconds.

2. In the second series of experiments where the two IMUs experienced large shocks, the navigation errors of the ADIS16497−3 did not increase, while the error in the case of VN−200 were 1.34× and 3.37× larger along the horizontal and vertical directions, respectively. We considered that the increased errors were caused by not only insufficient accelerometer's FSR but also insufficient accelerometer's and gyroscope's bandwidths. The consideration of insufficient bandwidth as an error source was because large shocks that occurred during the heel-strike phases generated a series of high-frequency components. These high-frequency components might exceed the accelerometer's and gyroscope's bandwidths of the VN−200, which were 260 Hz and 256 Hz, respectively. Note that in the case of the ADIS16497−3, insufficient sensor bandwidth was not considered as a primary error source because the ADIS16497−3 had an accelerometer's and gyroscope's bandwidths of 750 Hz and 550 Hz.

3. When using our developed reconstruction filter to pre-process the saturated measurements collected by the VN−200 IMU, the RMSEs along horizontal and vertical directions were reduced by 5% and 50%, respectively. The improvements indicated that it

is beneficial to pre-process IMU measurements using the developed approach in this series of experiments. It is worth mentioning that the amount of improvements along the vertical direction was significantly larger than in the horizontal direction. In our opinion, the discrepancy was because of an observation that most of the saturated accelerometer's signals were along the z-axis. In the x-axis and the y-axis, the forces experienced by the IMU saturated accelerometer's readings only a few times. As a result, the developed reconstruction filter had limited improvements in the navigation accuracy along the horizontal direction in these experiments.

Based on the experimental results in these specific series of experiments, we demonstrated that the developed reconstruction filter is capable to mitigate the navigation errors due to insufficient accelerometer FSR for the ZUPT-aided INS.

This section presented a reconstruction filter for the ZUPT-aided INS. The reconstruction filter aims to rebuild the accelerometer's saturated readings due to large shocks during the heel-strike phase in a gait cycle of pedestrian navigation. This section experimentally illustrated that acceleration shocks experienced by foot-mounted IMUs are an error source when an insufficient accelerometer's FSR is used, and the error cannot be ignored in pedestrian navigation. To evaluate localization performance of a ZUPT-aided INS enhanced by the developed reconstruction filter, two series of indoor pedestrian navigation experiments were conducted with a VectorNac VN$-$200 IMU and an Analog Device ADIS16497$-$3 IMU, which have distinct accelerometer's FSRs. In the first series of experiments, the accelerometer's readings were not saturated, while in the second series of experiments, foot-mounted IMUs experienced large forces during heel-strike phases that saturated accelerometer's readings. The ZUPT-aided INS based on the two IMUs in the first series of experiments had similar navigation accuracy. In the second series, the navigation performance of the VN$-$200 reduced by 1.34$\times$ and 3.37$\times$ along the horizontal and the vertical directions, while the position accuracy using the ADIS16497$-$3 remained on the same level as in the first series. When ap-

plying our developed reconstruction filter to the saturated accelerometer's measurements, the navigation accuracy along horizontal and vertical directions was increased by 5% and 50%, respectively. Two conclusions were drawn from the experimental results. First, the decreased performance of the VN−200 emphasized the importance of matching sensors characteristics to application requirements and, as an example, illustrated how the mismatch in the FSR of the accelerometer could degrade the performance of pedestrian inertial navigation. Second, it is beneficial to pre-process IMU measurements using the developed reconstruction in the indoor pedestrian experiments, especially when sensors with required characteristics are not available. The developed approach is general and can be applied in other pedestrian navigation scenarios and applications. The results presented in this section have been published in [87].

2.5 System-Level Enhancement Using Prioritizable IMU

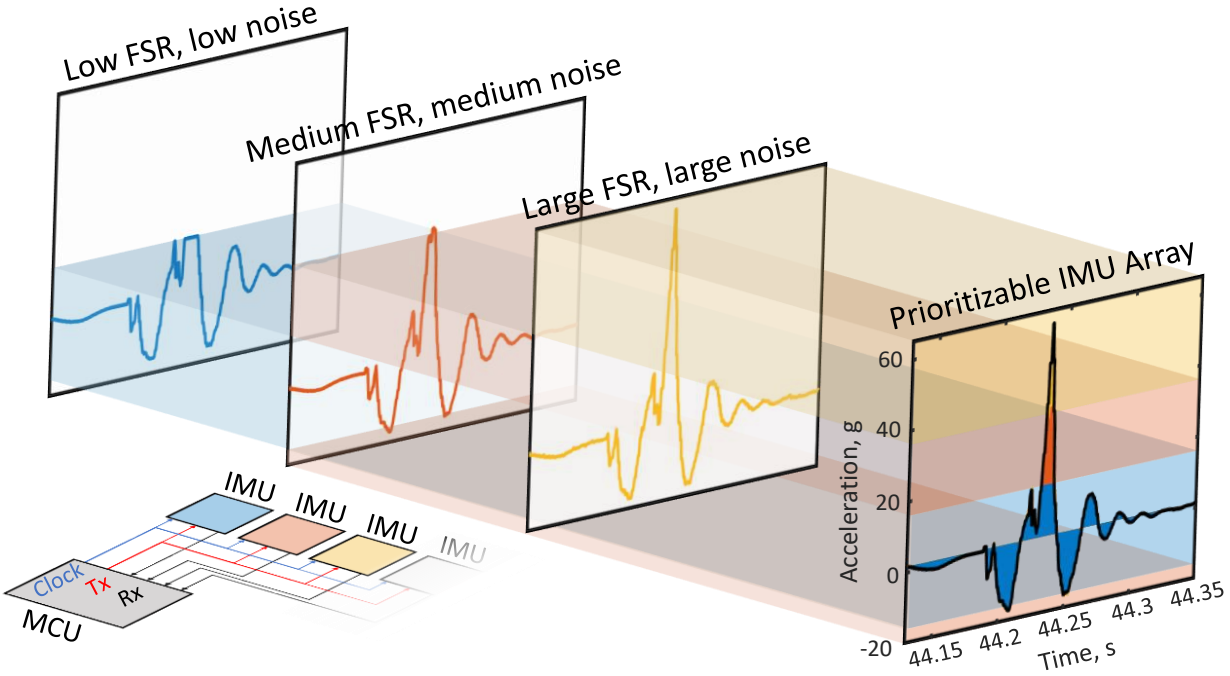


Figure 2.15: Concept of the developed Prioritizable IMU array (Prio-IMU).

This section reports a prioritizable IMU array (Prio-IMU), a systematic approach utilizing multiple different IMUs to mitigate the impact of insufficient sensor FSR and bandwidth on ZUPT-aided INS using foot-mounted IMUs. Figure 2.15 illustrates the concept of the developed Prio-IMU. The Prio-IMU integrates readings from multiple IMUs, each with different sensor FSRs and noise characteristics. The approach utilizes the properties that an IMU with good noise characteristics usually comes with a trade-off of low sensor FSR and bandwidth, and vice versa. In scenarios when the system experiences large accelerations and angular velocities, utilizing a sensor with great noise performance but insufficient FSR could lead to a larger navigation error, as compared to a sensor with poor noise performance but sufficiently high FSR.

In the developed Prio-IMU, each IMU in the system needs to be aligned to a universal coordinate system so that all the units measure similar physical quantities of accelerations and angular velocities. This section describes a sensor model, discusses the alignment of multiple IMUs, and presents a mechanism to prioritize the usage of different IMUs.

2.5.1 Measurement Model for Multiple Inertial Sensors

The developed Prio-IMU considers N calibrated IMUs mounted on different locations of a rigid body, such as a PCB. An IMU calibration process includes identifying errors in the sensor's scale factors, cross-axis sensitivities, and turn-on biases, and the procedure could be done with an estimation algorithm using self-measurements [29] or through external equipment, such as a shaker or a rate table [222]. The ith IMU of the Prio-IMU is characterized by eight different metrics, including accelerometer's FSR, $\mathrm{F}_{\mathrm{a}}^{i}$, bandwidth, $\mathrm{B}_{\mathrm{a}}^{i}$, VRW, $\sigma_{\mathrm{a,N}}^{i}$, and bias instability, $\sigma_{\mathrm{a,B}}^{i}$, and gyroscope's FSR, $\mathrm{F}_{\mathrm{g}}^{i}$, bandwidth, $\mathrm{B}_{\mathrm{g}}^{i}$, ARW, $\sigma_{\mathrm{g,N}}^{i}$, and bias instability, $\sigma_{\mathrm{g,B}}^{i}$. The N IMUs can be chosen such that the characterization metrics satisfy

the following conditions:

$$\forall i > 0 \text{ and } i < j < N, \mathrm{F}_{\mathrm{a}}^{i} \leq \mathrm{F}_{\mathrm{a}}^{j}, \mathrm{B}_{\mathrm{a}}^{i} \leq \mathrm{B}_{\mathrm{a}}^{j}, \sigma_{\mathrm{a,N}}^{i} \leq \sigma_{\mathrm{a,N}}^{j},$$
$$\sigma_{\mathrm{a,B}}^{i} \leq \sigma_{\mathrm{a,B}}^{j}, \mathrm{F}_{\mathrm{g}}^{i} \leq \mathrm{F}_{\mathrm{g}}^{j}, \mathrm{B}_{\mathrm{g}}^{i} \leq \mathrm{B}_{\mathrm{g}}^{j}, \sigma_{\mathrm{g,N}}^{i} \leq \sigma_{\mathrm{g,N}}^{j}, \sigma_{\mathrm{g,B}}^{i} \leq \sigma_{\mathrm{g,B}}^{j}.$$

The Prio-IMU produces a single measurement vector at time k, denoted as $\mathbf{u}_k$, by prioritizing the measurements collected by one of the IMUs integrated into the system. The prioritization mechanism is discussed in Section 2.5.3. A Prio-IMU measurement vector $\mathbf{u}_k$ includes accelerometer readings, $\mathbf{a}_k$, and gyroscope readings, $\boldsymbol{\omega}_k$, along the three axes. $\mathbf{a}_k$ and $\boldsymbol{\omega}_k$ are modeled as follows:

$$\mathbf{a}_k = \bar{\mathbf{a}}_k + \bar{\mathbf{b}}_{\mathrm{a},k} + \mathbf{n}_{\mathrm{a},k}, \boldsymbol{\omega}_k = \bar{\boldsymbol{\omega}}_k + \bar{\mathbf{b}}_{\mathrm{g},k} + \mathbf{n}_{\mathrm{g},k}, \tag{2.16}$$

where $\bar{\mathbf{a}}_k$ and $\bar{\boldsymbol{\omega}}_k$ are the true accelerations and angular velocities that are not measurable, $\bar{\mathbf{b}}_{\mathrm{a},k}$ and $\bar{\mathbf{b}}_{\mathrm{g},k}$ are unknown accelerometer and gyroscope time-varying stochastic biases, and $\mathbf{n}_{\mathrm{a},k}$ and $\mathbf{n}_{\mathrm{g},k}$ are accelerometer and gyroscope white noise components, modeled as zero-mean Gaussian with standard deviations of $\sigma_{\mathrm{a,N},k}$ and $\sigma_{\mathrm{g,N},k}$, respectively.

2.5.2 Alignment of Multiple Inertial Sensors

This study denotes $\mathbf{u}_k^{ii}$ as a measurement vector collected by the ith calibrated IMU at time k and expressed in the sensor's own body frame. $\mathbf{u}_k^{ii} = \left[\mathbf{a}_k^{ii}, \boldsymbol{\omega}_k^{ii}\right]^\top$, where $\mathbf{a}_k^{ii}$ and $\boldsymbol{\omega}_k^{ii}$ represent accelerometer and gyroscope readings along the three axes, respectively. The acceleration and angular velocity measured by the ith IMU can also be expressed in the body frame of the jth IMU, denoted as $\mathbf{u}_k^{ij} = \left[\mathbf{a}_k^{ij}, \boldsymbol{\omega}_k^{ij}\right]^\top$.

Angular rates of $\boldsymbol{\omega}_k^{ii}$ and $\boldsymbol{\omega}_k^{ij}$ have the following relationships:

$$\boldsymbol{\omega}_k^{ij} = \mathbf{T}_i^j \boldsymbol{\omega}_k^{ii}, \tag{2.17}$$

where $\mathbf{T}_i^j$ is a DCM transforming the body frame of ith IMU to the body frame of the jth IMU. Acceleration of $\mathbf{a}_k^{ii}$ and $\mathbf{a}_k^{ij}$ have the following relationships:

$$\mathbf{a}_k^{ij} = \mathbf{T}_i^j \mathbf{a}_k^{ii} + [\boldsymbol{\omega}_k^{ij}]_\times ([\boldsymbol{\omega}_k^{ij}]_\times \mathbf{r}_i^j) + [\dot{\boldsymbol{\omega}}_k^{ij}]_\times \mathbf{r}_i^j. \tag{2.18}$$

In (2.18), $\dot{\boldsymbol{\omega}}_k^{ij}$ is angular acceleration, $[\mathbf{x}]_\times$ represents the skew-symetric matrix of a vector $\mathbf{x}$, and $\mathbf{r}_i^j$ represents the position of the ith IMU in the body frame of the jth IMU. The terms $[\boldsymbol{\omega}_k^{ij}]_\times ([\boldsymbol{\omega}_k^{ij}]_\times \mathbf{r}_i^j)$ and $[\dot{\boldsymbol{\omega}}_k^{ij}]_\times \mathbf{r}_i^j$ in (2.18) correspond to the centrifugal force and the Euler force, respectively.

The DCM $\mathbf{T}_i^j$ and the position vector $\mathbf{r}_i^j$ in (2.17) and (2.18) are unknown and assumed to be time-independent values. In this study, we followed the estimation algorithm discussed in [185] to determine the relative geometry $\mathbf{T}_i^j$ and $\mathbf{r}_i^j$ between two IMUs. Additionally, the angular acceleration $\dot{\boldsymbol{\omega}}_k^{ij}$ is calculated by taking the difference between two consecutive gyroscope measurements. That is, $\dot{\boldsymbol{\omega}}_k^{ij} = (\boldsymbol{\omega}_k^{ij} - \boldsymbol{\omega}_{k-1}^{ij})/\mathrm{dt}$, where dt is the sampling rate of the IMU. In this study, we aligned all IMUs to the body frame of the 1st IMU, which has the best noise performance and lowest FSR and bandwidth.

2.5.3 Prioritization Mechanism

At time k, the developed Prio-IMU chooses the accelerometer measurements collected by the IMU with the best noise performance among the IMUs that do not have saturated accelerometer measurements. The accelerometer white noise component $\mathbf{n}_{\mathrm{a},k}$ in (2.16) follows the noise characteristics of the chosen IMU. The choice mechanism can be mathematically

expressed as follows:

$$\mathbf{a}_k = \mathbf{a}_k^{n1}, \sigma_{\mathrm{a,N},k} = \sigma_{\mathrm{a,N}}^{n}, \tag{2.19}$$

where

$$n = \min\{j \mid \forall 1 \leq i < j \leq N, |\mathbf{a}_k^{i1}| \geq \mathrm{F}_{\mathrm{a}}^{i} \text{ and } |\mathbf{a}_k^{i1}| < \mathrm{F}_{\mathrm{a}}^{i}\}.$$

With the nth IMU being chosen, this study also estimates the time-varying accelerometer biases $\bar{\mathbf{b}}_{\mathrm{a},k}$ in (2.16). The estimated bias, denoted as $\mathbf{b}_{\mathrm{a},k}$, is updated at each time step as

$$\mathbf{b}_{\mathrm{a},k} = \mathbf{b}_{\mathrm{a},k}^{n1}, \tag{2.20}$$

where $\mathbf{b}_{\mathrm{a},k}^{ij}$ represents the estimated accelerometer stochastic time-varying bias of the ith IMU expressed in the body frame of the jth IMU. In our Prio-IMU, the stochastic biases $\mathbf{b}_{\mathrm{a},k}^{ij}$ of the accelerometer of the ith IMU are estimated based on unsaturated accelerometer measurements collected by the IMU with the best noise performance. At each timestamp k, $\mathbf{b}_{\mathrm{a},k}^{ij}$ is estimated as follows:

$$\mathbf{b}_{\mathrm{a},k}^{ij} = \mathbf{a}_{k-1}^{ij} - (\mathbf{a}_{k-1}^{lj} - \mathbf{b}_{\mathrm{a},k-1}^{lj}), \tag{2.21}$$

where lth IMU is chosen such that

$$l = \min\{m \mid \forall 1 \leq m \leq i, |\mathbf{a}_{k-1}^{mm}| \leq \mathrm{F}_{\mathrm{a}}^{m}\}.$$

A result of (2.21) is that $\mathbf{b}_{\mathrm{a},k}^{11} = \mathbf{0}$. This result was intended as the 1st IMU of the Prio-IMU has the lowest bias instability, and the available information from the other IMUs does not allow a more accurate estimation of the bias than zero.

The gyroscope readings of the developed Prio-IMU, $\boldsymbol{\omega}_k$, are obtained with similar procedures discussed in (2.19)-(2.21).

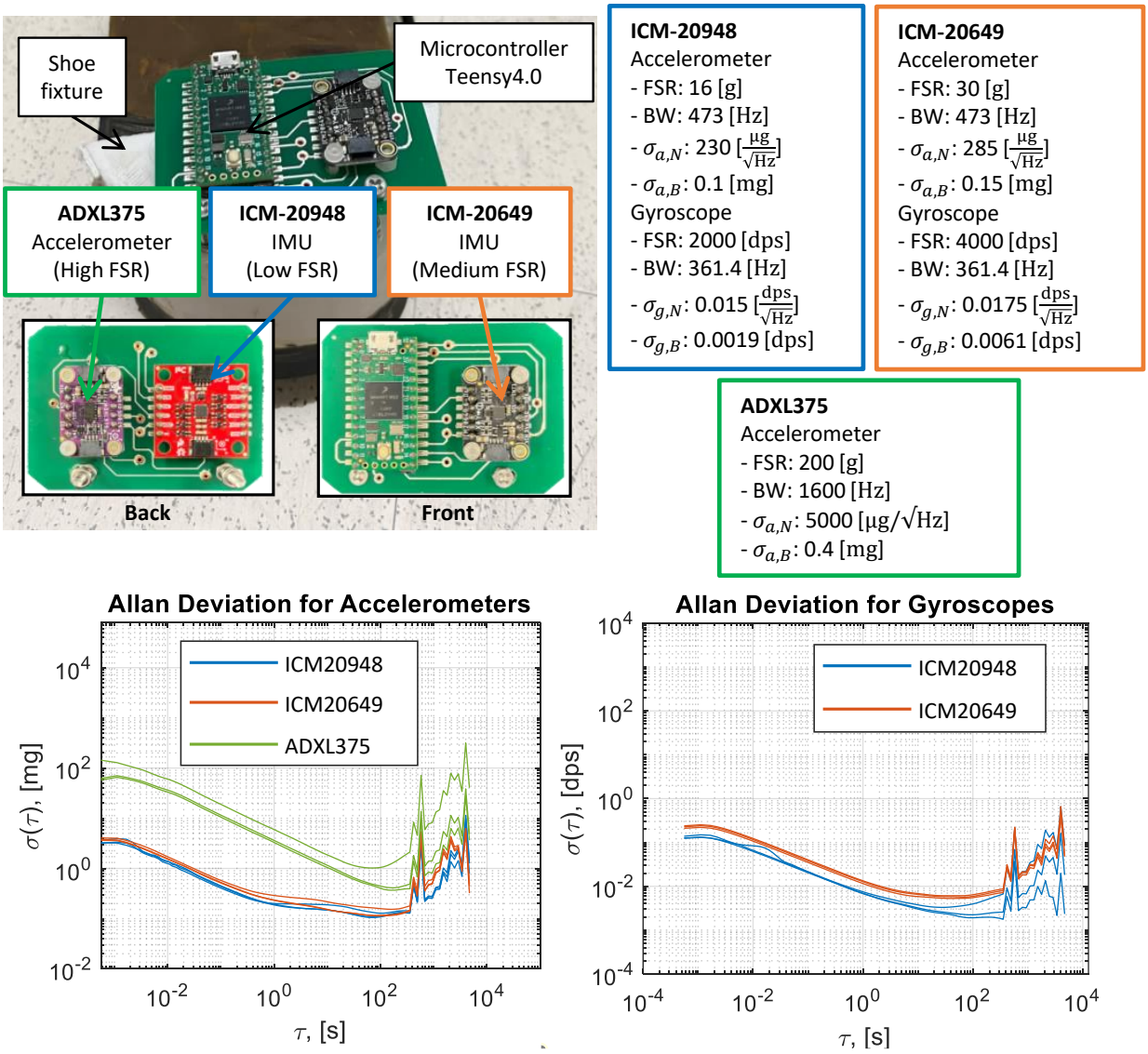


Figure 2.16: A prototype of the developed Prio-IMU and the characteristics of the deployed sensors.

2.5.4 Experimental Validation

Experimental Setup

To demonstrate the developed Prio-IMU, we developed a Prio-IMU prototype, shown in Figure 2.16. The current implementation of the system integrates a Teensy 4.0 microcontroller with an ICM−20948 6-DoF IMU, an ICM−20649 6-DoF IMU, and a 3-DoF ADXL375 accelerometer. Serial Peripheral Interface (SPI) communication protocol was used to communicate with all three sensors, and the sampling rate of the system was programmed at 1800 [Hz]. We experimentally characterized the three sensors, and the characteristics and the Allan deviation plots of each sensor are shown in Figure 2.16.

Figure 2.17 presents profiles of accelerometer and gyroscope measurements collected by the Prio-IMU prototype during the heel-strike phase of a running experiment. We could observe that the accelerometers of both ICM−20948 and ICM−20649 were saturated while the measurements of the ADXL375 were below the sensor accelerometer FSR of 200 [g]. It was also observed in Figure 2.17 that the gyroscope measurements of the ICM−20948 were saturated at 2000 [dps] while the measurements of ICM−20649 peaked at around 2600 [dps].

Experimental Results

To validate the navigation performance of the developed Prio-IMU, we conducted a series of 10 sets of pedestrian indoor navigation experiments at the University of California, Irvine. In each trial, a subject first walked a straight line for around 45 [m] at a pace of around 80 [steps per minute (spm)] and then ran a straight line for 42.8 [m] at a pace of around 180 [spm]. The total trajectory length was 87.8 [m]. We compared the navigation performance of the ZUPT-aided INS using four different configurations of the Prio-IMU prototypes. Each configuration used a unique combination of the three inertial sensors shown in Figure 2.16.

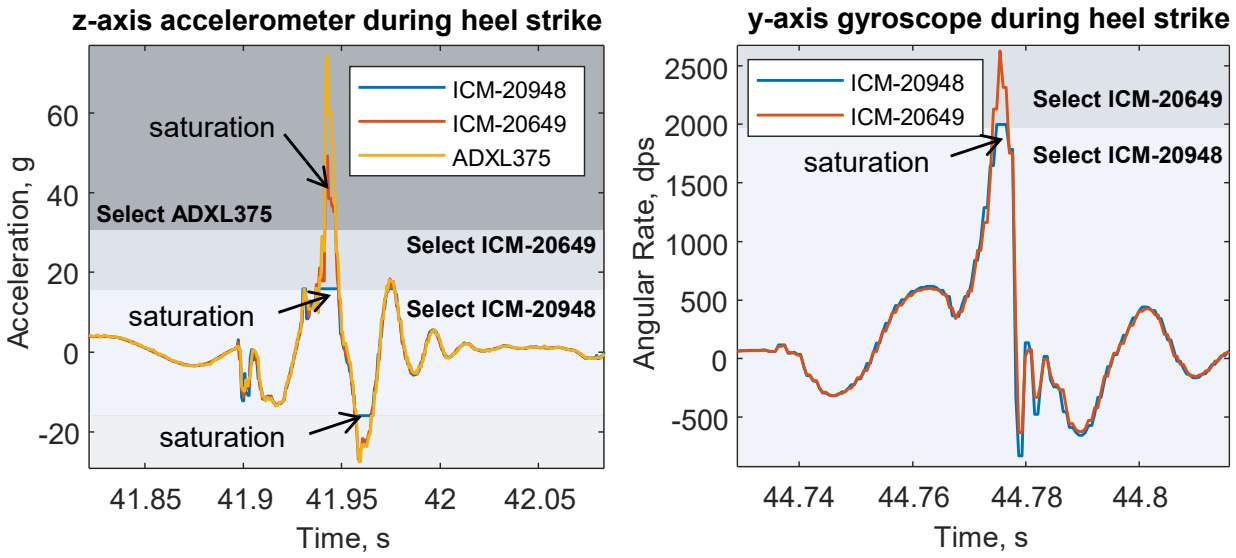


Figure 2.17: Profiles of accelerometer and gyroscope measurements collected by the Prio-IMU prototype.

In this study, we implemented the ZUPT-aided INS in an EKF framework with the SHOE detector [83, 181]. Two fixed thresholds for the SHOE detector were determined, one for the case of walking and the other for running, such that the navigation errors were minimized.

Figure 2.18 presents the experimental results using the four different configurations. We used the horizontal RMSEs (2D RMSE), CEPs, and vertical ($\perp$) RMSEs to evaluate each navigation solution. We could observe that Configuration 4, where all three sensors on the Prio-IMU prototype were used, had the minimum navigation errors, as compared to the other configurations. The experimental results proved that it is beneficial to use the developed Prio-IMU to improve navigation accuracy in the case of foot-mounted IMUs.

Two remarks can be made on the developed Prio-IMU prototype. First, the quality of the Prio-IMU measurements was sensitive to errors in alignments of multiple IMUs. In our current approach, aligning the accelerometers of different IMUs, as discussed in (2.18), involved compensation of the centrifugal force and the Euler force, which required information of relative position vectors between each integrated IMU. The positions were results of algorithmic estimation with uncertainties. Moreover, the angular accelerations in (2.18) were

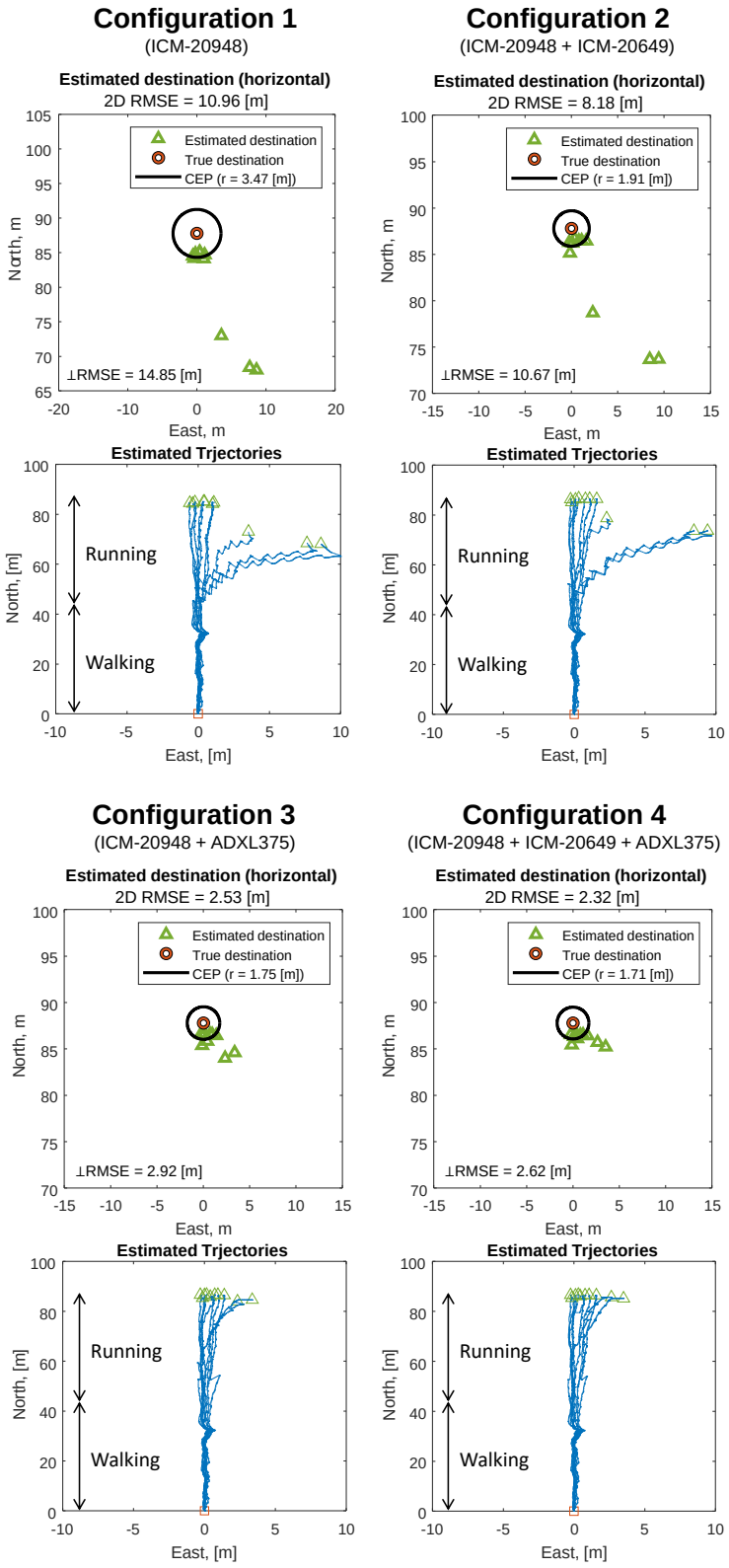


Figure 2.18: Estimated trajectories of the experiments.

derived from gyroscope measurements by taking the derivative, which could introduce high-frequency noise components. One approach to reducing errors introduced by IMU alignment is to minimize displacements between each inertial sensor, and this could potentially be achieved through micro-fabrication technology. Second, the three inertial sensors integrated into the Prio-IMU prototype shown in Figure 2.16 were chosen with a consideration of flexible development. The choice of sensors could be refined by not only increasing the FSR and bandwidth but also optimizing the noise performance of a particular axis of an accelerometer or gyroscope. For example, integrating an ultra-low-noise z-axis gyroscope could reduce the unobservable yaw angle errors in the ZUPT-aided INS, increasing the long-term navigation accuracy.

2.6 Conclusion

This chapter presented two approaches, an algorithmic reconstruction filter and a prioritizable IMU array, to mitigate the problem of insufficient sensor FSR and bandwidth. The developed reconstruction filter, designed for a single IMU, used a Gaussian Process Regression (GPR) to predict profiles of accelerometer signals exceeding the sensor's FSR based on saturation periods and compensate saturated accelerometer signals with the predicted profiles. The prioritizable IMU array integrated multiple IMUs with different specifications and prioritized the usages of the different sensors in different scenarios. The two developed approaches were separately verified with experiments to improve the navigation accuracy of the ZUPT-aided INS in non-walking scenarios, as compared to the traditional ZUPT-aided INS.

Chapter 3

On Motion Sensor – Mitigating Thermal-Induced Errors

3.1 Introduction

This chapter presents an algorithmic approach to compensate thermal-induced errors of inertial sensors for ZUPT-aided INS. The chapter is organized as follows. Section 3.2 discusses a developed thermal-compensation approach, Section 3.3 presents experimental results validating the developed approach, and Section 3.2 summarizes and concludes the results presented in this chapter.

3.2 Thermal Compensation Using a Neural Network

This section discusses the sensor measurement model, thermal-induced errors due to temperature and temperature rate, details of the developed BPNN-based thermal compensation approach, and a temperature-compensated ZUPT-aided INS.

3.2.1 Sensor Measurement Model

6-axis IMU measurements, including accelerometer and gyroscope readings along the three axes collected directly from a COTS sensor at time k, are denoted as $\bar{\mathbf{u}}(k)$. The IMU measurements are modeled as follows:

$$\bar{\mathbf{u}}(k) = \mathbf{u}(k) + \mathbf{b}_{\text{turn-on}} + \mathbf{b}_{\text{in-run}}(k) + \mathbf{b}_{\text{T}}(T, \Delta T) + \tilde{\mathbf{n}}_{\text{T}}(k), \tag{3.1}$$

$$\tilde{\mathbf{n}}_{\text{T}}(k) \sim \text{N}(\mathbf{0}, \boldsymbol{\sigma}_{\text{T}}^2(T, \Delta T)\mathbf{I}). \tag{3.2}$$

Here, $\mathbf{u}(k)$ is uncontaminated IMU measurements that contain the actual acceleration and angular velocity, which are not available from the sensor. $\mathbf{b}_{\text{turn-on}}$ is the turn-on bias of an IMU. Gyroscope components of $\mathbf{b}_{\text{turn-on}}$ can be estimated by taking the average of measurements collected from a stationary IMU, and accelerometer components can be estimated by applying a ZUPT algorithm on the stationary measurements [69]. $\mathbf{b}_{\text{in-run}}(k)$ denotes IMU in-run biases, which are considered unpredictable in this section. T and ΔT are the measured temperature and temperature rate with respect to time. $\mathbf{b}_{\text{T}}(T, \Delta T)$ is thermal-induced biases of accelerometer and gyroscope along the 3 axes. $\tilde{\mathbf{n}}_{\text{T}}(k)$ is white Gaussian noise components that have a temperature-dependent covariance matrix $\boldsymbol{\sigma}_{\text{T}}^2(T, \Delta T)$. $\mathbf{I}$ is an identity matrix.

3.2.2 Thermal-induced Errors

This section considers 12 different thermal-induced errors, including bias drifts and noise SD variations of accelerometers and gyroscopes along the three axes. To understand the characteristics of these errors, an experiment was conducted with a setup shown in Figure 3.1(a). In order to guarantee that the IMU experienced minimal motion, we removed the IMU, the Thermal Electric Cooler (TEC), and the heat sink from the shoe shown in Figure 3.1, firmly mounted the IMU on a laboratory table, and attached the TEC and the heat sink on top

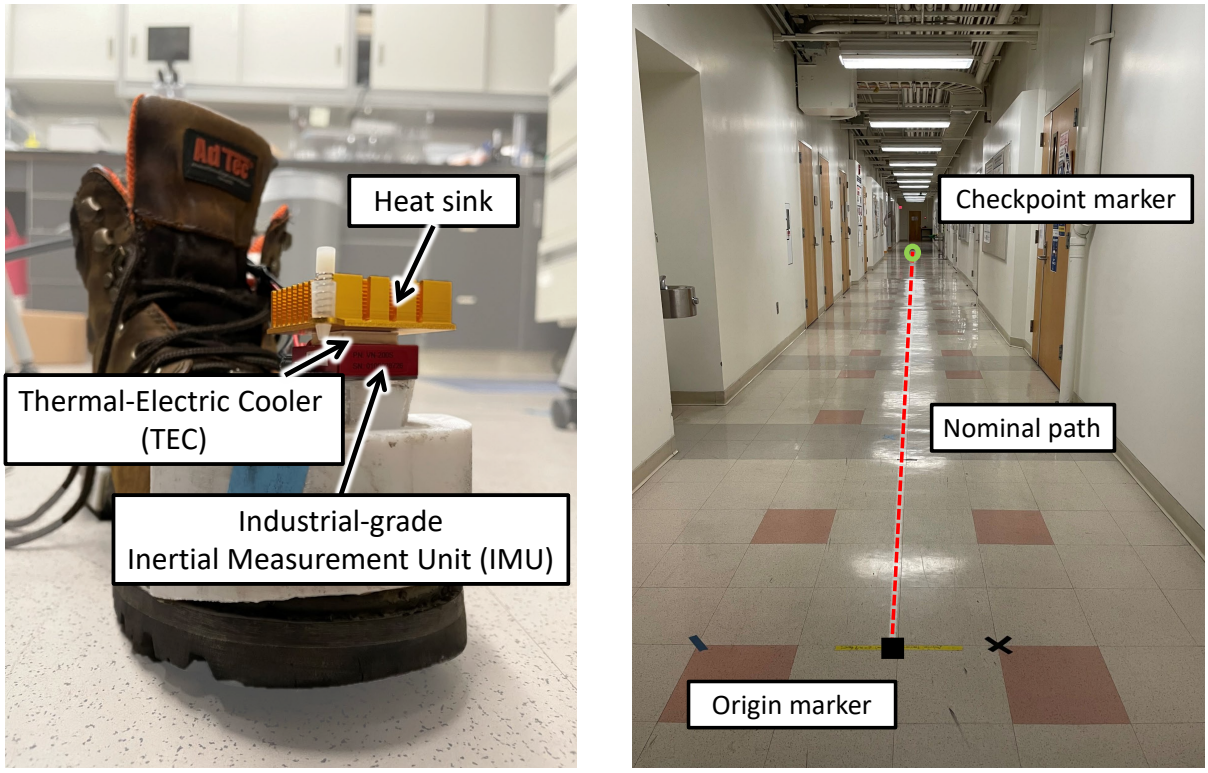


Figure 3.1: (a) Experimental setup and (b) experimental scenario used in the experiments described in subsection 3.3.

of the IMU. The sampling rate of the IMU was set to 800 Hz. In the experiment, ambient temperature around the IMU varied between 20°C to 50°C at a rate between −0.2°C/s to 0.2°C/s. Note that these values were measured by the thermometer of the IMU. The IMU was stationary throughout the roughly 3.3 hour long experiment.

In the experiment, measurements of thermal-dependent bias drifts, denoted as $\bar{\mathbf{b}}_{\mathrm{T}}(k)$, and measurements of noise SD variations, denoted as $\bar{\boldsymbol{\sigma}}^2_{\mathrm{T}}(k)$, were collected as follows:

$$\bar{\mathbf{b}}_{\mathrm{T}}(k) = \mathrm{movmean}(\bar{\mathbf{u}}(k) - \hat{\mathbf{g}} - \hat{\mathbf{b}}_{\text{turn-on}}) \tag{3.3}$$

$$\bar{\boldsymbol{\sigma}}^2_{\mathrm{T}}(k) = \mathrm{movvar}(\bar{\mathbf{u}}(k) - \hat{\mathbf{b}}_{\text{turn-on}} - \bar{\mathbf{b}}_{\mathrm{T}}(k)). \tag{3.4}$$

In (3.3) and (3.4), movmean() is the moving average function and movvar() is the moving variance function. This section uses a window size of 12.5 s for both movmean() and movvar(). The window size was determined based on the average integration periods where the IMU reached its noise floors. $\hat{\mathbf{g}}$ is the estimated gravity vector that was computed by taking average on static accelerometer readings.

The 12 thermal-dependent errors were collected based on (3.3) and (3.4). In this section, only dataset of the thermal-induced error for the z-axis gyroscope bias is presented, as this error component tends to have the largest impact on the performance of a ZUPT-aided INS implemented in the EKF. For the other 11 errors, their corresponding data are not shown here and will be presented in future research work on this topic. Figure 3.2 shows the measured and predicted z-axis gyroscope biases versus temperature and temperature rate. The measured gyroscope biases are colored in blue in Figure 3.2. Discussion regarding the predicted results will be presented in subsection 3.2.3. In Figure 3.2(a), we could observe that the collected biases were correlated with both temperature and temperature rate. In Figure 3.2(c), it appeared that the trend of the bias drifts during the heating process and the cooling process were different. Moreover, the bias profiles corresponding to different heating rates

were distinct, and the same phenomenon can also be observed during the cooling processes. These trends led to the appearance of hysteresis effects in the bias-temperature relation plot in Figure 3.2(b).

3.2.3 Back-Propagation Neural Network

This section considers the 12 thermal-induced errors as 12 different unknown functions, six of which corresponding to biases, denoted as $\mathbf{b}_{\mathrm{T}}(T, \Delta T)$, and the other six of which corresponding to noise SD variations, denoted as $\boldsymbol{\sigma}_{\mathrm{T}}^2(T, \Delta T)$. Our developed temperature compensation approach aims to approximate the unknown functions of $\mathbf{b}_{\mathrm{T}}(T, \Delta T)$ and $\boldsymbol{\sigma}_{\mathrm{T}}^2(T, \Delta T)$ with $\hat{\mathbf{b}}_{\mathrm{T}}(T, \Delta T)$ and $\hat{\boldsymbol{\sigma}}_{\mathrm{T}}^2(T, \Delta T)$ that have feedforward Neural Network (NN) structures using a 2×1 input feature vector consisting of temperature measurement and temperature rate. We trained 12 different BPNNs to separately predict the 12 thermal-induced errors. For each of the 12 BPNNs, a collected input feature vector is labeled with a corresponding thermal-induced error. For example, in the BPNN for the z-axis gyroscope bias, the feature vector was labeled with a collected z-axis gyroscope bias measurement. The BPNNs used in this section utilized three layers: an input layer, a hidden layer, and an output layer. In the training step of the BPNNs, we utilized the Levenberg-Marquardt (LM) backpropagation approach [241] to train the NN based on features and labels obtained using (3.3) and (3.4) for each of the 12 thermal-dependent errors. The training data contained around 9.8 million data points, and the dataset was randomly divided as 70% for training, 15% for verification, and 15% for testing. Mean Squared Error (MSE) was used as the performance evaluation. Hyperbolic tangent was selected as associated activation functions. In each of the BPNNs, the hidden layer had five neurons. This configuration was chosen because using a higher number of neurons only leads to marginally improved MSE during the BPNN verification process in this experiment.

Figure 3.2 demonstrates z-axis gyroscope biases predicted using the trained BPNN corresponding to the z-axis gyroscope bias. We can observe in Figure 3.2 that the prediction results, colored in orange, were visually aligned with the measured gyroscope biases, colored in blue, indicating that the trained BPNN could describe the nonlinear and the hysteresis effects appeared in Figure 3.2(b). In this experiment, the measured gyroscope biases had a RMSE of 0.343 degrees per second (dps). When the predicted z-axis gyroscope biases were subtracted from the measured gyroscope biases, the RMSE was significantly reduced to 0.024 dps.

3.2.4 Thermal-compensated ZUPT-aided INS

In our developed approach, the 12 thermal-induced errors predicted by the 12 trained BPNNs are inputted to the ZUPT-aided INS for temperature compensation. Figure 3.3 illustrates the thermal-compensated ZUPT-aided INS. In this approach, IMU biases predicted by the 6 BPNNs for biases are removed from raw measurements collected from an IMU. The predicted noise SD variations corresponding to gyroscopes and accelerometers are divided by the nominal ARW and VRW, respectively. These quotients are inputted to a zero-velocity detector and the EKF to vary the entries in the covariance matrix corresponding to orientation and velocity states. This section used the SHOE detector for zero-velocity detection.

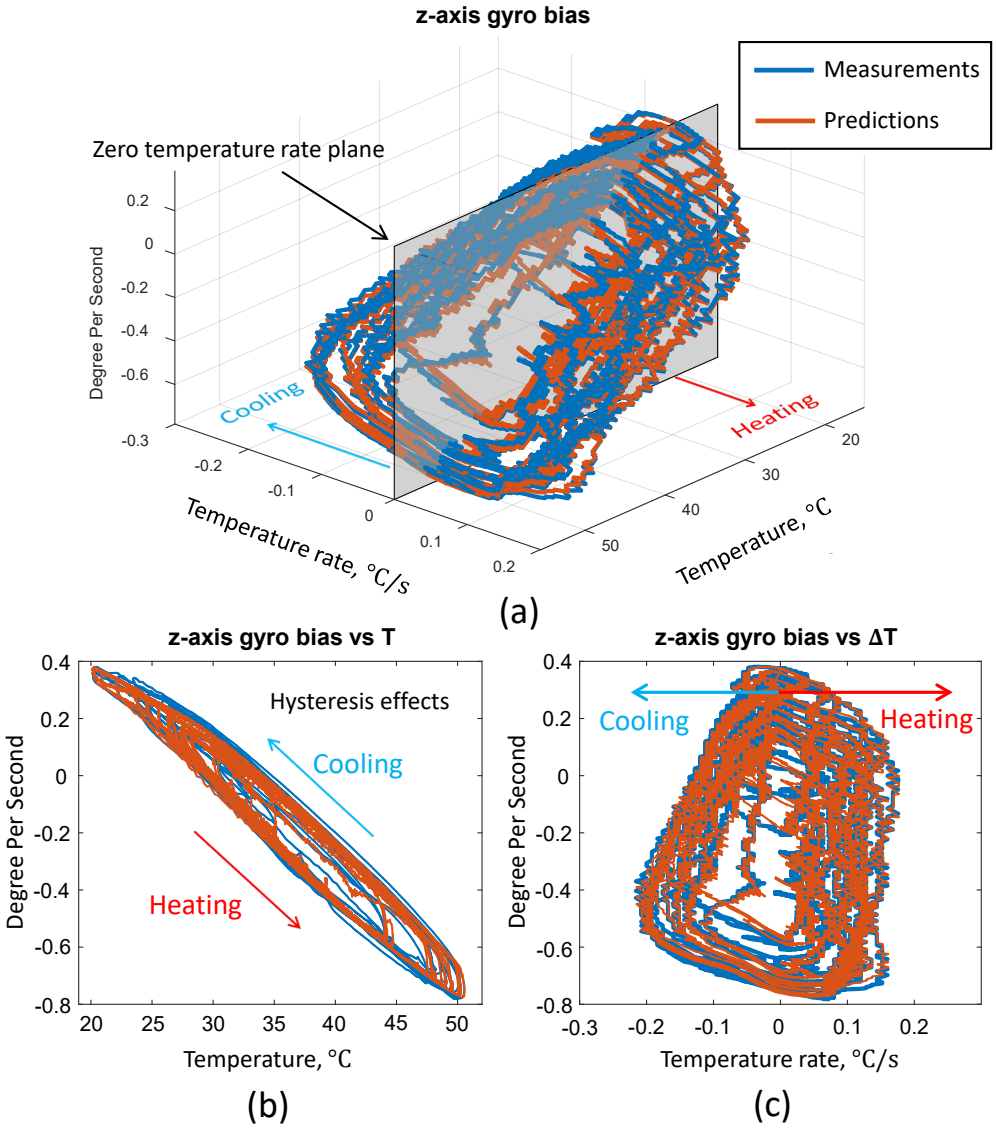


Figure 3.2: (a) Relations of z-axis gyroscope biases measured in the experiment presented subsection 3.2.2 and predicted using the trained BPNN discussed in subsection 3.2.3 versus temperature and temperature rate. The gray transparent plane indicates zero temperature rates and divides the dataset into cooling and heating processes. (b) The relationship between sensor biases and temperature measurements in the dataset shown in (a). Hysteresis effects can be observed. (c) Relations of biases and temperature rates in the dataset shown in (a).

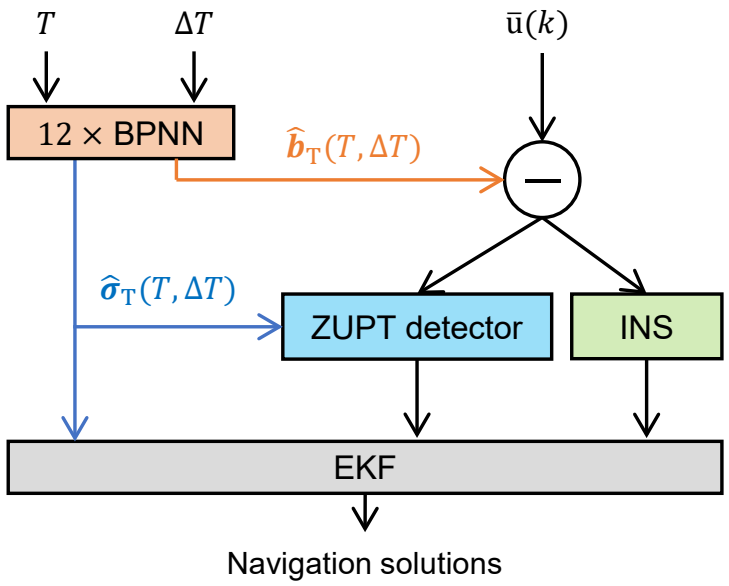


Figure 3.3: Developed temperature-compensated ZUPT-aided INS. The thermal compensation approach uses 12 different BPNNs to separately predict bias drifts and noise standard deviation variations of accelerometers and gyroscopes along the 3 axes. In each BPNN, a 2×1 feature vector including temperature and temperature is used as input.

3.3 Experimental Validation

3.3.1 Experimental Setup

To validate the developed approach in real scenarios, a series of 20 indoor walking experiments was conducted in a temperature-varying environment. The experimental setup is shown in Figure 3.1(a). In the experiment, the same IMU used in the experiment discussed in subsection 3.2.2 was mounted on a pedestrian's right boot, and a TEC was mounted on top of the IMU. Figure 3.1(b) shows an experimental field, where one origin marker and one checkpoint marker, represented respectively by the black square and the green circle in Figure 3.1(b), were placed on the floor. The nominal distance between the two markers was 42.6 [m], measured with an industrial ruler. In each experiment, the pedestrian started by standing still at the origin marker for roughly 20 seconds. This period was used to perform the initial calibration of IMUs. Then, the pedestrian walked at a rate of around 80 steps per minute

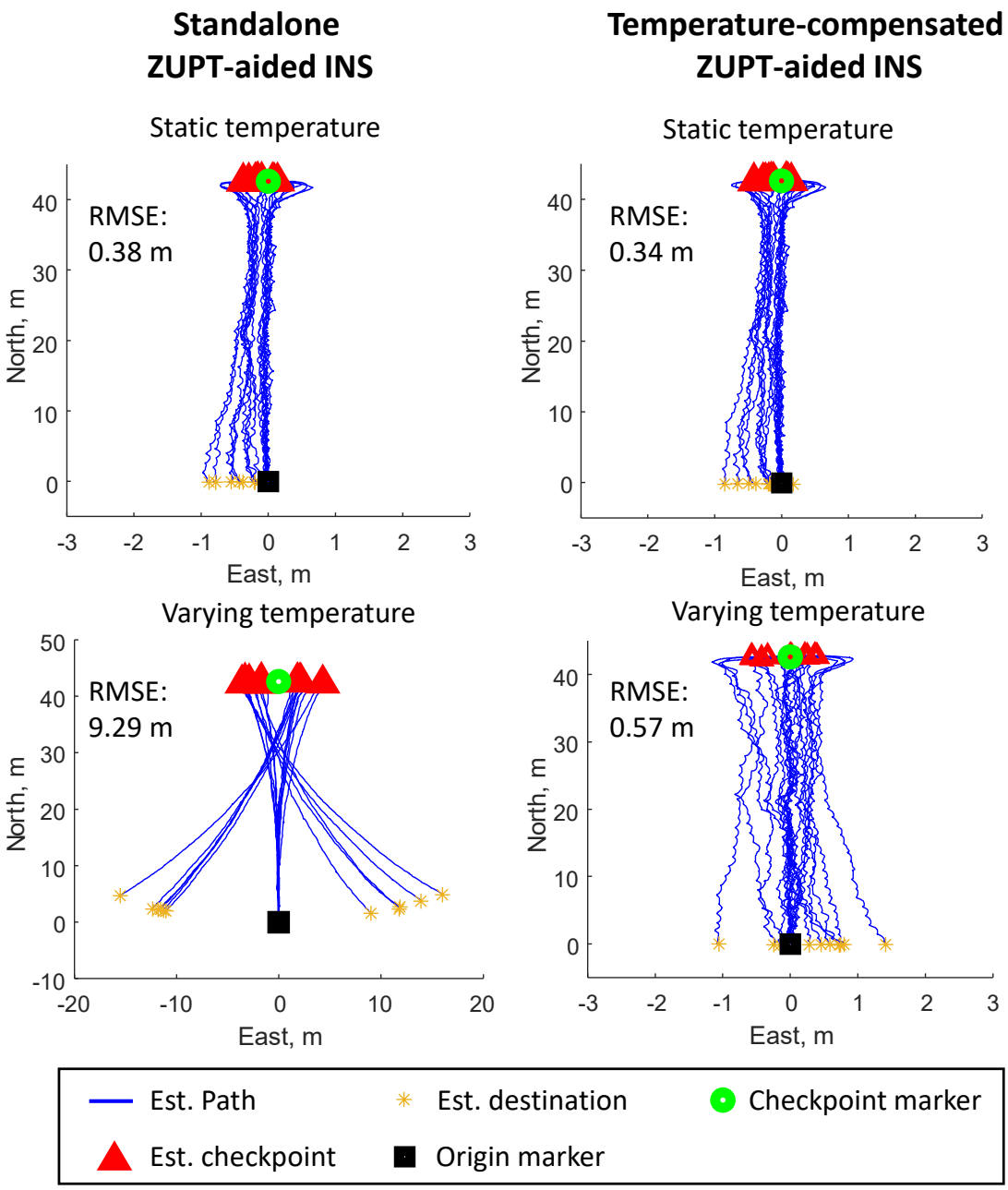


Figure 3.4: Comparison of navigation solutions obtained with a standalone ZUPT-aided INS and the developed temperature-compensated ZUPT-aided INS when operating in environments where temperatures were static or varying. In the static case, the standalone ZUPT-aided INS and the developed approach had similar position RMSE. In the temperature varying cases, our approach outperformed the standalone ZUPT-aided INS.

toward the checkpoint marker and then walked back to the origin marker. The duration of each experiment was around 2 minutes, and the length of the trajectory was approximately 85.2 [m]. In the first ten sets of experiments, the ambient temperature was static at around $25°C \pm 0.05°C$. In the other ten sets of experiments, the ambient temperature measured by the IMU varied and was either increased from 20°C to 50°C or decreased from 50°C to 20°C, five trials were performed for each combination of parameters.

3.3.2 Experimental Results

Figure 3.4 demonstrates the navigation solutions produced by a standalone ZUPT-aided INS and the developed temperature-compensated ZUPT-aided INS in the static temperature and varying temperature environments. The position RMSE was used to evaluate at the origin and checkpoint markers. In this experiment, it could be observed that standalone ZUPT-aided INS had an RMSE of 0.38 [m] while operating at static temperature situations and 9.29 [m] while operating at the temperature-varying situation. The increase in error indicates that the IMU experienced large thermal-induced biases, and the standalone ZUPT-aided INS could not effectively compensate for them. The developed temperature-compensated ZUPT-aided INS demonstrated a similar performance as the standalone ZUPT-aided INS in static temperature situations with an RMSE of 0.34 [m]. However, in temperature-varying situations, the developed approach significantly reduced the RMSE to 0.57 [m]. Based on these experiments, we concluded that enhancing the ZUPT-aided INS with the developed BPNN-based temperature compensation method improved navigation accuracy when operating in the temperature-varying situations, as compared to the standalone ZUPT-aided INS.

3.4 Conclusion

This chapter discussed a BPNN-based temperature compensation method for enhancing ZUPT-aided INS while operating in temperature-varying environments. The temperature compensation method used 12 separately trained feedforward NNs to predict thermal-induced errors, including bias drifts and noise variations of accelerometers and gyroscopes along the three reference axes of IMU. A series of pedestrian indoor walking experiments with a nominal length of 85.2 [m] were conducted to evaluate the developed approach. The experimental results showed that when operating in environments where ambient temperature changed between 20°C and 50°C, a standalone ZUPT-aided INS had a position RMSE of 9.29 [m] while our developed temperature-compensated ZUPT-aided INS achieved a RMSE of 0.57 [m]. We concluded that applying the temperature compensation method to enhance ZUPT-aided INS in the experiments is beneficial. The result presented in this chapter has been published in [90].

Chapter 4

On Algorithm Assumption – Reinforcing Stance Phase Detection

4.1 Introduction

This chapter discusses the development of stance phase detectors used in the ZUPT-aided INS. Two stance phase detectors were developed, and the corresponding navigation performances were experimentally investigated. In this chapter, Section 4.2 identifies a problem in conventional IMU-based stance phase detector, Section 4.3 presents the development of a stance phase detector using foot-mounted IMU and Dynamic Vision Sensor (DVS) with experimental validation, Section 4.4 derives a stance phase detector utilizing a foot-mounted IMU and a downward-facing ultrasonic sensor and evaluates the navigation performance with indoor pedestrian navigation experiments, Section 4.5 concludes this chapter with a discussion of the results.

4.2 False Alarm in Traditional IMU-based Detection

In early developments of the ZUPT algorithm for pedestrian navigation, stance phase detection, or zero velocity detection, was often achieved by comparing a fixed threshold with statistics of likelihood computed from accelerometers' and gyroscopes' measurements [181]. One of the frequently used detector is the SHOE detector. The statistics of the SHOE detector directly relates to stability of the foot, and its detection mechanism is based on an observation that the foot is more stable in the stance phase than in the swing phase. As a result, if the statistics is higher than the defined threshold, the SHOE detector determines the swing phase. Otherwise, the detector indicates the stance phase.

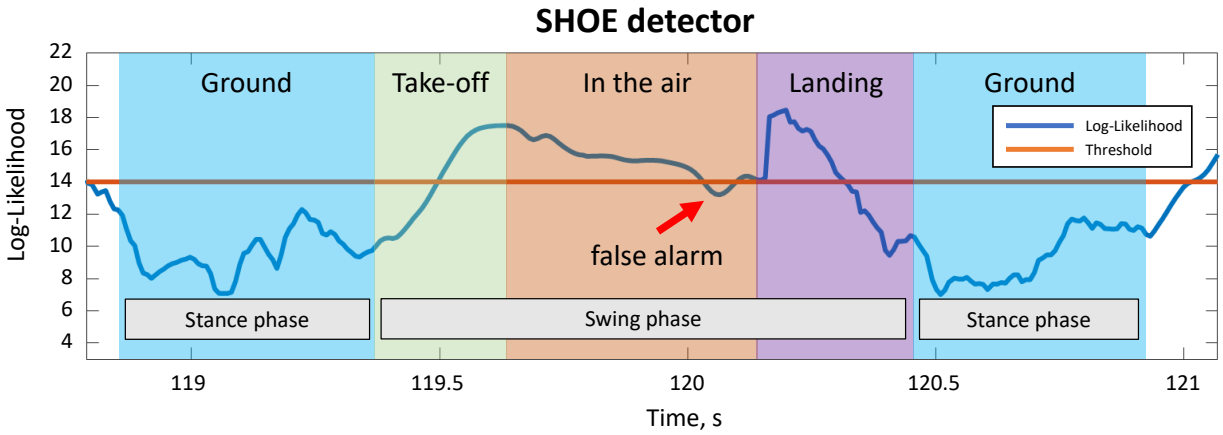


Figure 4.1: An example of the SHOE statistics in one gait cycle. The gait cycle is split into two stance phases and a swing phase. The swing phase can be further divided into three stages. In the first stage, the foot takes off the ground. In the second stage, the foot travels in the air. In the third stage, the foot lands on the ground.

The statistics of the SHOE detector, however, has an undesirable property. Figure 4.1 presents an example of statistics of the SHOE detector of one gait cycle in an indoor walking experiment. In the second stage of the swing phase, the statistics of the SHOE detector decreased when compared to the first and the second stages. The decrease was due to the fact that the foot is more stable when traveling in the air than when taking off or landing. False alarms of zero velocity detection, which is defined as the case that a detector determines

as a stance phase when the foot is in the swing phase, often occur during the period when the foot is in the air. For example, using the threshold indicated by the red line in 4.1 would trigger a false alarm. Notice that although a lower the threshold value can be used to eliminate the false alarm, the lower value does not necessarily lead to an improved overall navigation result.

4.3 Aiding by a Dynamic Vision Sensor

This section presents an event-based camera-assisted zero velocity detector for foot-mounted INS. The developed detector achieves zero-velocity detection by comparing a threshold with its statistics, computed using the firing rate, defined as the total number of events generated during a period, of the event-based camera and the statistics from the SHOE detector. The developed detector is shown to reduce the rate of false alarms of zero velocity detection and to increase the accuracy of pedestrian navigation. This section gives an overview of the DVS, presents hardware design of a foot-mounted INS integrated with a DVS, analyzes the properties of the DVS firing rate during indoor navigation experiments, derives the event-based camera-aided zero velocity detector in a GLRT framework, evaluates the developed detector in terms of detection rate, false alarm rate, and navigation error, and experimentally demonstrates the validity of the developed detector.

4.3.1 Dynamic Vision Sensor Overview

DVS, or Event-based cameras, work differently from a traditional CMOS camera in a way that a DVS asynchronously detects light intensity changes, called events [126]. Some previous work has been conducted to apply the DVS to the field of navigation [229, 230, 137]. The DVS has a high dynamic range, high temporal resolution, low power consumption, and

reduced motion blur [60], making it an intriguing alternative for efficient event detection in foot-mounted INS navigation. Since DVS detects light intensity changes, no event would be generated in an ideal case if no object is moving inside its Field Of View (FOV). This idea can be extended to detect whether a DVS is stationary. Under the assumption that the background inside the FOV is static, a DVS has a property that no event would be produced when DVS is static, and a large number of events would be generated when the DVS is moving. In this section, the developed approach utilizes this property of a DVS to assist the SHOE detector.

4.3.2 Foot-mounted Dynamic Vision Sensor

The Lab-On-Shoe Platform Integrated With a DVS128

To understand the characteristics of measurements obtained from a foot-mounted DVS and investigate its effect when aiding foot-mounted IMU for pedestrian inertial navigation, the Lab-On-Shoe platform was integrated with a mini-eDVS system. The integrated system is shown in Figure 4.2. The Lab-On-Shoe was previously developed by UCI MicroSystems Lab for evaluating navigation performance of foot-mounted IMUs when aided by other non-inertial sensors, such as barometers, ultrasonic sensors, and cameras [18]. The mini-eDVS system contains an event-based camera DVS128 and a micro-controller [229]. The detail description of the event-based camera DVS128 can be found in [126]. In this section, the IMU sampling rate was set to 120 Hz. The DVS firing rate was obtained by calculating the number of events generated in a period of 0.0083 s.

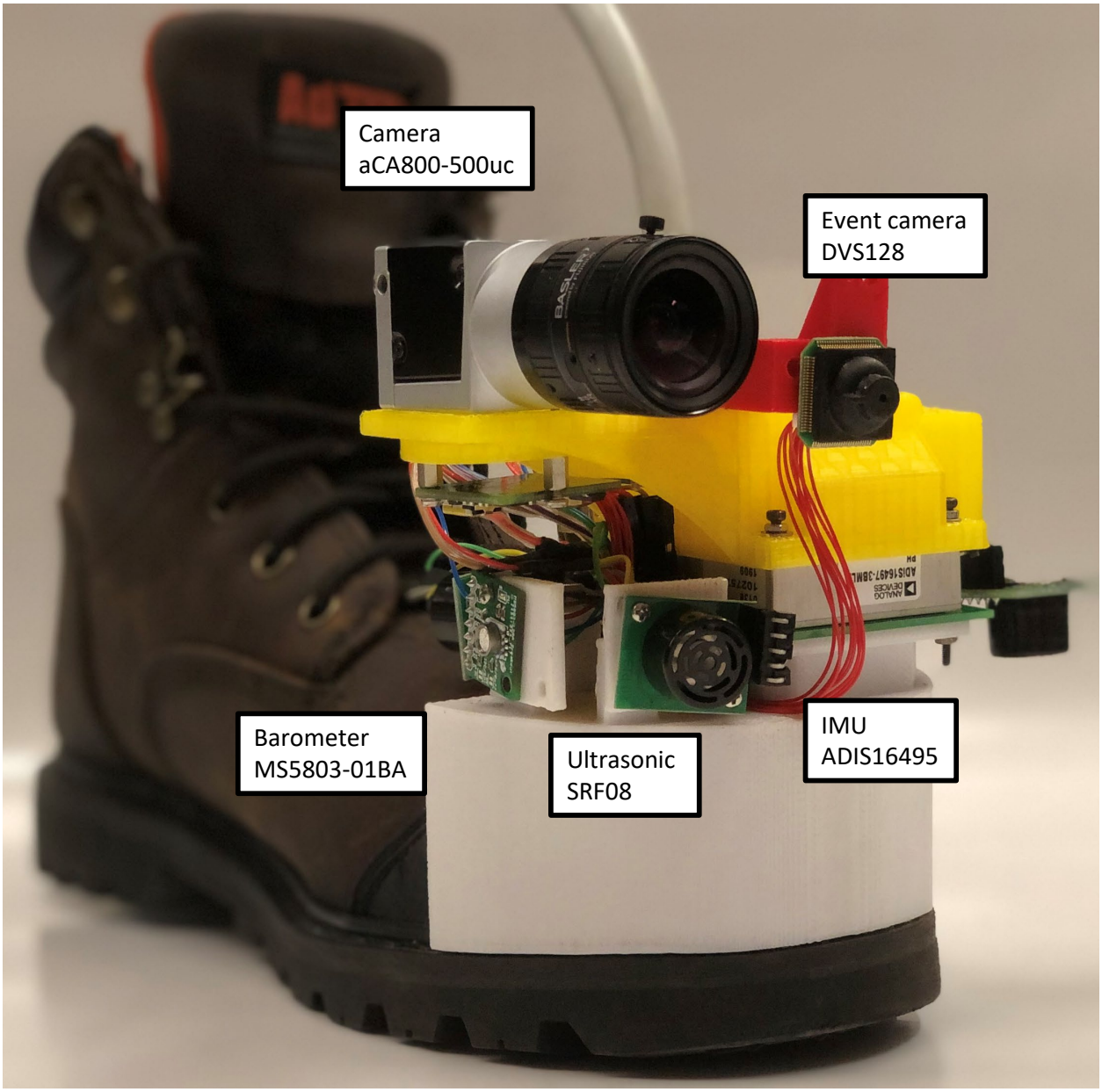


Figure 4.2: The Lab-On-Shoe platform integrated with DVS128. The Lab-On-Shoe platform is equipped with an IMU, three ultrasonic sensors, a barometer, a CMOS camera, and a DVS. The developed DVS-aided SHOE detector discussed in this section only uses the DVS and the IMU.

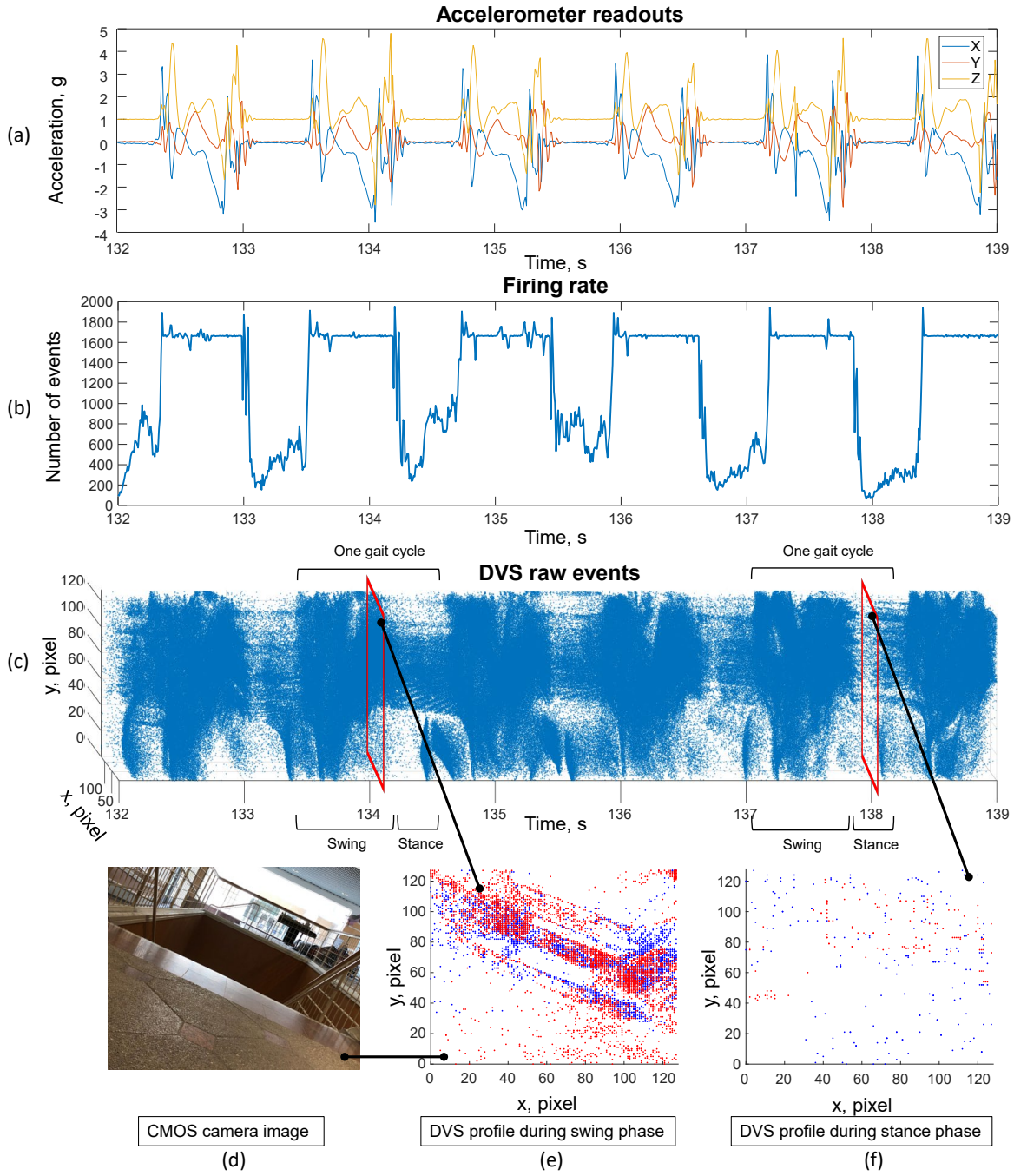


Figure 4.3: (a) The accelerometer readouts in an indoor walking experiment. (b) The corresponding DVS firing rate in the same experiment. (c) An example of DVS events collected in the same experiment. (d) A group of events collected during a swing phase in the experiment. (e) A CMOS image of the scene generating DVS events during the swing phase. (f) A group of events collected during a stance phase in the experiment.

Measurements of Foot-mounted DVS

A DVS asynchronously detects light intensity changes, called events. A positive light intensity change is represented as a positive event, and a negative change is a negative event. In this section, the two types of events are not distinguished as they are both generated by light intensity changes. One scenario that events are produced is when a scene observed is moving relatively to the DVS. In the case of a shoe-mounted DVS in a walking experiment, a large number of events are produced when the shoe is moving, and fewer events present when the shoe is stationary. An example of DVS events generated in an indoor walking experiment is shown in Figure 4.3(c). Each of the blue particles in Figure 4.3(c) is an event. For visualization, we collected the events generated in a 0.0083 s window and displayed the events as an DVS image. Figure 4.3(e) and (f) illustrate DVS images taken during a swing phase and a stance phase, respectively. The blue and the red particles in Figure 4.3(e) and (f) represent positive and negative events. We can observe that the number of events generated during the stance phase is distinctively less than the swing phase. Note that even though the shoe-mounted DVS had minimum movement during the stance phase, events exist in Figure 4.3(f). Those events are considered as noise events.

The firing rate of a DVS is defined as the total number of events triggered within a period. The DVS firing rate of the walking experiment is presented in Figure 4.3(b). Accelerometer readouts collected in the same walking experiment, shown in Figure 4.3(a), are used as references to the stance phases and the swing phases. Figure 4.3(b) illustrates that the firing rate increases during the swing phases and decreases during the stance phases. Thus, the firing rate of a shoe-mounted DVS can be used to assist the SHOE detector, which only relies on IMU measurements.

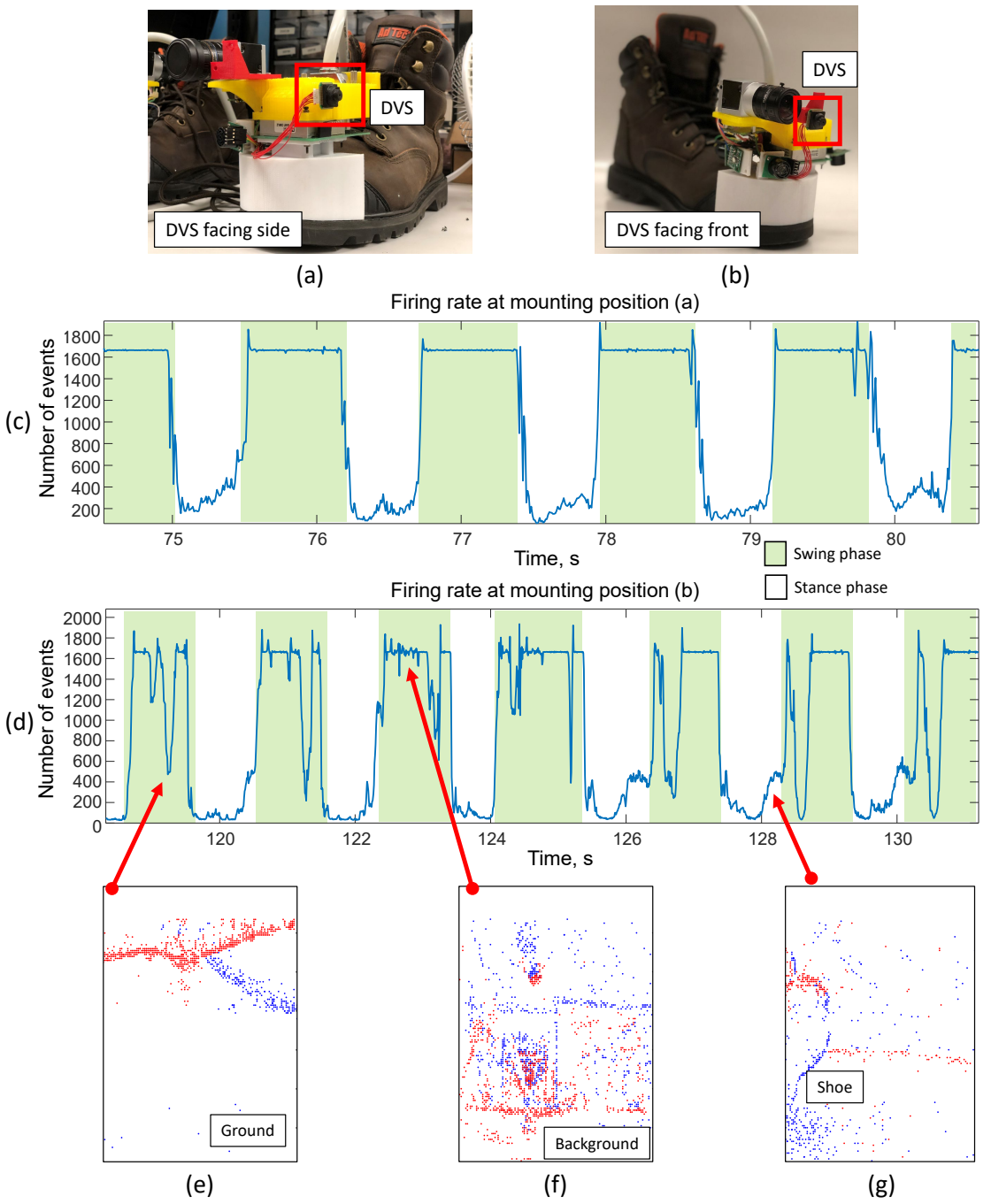


Figure 4.4: (a) DVS is mounted next to the IMU and faces outward. (b) DVS is mounted next to the IMU and faces outward. (c) Firing rate in an indoor walking experiment with DVS mounting configuration shown in (a). (d) Firing rate in an indoor walking experiment with DVS mounting configuration shown in (b). (e) An example of a DVS image taken when the DVS is facing the ground. (f) An example of a DVS image taken when the DVS is moving and facing forward. (g) An example of a DVS image taken during the stance phase, capturing events generated by the other shoe.

Mounting Position of DVS

To maximize the assistance that a DVS can provide for zero-velocity detection, two indoor walking experiments were conducted with the Lab-On-Shoe platform integrated with the DVS128 to determine a mounting configuration for the DVS. The two experiments were conducted with the same path and walking speed and different mounting configurations for the DVS. The mounting configuration of the DVS in the first experiment is illustrated in Figure 4.4(a), where the DVS was mounted next to the IMU and faced outward to surrounding walls. In the second experiment, the DVS was mounted on top of the IMU and faced toward the front. The configuration of the second experiment is shown in Figure 4.4(b).

Figures 4(c) and (d) illustrate examples of the firing rates collected during the first and the second experiments, respectively. Comparing the firing rates in the swing phase, we can see that the configuration demonstrated in Figure 4.4(a) led to a consistent DVS firing rate. In contrast, the firing rate collected with the configuration shown in Figure 4.4(b) fluctuated dramatically. The fluctuation was contributed by the fact that with the configuration in Figure 4.4(b), there is a period during the swing phase that the FOV of the DVS was facing the ground. The ground in the experiments does not have many visual features. Thus, the number of events that were generated was less when the DVS is facing the ground. An example of events generated when the DVS was facing the ground is shown in Figure 4.4(e). In this series of experiments, we also found that using the configuration in Figure 4.4(b) has another drawback. During the stance phase, the other shoe can come inside the FOV of the DVS, generating events that are not beneficial for zero-velocity detection. Figure 4.4(g) shows an example of the events caused by the other shoe. Since zero-velocity detection is based on comparing a statistics of likelihood with a threshold, one of the desired properties is that the statistics during the stance phase is distinct from that during the swing phase. Thus, to achieve the best performance for zero velocity detection, we conclude that the DVS should be mounted next to the IMU, facing outward.

4.3.3 DVS-aided Zero Velocity Detection (DVS-SHOE)

A zero velocity detector that utilizes statistics of the SHOE detector and firing rate of a DVS is developed. The developed detector will be referred to as the DVS-aided SHOE (DVS-SHOE) detector in the following discussion. The derivation of the DVS-SHOE is similar to the derivation of the SHOE detector presented in [181], which uses the GLRT framework. In the derivation of the DVS-SHOE detector, the inertial sensors measurements, y_k^α and y_k^ω, was augmented with the DVS firing rate, y_k^λ, and the measurement vector y_k can be expressed in the following form:

$$y_k = \begin{bmatrix} y_k^\alpha \\ y_k^\omega \\ y_k^\lambda \end{bmatrix} = \begin{bmatrix} s_k^\alpha \\ s_k^\omega \\ s_k^h \end{bmatrix} + \begin{bmatrix} v_k^\alpha \\ v_k^\omega \\ v_k^\lambda \end{bmatrix} = s_k + v_k.$$

Here, $s_k^\alpha \in \mathbb{R}^3$ and $s_k^\omega \in \mathbb{R}^3$ denote the IMU-experienced acceleration and angular rate, respectively. $s_k^\lambda \in \mathbb{R}$ denotes denotes the firing rate of the DVS. $v_k^\alpha \in \mathbb{R}^3, v_k^\omega \in \mathbb{R}^3$ and $v_k^h \in \mathbb{R}$ represent the measurement noises of the accelerometer, gyroscope, and ultrasonic sensor, respectively. The derivation used an assumption that the measurement noises of accelerometers, gyroscopes, and DVS firing rate are independent and identically-distributed white Gaussian noises with respective variances $\sigma_\alpha^2, \sigma_\omega^2$, and σ_λ^2.

As discussed in [181], the two hypotheses, H_0 and H_1, in zero-velocity detection correspond to the cases of the swing phase and the stance phase, respectively. Under the hypothesis H_1, the accelerometer only experiences the gravitational acceleration; the angular rate experienced by the gyroscope is zero; the DVS firing rate is a constant value λ. For an ideal noise-free DVS, λ is zero. In a practical situation, noise events can be generated even when no object is moving in a DVS field of view, and therefore, λ is non-zero. In the implementation, λ=30 was experimentally determined. Under the hypothesis H_0, forces from the foot lead to complicated foot motion, so that the accelerometer experiences acceleration exceeding

gravity, the gyroscope readouts fluctuate, and the DVS detects events generated by the foot motion. For the two hypotheses, the sensor measurements were assumed to satisfy the following conditions:

$$H_0 : \exists k \in \Omega_n, s_k^{\alpha} \neq g u_n, s_k^{\omega} \neq 0_{3\times 1}, s_k^{\lambda} \neq \lambda,$$
$$H_1 : \forall k \in \Omega_n, s_k^{\alpha} = g u_n, s_k^{\omega} = 0_{3\times 1}, s_k^{\lambda} = \lambda,$$

where u_n is a 3×1 unit vector, g is the gravitational constant, and $\Omega_n = \{l \in \mathbb{N}, n \leq l < N-1\}$ is a collection of the sensor measurement indexes at time n with a window of length N.

Following the derivation described in [181], the developed DVS-SHOE detector chooses H_1 if

$$T_{\lambda}(z_n) = \frac{1}{N} \sum_{k \in \Omega_n} \left(\frac{1}{\sigma_{\alpha}^2} \parallel y_k^{\alpha} - g \frac{\bar{y}_k^{\alpha}}{\parallel \bar{y}_k^{\alpha} \parallel} \parallel^2 + \frac{1}{\sigma_{\omega}^2} \parallel y_k^{\omega} \parallel^2 + \frac{1}{\sigma_{\lambda}^2} \parallel y_k^{\lambda} - \lambda \parallel^2 \right) < \gamma, \tag{4.1}$$

where $z_n = \{y_k\}_{k=n}^{k=N-1}$ and γ are user-defined thresholds.

4.3.4 Experimental Results

To validate the DVS-SHOE detector, a series of indoor close-loop walking experiments was performed with the Lab-On-Shoe platform integrated with the DVS128. The series of experiments included ten nominally identical trials. In each trial, a subject walked over 150 steps for a duration of 160 m in about 120 s. The trajectory included a flat surface, a ramp, and stairs. A reference trajectory generated by ZUPT-aided INS with the DVS-SHOE detector is shown in Figure 4.6(a).We used two methods to evaluate the DVS-SHOE detector. The first method was to study the detection performance in terms of the detection rate and the false alarm rate. The second method was to investigate the navigation results of ZUPT-aided

INS with different detectors. Detailed implementation of the ZUPT-aided INS is presented in [216].

Detector Performance

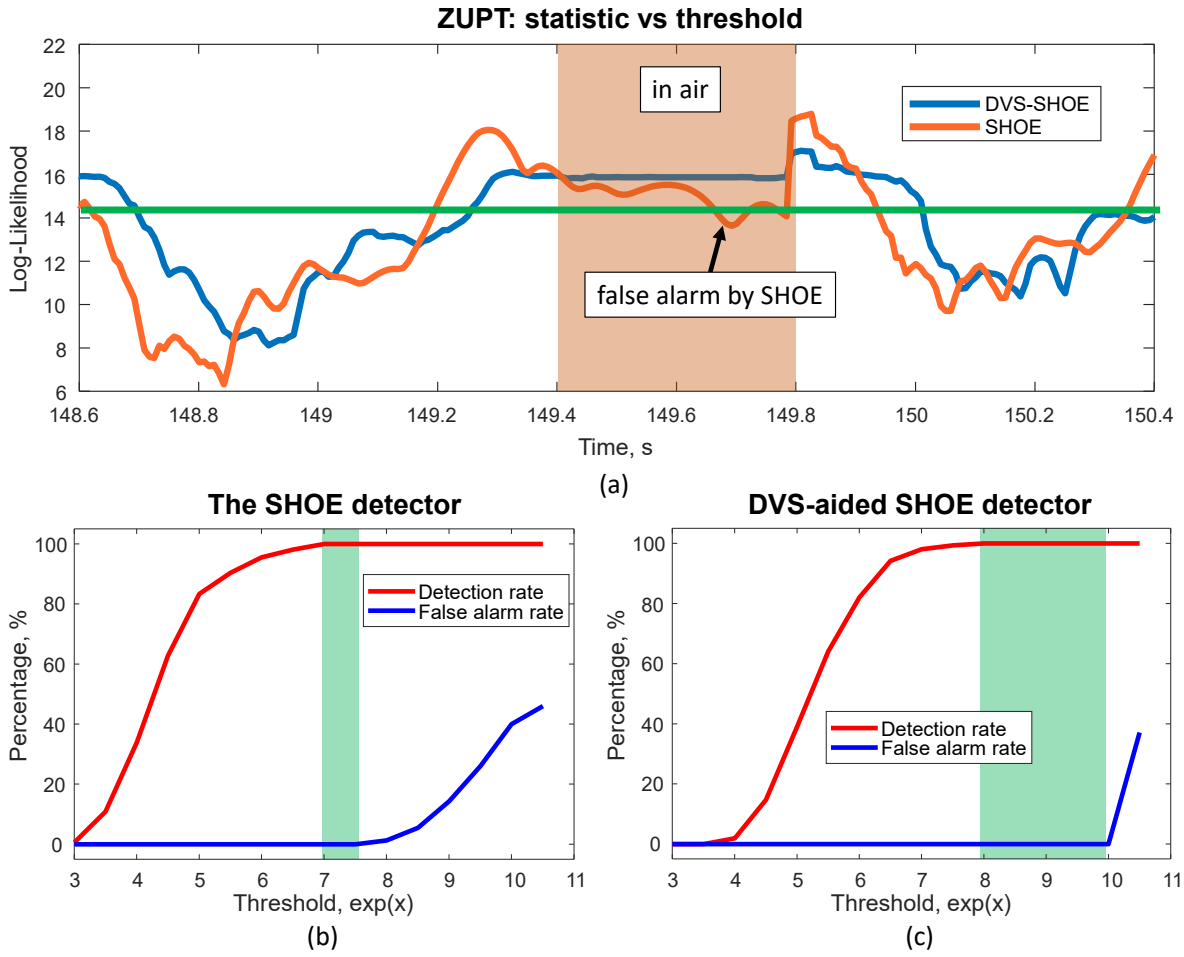


Figure 4.5: (a) shows an example of SHOE and DVS-SHOE statistics for one gait cycle in the indoor walking experiment. The orange area indicates the stage of shoe traveling in the air during the swing phase. (b) presents the detection performance of the SHOE detector in the indoor walking experiments. The green area indicates the range of thresholds that achieves a near 0% false alarm and 100% detection rate. (c) demonstrates the detection performance of the DVS-SHOE detector in the indoor walking experiments. The green area in (c) is larger than the one shown in (b). The larger green area implies that the DVS-SHOE detector is more robust than the SHOE detector.

The detection performance of the DVS-SHOE detector is first compared with that of the SHOE detector. False alarm rate and detection rate were used as metrics for evaluation.

In zero-velocity detection, a false alarm is generated when the shoe is moving, but a stance phase detector indicates a stance phase. A mis-detection is produced when the shoe is on the ground, but the detector determines a swing phase. Figure 4.5(a) illustrates that using the threshold indicated by the green line in Figure 4.5(a) for the SHOE detector would lead to a false alarm in the walking experiment. The false alarm was eliminated when the statistics of the DVS-SHOE was used. The elimination was due to the fact that during the period when the shoe is traveling in the air, the DVS firing remains consistently on a high level. Thus, the statistics of the DVS-SHOE was kept high during the orange area in Figure 4.5(a), leading to a reduction of the false alarm.

Figure 4.5(b) and (c) demonstrate the detection rates of the SHOE detector and the DVS-SHOE detector when different values of thresholds were used. We can observe that the first false alarm for the DVS-SHOE detector happened at a much larger threshold value than in the case of the SHOE detector. The green areas in both Figure 4.5(b) and (c) indicate that when a threshold value is chosen from this range, the detector achieves a near 0% false alarm rate and a 100% detection rate. In this series of experiments, the green area in Figure 4.5(e) is larger than Figure 4.5(d), implying that the DVS-SHOE detector can handle a larger range of thresholds.

Navigation Performance

To evaluate the DVS-SHOE detector in a more realistic case, the navigation accuracy of ZUPT-based INS when using the SHOE detector is compared with case using the DVS-SHOE detector. The navigation accuracy is evaluated with the CEP. The estimated final destinations of the ten sets of experiments and the CEPs are presented in Figure 4.6(b) and (c). The experimental results showed that the CEP is reduced by around 25%, from 1.2 m to 0.9 m, when the DVS-SHOE detector is applied. The improvement is a direct result of the fact that the DVS-SHOE detector has a better zero-velocity detection performance than

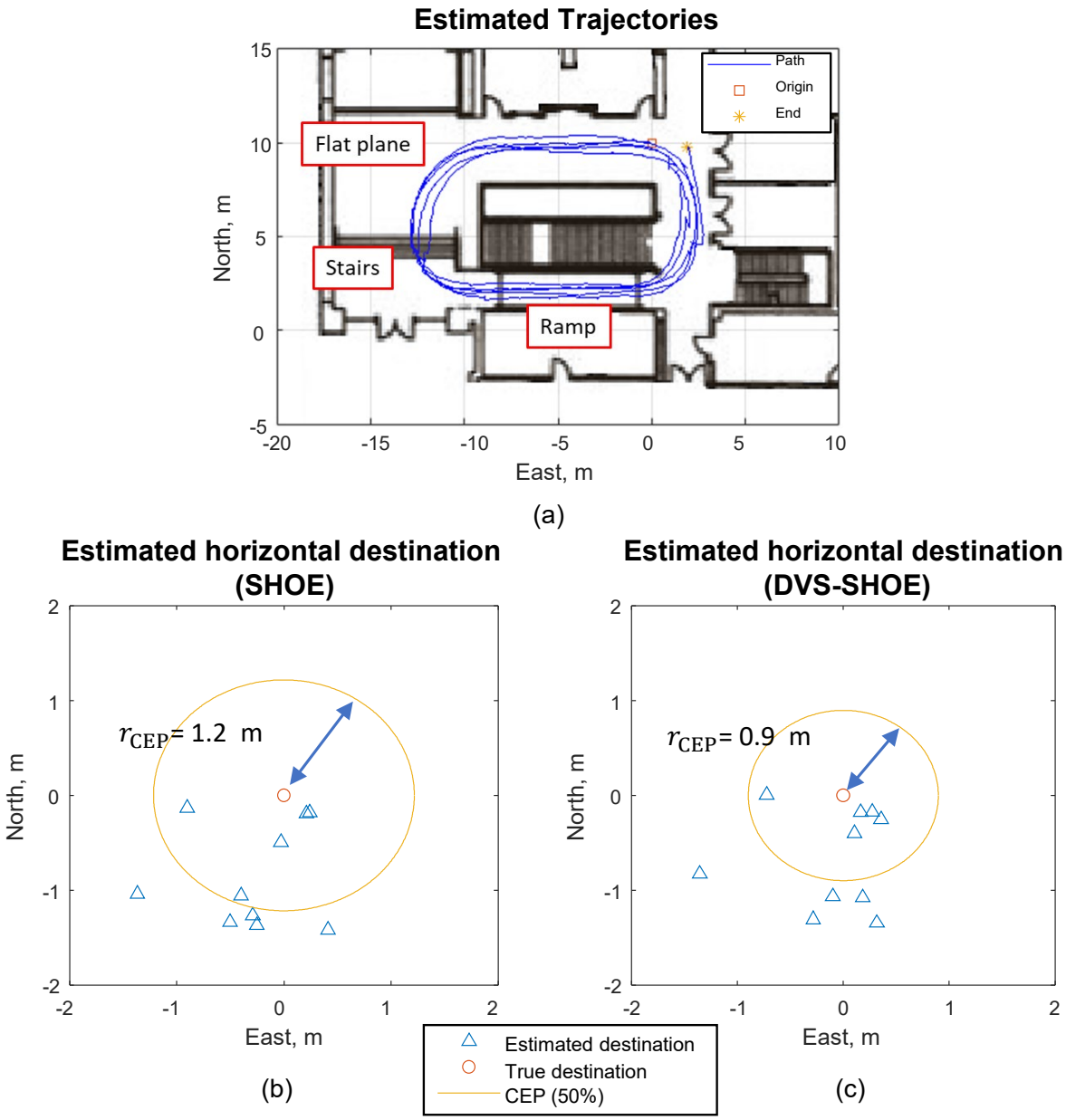


Figure 4.6: (a) shows an example of SHOE and DVS-SHOE statistics for one gait cycle in the indoor walking experiment. The orange area indicates the stage of shoe traveling in the air during the swing phase. (b) presents the detection performance of the SHOE detector in the indoor walking experiments. The green area indicates the range of thresholds that achieves a near 0% false alarm and 100% detection rate. (c) demonstrates the detection performance of the DVS-SHOE detector in the indoor walking experiments. The green area in (c) is larger than the one shown in (b). The larger green area implies that the DVS-SHOE detector is more robust than the SHOE detector.

the SHOE detector.

The DVS-SHOE detector has two constraints. First, the detector used an assumption that the scene observed by the DVS is static to use the DVS firing rate for zero-velocity detection. Nevertheless, this is not always the case. If the DVS observes moving objects during the stance phase, the statistics of the DVS-SHOE will increase, leading to mis-detections. Second, if the scene inside the field of view does not have many visual features, for example, a white wall, then the statistics of the DVS-SHOE will decrease, leading to false alarms.

This section presented a novel zero velocity detector, the DVS-SHOE detector, for ZUPT-aided INS augmented by a foot-mounted event-based camera DVS128. The firing rate of the foot-mounted DVS was demonstrated to increased during the swing phase and decreased during the stance phase in a walking experiment. The DVS mounting configuration, which is to mount the DVS next to an IMU and face the sensor outward for optimal performance of zero-velocity detection, was experimentally determined. Two methods were used to evaluate the developed DVS-SHOE detector. First, we compared the detection performances of the SHOE detector and the DVS-SHOE detector in terms of false alarm rate and detection rate. The experimental results showed that the DVS-SHOE detector achieved a lower false alarm rate than the SHOE detector. Second, we compared the navigation performance of the ZUPT-aided INS using the SHOE detector and the DVS detector. The experimental results showed that the CEP of the case using DVS-SHOE is reduced by around 25%, from 1.2 m to 0.9 m, as compared to the case of the SHOE detector. The results presented in this section was published in [88].

4.4 Aiding by Downward-facing Range Sensor

This section presents the development of a zero velocity detector, referred to as the Ultrasound-Aided SHOE (UA-SHOE) detector, for foot-mounted INS assisted by a downward-facing range sensor. The UA-SHOE detector is based on statistics combining the SHOE detector, which relies only on IMU measurements, with an additional information about the height of the foot relative to the ground. In the rest of this section, for clarity, the detectors presented in [181] and [255] will be referred to as the SHOE detector and the Ultrasonic-only Stance Phase Detection (USPD) detector, respectively. In the implementation of the UA-SHOE detector, the height information is acquired with the downward-facing range sensor, which is an ultrasonic sensor mounted at the foot's arch. A combination of the SHOE detector and the ultrasonic measurements is shown in this section to result in an improved navigation performance.

This section derives the statistics of the detector in a General Likelihood Ratio Test (GLRT) framework, verifies the developed UA-SHOE detector with experiments conducted by traveling at different speeds, and compares the performance of the developed UA-SHOE detector with a commonly used SHOE detector and the USPD detector in terms of the navigation errors of the ZUPT-aided INS.

4.4.1 Detector Derivation With General Likelihood Ratio Test

The developed UA-SHOE detector can be derived similarly to the GLRT framework presented in [181]. The inertial sensors measurements $\mathbf{y}_k$ are augmented by the height of the

shoe relative to the ground, and can be presented in the following form:

$$\mathbf{y}_k = \begin{bmatrix} \mathbf{y}_k^{\alpha} \\ \mathbf{y}_k^{\omega} \\ \mathbf{y}_k^{h} \end{bmatrix} = \begin{bmatrix} \mathbf{s}_k^{\alpha} \\ \mathbf{s}_k^{\omega} \\ \mathbf{s}_k^{h} \end{bmatrix} + \begin{bmatrix} \mathbf{v}_k^{\alpha} \\ \mathbf{v}_k^{\omega} \\ \mathbf{v}_k^{h} \end{bmatrix} = \mathbf{s}_k + \mathbf{v}_k.$$

Here, $\mathbf{s}_k^{\alpha} \in \mathbb{R}^3$ and $\mathbf{s}_k^{\omega} \in \mathbb{R}^3$ denote the IMU-experienced acceleration and angular rate, respectively. $\mathbf{s}_k^{h} \in \mathbb{R}$ denotes the height of the shoe relative to the ground. $\mathbf{v}_k^{\alpha} \in \mathbb{R}^3, \mathbf{v}_k^{\omega} \in \mathbb{R}^3$ and $\mathbf{v}_k^{h} \in \mathbb{R}$ represent the measurement noises of the accelerometer, gyroscope, and ultrasonic sensor, respectively. In the derivation, the measurement noises of accelerometers, gyroscopes, and range sensors were assumed to be independent and identically-distributed white Gaussian noises with respective variances $\sigma_{\alpha}^2, \sigma_{\omega}^2$, and σ_h^2.

Recall that zero velocity detection is a binary hypothesis test problem, and the two hypotheses, H_0 and H_1, are defined as the cases of the swing phase and the stance phase, respectively. Under the hypothesis H_0, the sensor measurements are not consistent, so no simplifying assumption is made. Under the hypothesis H_1, the accelerometer only experiences the gravitational acceleration, and the angular rate experienced by the gyroscope is zero. Moreover, under H_1, the shoe height is a constant value h. In other words, for the two hypotheses, we assume the sensor measurements should satisfy the following conditions:

$$H_0 : \exists k \in \Omega_n, \mathbf{s}_k^{\alpha} \neq g\mathbf{u}_n, \mathbf{s}_k^{\omega} \neq \mathbf{0}_{3\times 1}, \mathbf{s}_k^{h} \neq h,$$
$$H_1 : \forall k \in \Omega_n, \mathbf{s}_k^{\alpha} = g\mathbf{u}_n, \mathbf{s}_k^{\omega} = \mathbf{0}_{3\times 1}, \mathbf{s}_k^{h} = h,$$

where $\mathbf{u}_n$ is a 3×1 unit vector, g is the gravitational constant, and $\Omega_n = \{l \in \mathbb{N}, n \leq l < N-1\}$ is a collection of the sensor measurement indexes at time n with a window of length N.

Following the derivation described in [181], the developed UA-SHOE detector chooses H_1 if

$$T_h(\mathbf{z}_n) = \frac{1}{N}\sum_{k\in\Omega_n}(\frac{1}{\sigma_\alpha^2} \parallel y_k^\alpha - g\frac{\bar{y}_k^\alpha}{\parallel \bar{y}_k^\alpha \parallel} \parallel^2 + \frac{1}{\sigma_\omega^2} \parallel y_k^\omega \parallel^2 + \frac{1}{\sigma_h^2} \parallel y_k^h - h \parallel^2) < \gamma, \tag{4.2}$$

where $\mathbf{z}_n = \{\mathbf{y}_k\}_{k=n}^{k=N-1}$ and γ are user-defined thresholds.

4.4.2 Performance Evaluation

Figure 4.7: The Lab-On-Shoe platform integrated with a downward-facing ultrasonic sensor. The camera in the picture was not used in this work.

This subsection compares the performance of the developed UA-SHOE detector with a commonly used SHOE detector and USPD detector. The USPD detector uses the range sensor measurements smoothed by the 2nd order Butterworth low-pass filter. The normalized cut-off frequency of the filter was set to 0.1 [Hz]. Two series of experiments were conducted with the Lab-On-Shoe platform integrated with a downward-facing range sensor. The Lab-On-Shoe system is documented in [18]. The downward-facing range sensor was mounted at the middle part of the shoe, shown in Figure 4.7. The range sensor used in this section is an ultrasonic sensor SRF08, but it can be replaced by other range sensors, such as LiDAR. The

sampling rate of the ultrasonic sensor was 25 [Hz], while the IMU sampling rate was 120 [Hz]. Each series contained 10 sets of identical experiments. In the first series of experiments, the subject walked straight towards the North on a flat surface for 42.67 [m] at a speed of around 1 [m/s] (80 [steps/min]) in 45 [s]. In the second series, the subject repeated the same experiments but running at a speed of around 2 [m/s] (160 [steps/min]) in 23 [s].

Statistics Comparison

Figure 4.8(a), (b), and (c) show examples of the statistics of the developed UA-SHOE detector, the commonly used SHOE detector, and USPD detector. The red line in each plot is the threshold used to compare the statistics of the detectors. Two trends shall be noted in these plots. First, the smoothed ultrasonic sensor measured, at least once during the stance phases, the minimum shoe height selected to be $4cm$ in these experiments. Second, as compared to the original SHOE detector, the statistics of the developed UA-SHOE detector is higher during swing phases, implying that the rate of false alarms was reduced. This is because the UA-SHOE detector, described by (4.2), has an additional term $\frac{1}{\sigma_h^2} \parallel y_k^h - h \parallel^2$. In our view, the major advantage of adding this term is that its value directly tests if the shoe is on the ground, while the other two terms in (4.2) relate to stability of the shoe and are not necessarily a reliable indicator whether the shoe is on the ground. Since the ultrasonic readouts are equal to the minimum shoe height, h, when the shoe contacts the ground, the noise variance σ_h^2 can be set to a minimal value. In this section, $\sigma_h^2 = 10^{-38}$.

Figure 4.8(d) shows the detection results in a gait cycle of the walking experiment using the three detectors with the thresholds indicated by the red line in Figure 4.8(a), (b), and (c). The accelerometer measurements are superimposed in the plot as a reference for the swing and stance phases. Three lessons can be learned from results shown in Figure 4.8. First, during the gait cycle, all three detectors identified correctly the stance phase at least once. Second, during $33.15s$ and $33.25s$, a stance phase period was detected by the USPD detector,

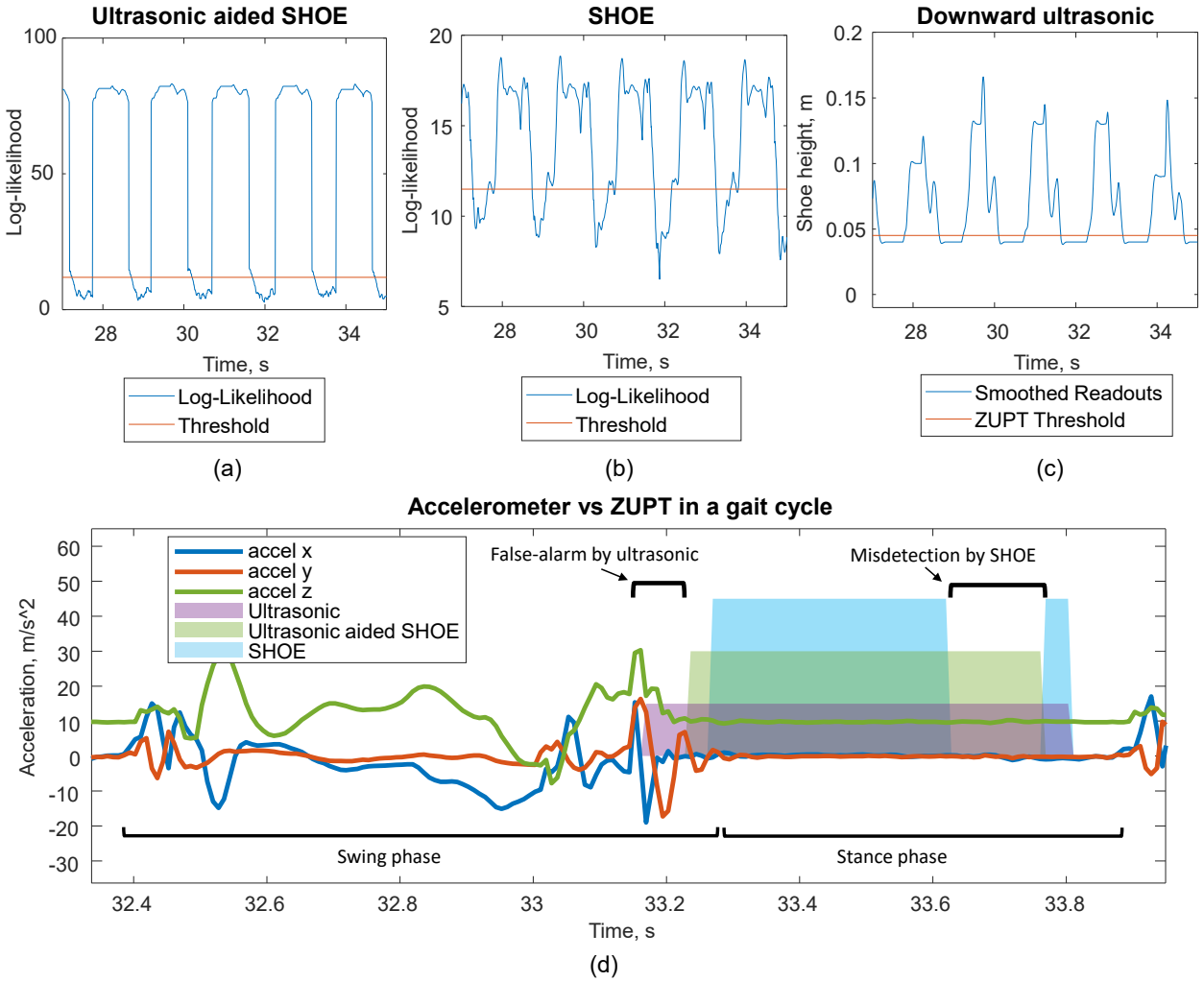


Figure 4.8: (a), (b), and (c) are examples of the statistics of the developed UA-SHOE detector, the SHOE statistics, and the smoothed ultrasonic sensor readouts in a walking experiment. (d) shows the stance phase detected by the three detectors in a gait cycle. In this gait cycle, the USPD detector had a false alarm, the SHOE detector had a mis-detection, and the UA-SHOE detector had the best detection performance.

but the accelerometer measurements indicated that IMU was experiencing an acceleration generated by not only the gravity but also by other inertial forces. We considered this period as a false alarm of the detector. The false alarm was possibly generated because of two reasons: 1) when the arch of the foot touched the ground, the IMU mounted at the toe side was still experiencing a short period of vibration, and 2) the lower sampling rate of the ultrasonic sensor resulted in a delay between its measurements and the IMU measurements. Third, during 33.6 [s] and 33.8 [s], the SHOE detector had a mis-detection. Although this mis-detection can be eliminated by using a higher threshold, the higher threshold may not necessarily result in a better navigation accuracy in this experiment. In our experiment, the UA-SHOE detector had the best zero-velocity detection performance.

Navigation Error Comparison

Figure 4.9 presents the navigation results of two series of experiments utilizing ZUPT-aided INS with three detectors, the UA-SHOE detector, the SHOE detector, and the USPD detector. The thresholds used for walking and running experiments were the same, and were set to e^{12}, $e^{11.5}$, and 0.045 for the three detectors, respectively. The initial heading of each experiment was adjusted so that the first $10m$ of all the trajectories were aligned. The navigation errors are expressed in terms of CEP, whose radius by definition contains at least 50% of the destinations, and RMSE.

Figure 4.9 illustrates that the RMSEs of the UA-SHOE detector are marginally better than the USPD detector by 27.5% ($24cm$) for the walking case and 11.3% ($25cm$) for the running case. These improvements are mainly because the UA-SHOE detector reduces false alarms introduced by the USPD detector. One example of the USPD false alarm reduced by the UA-SHOE detector is illustrated in Figure 4.8(d). The RMSEs of the UA-SHOE detector and the SHOE detector in the walking experiments had a comparable performance, with a discrepancy of less than 7% ($5cm$). In the running experiments, the UA-SHOE detector

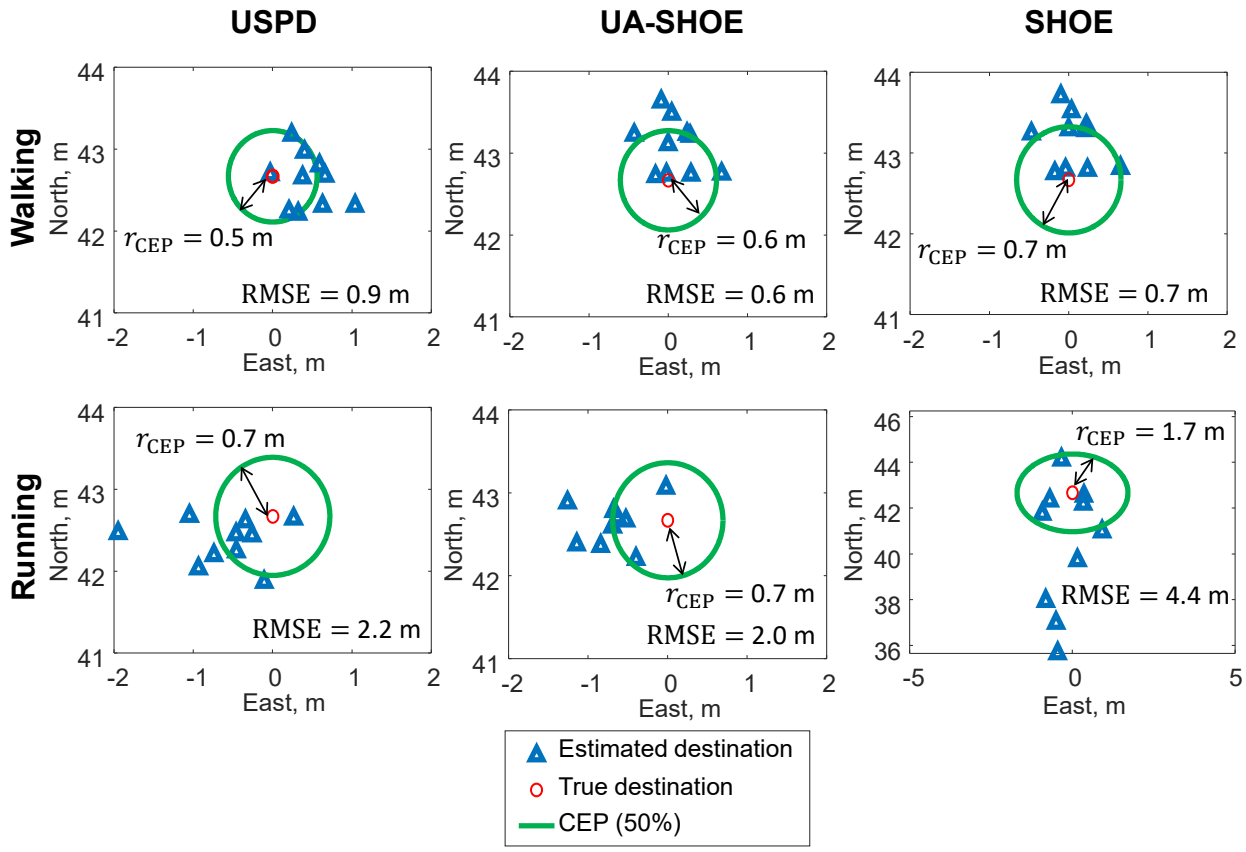


Figure 4.9: Comparison of navigation results estimated by ZUPT-aided INS using three different detectors of zero-velocity events.

outperformed the original SHOE detector by more than 50%.

This section presents a new zero velocity detector, the UA-SHOE detector, for foot-mounted INS assisted by a downward-facing range sensor. The experimental results of walking and running cases showed that, compared to the USPD detector, the RMSEs of the UA-SHOE detector are improved by 27.5% and 11.3%, respectively. The RMSEs of the UA-SHOE detector and the original SHOE detector in the walking experiments had a comparable performance, with a discrepancy of less than 7%. In the running experiments, the UA-SHOE detector outperformed the original SHOE detector by more than 50%. The result of this section was published in [93].

4.5 Conclusion

This chapter presents two stance phase detectors for the ZUPT-aided INS, the UA-SHOE detector and the DVS-SHOE detector, that fuse IMU measurements with additional non-inertial sensing modalities of a downward-facing ultrasonic sensor and a DVS, respectively. The additional sensing modalities were included to reduce the false alarm rate in traditional IMU-based detectors. In a series of experiments where a pedestrian walked close-loop trajectories on different terrains, including flat planes, stairs, and ramps, navigation accuracy of the ZUPT-aided INS was improved by 25% when using the developed DVS-SHOE detector, as compared to the case of using the SHOE detector. In another series of experiments where a pedestrian ran along a 42.6 [m] straight-line trajectory, the experimental results show that using the developed UA-SHOE detector for the ZUPT-aided INS led to more than a 2$\times$ improvement on the navigation accuracy, as compared to the case of using the SHOE detector.

Chapter 5

On Algorithm Assumption – Bypassing Binarism by Using Adaptive Covariance

5.1 Introduction

This chapter presents the development of an adaptive covariance allowing for bypassing the necessity of using a stance phase detection in implementation of ZUPT-aided INS and reducing the impact of modeling error caused residual velocities of a foot during the stance phases on navigation performance. Figure 5.1 illustrates the concept of the developed adaptive mechanism. The rest of this chapter is organized as follows. Section 5.2 derives the developed adaptive covariance, Section 5.3 validates the developed approach with indoor pedestrian navigation experiments and discusses experimental results, and Section 5.2.5 concludes the chapter.

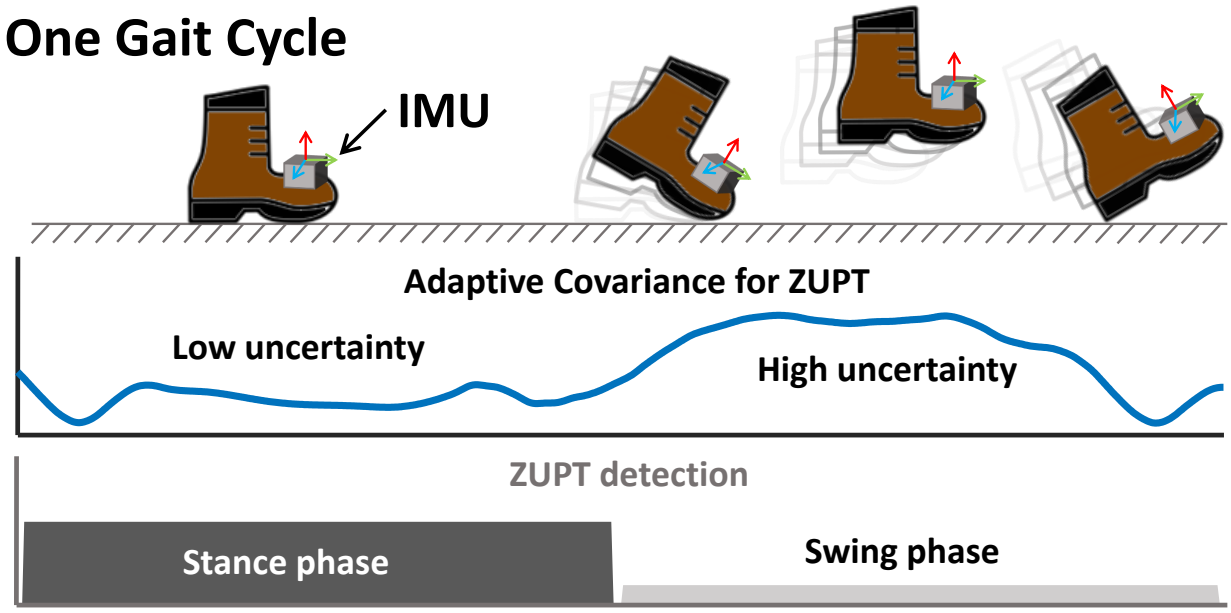


Figure 5.1: Concept of Foot-Instability-Based-Adaptive (FIBA) Covariance for ZUPT-aided INS.

5.2 ZUPT-aided INS Using FIBA Covariance

This section first gives an overview of the developed adaptive mechanisms and then discusses the derivation of the log-likelihood ratio that quantifies the foot instability level, the design of the developed Foot-Instability-Based Adaptive (FIBA) covariance, parameters selection for the FIBA covariance, and properties of the EKF innovation sequences in the developed ZUPT-aided INS.

5.2.1 Concept Overview

This section presents the mechanism and implementation of a ZUPT-aided INS with a FIBA covariance in an EKF framework. The developed mechanism, shown in Figure 5.2(b), aims to minimize one of the modeling errors in the ZUPT algorithm by using an adaptive measurement covariance matrix to avoid feeding back the zero-velocity measurements with high confidence when an inertial sensor mounted on a foot is not completely stationary. Com-

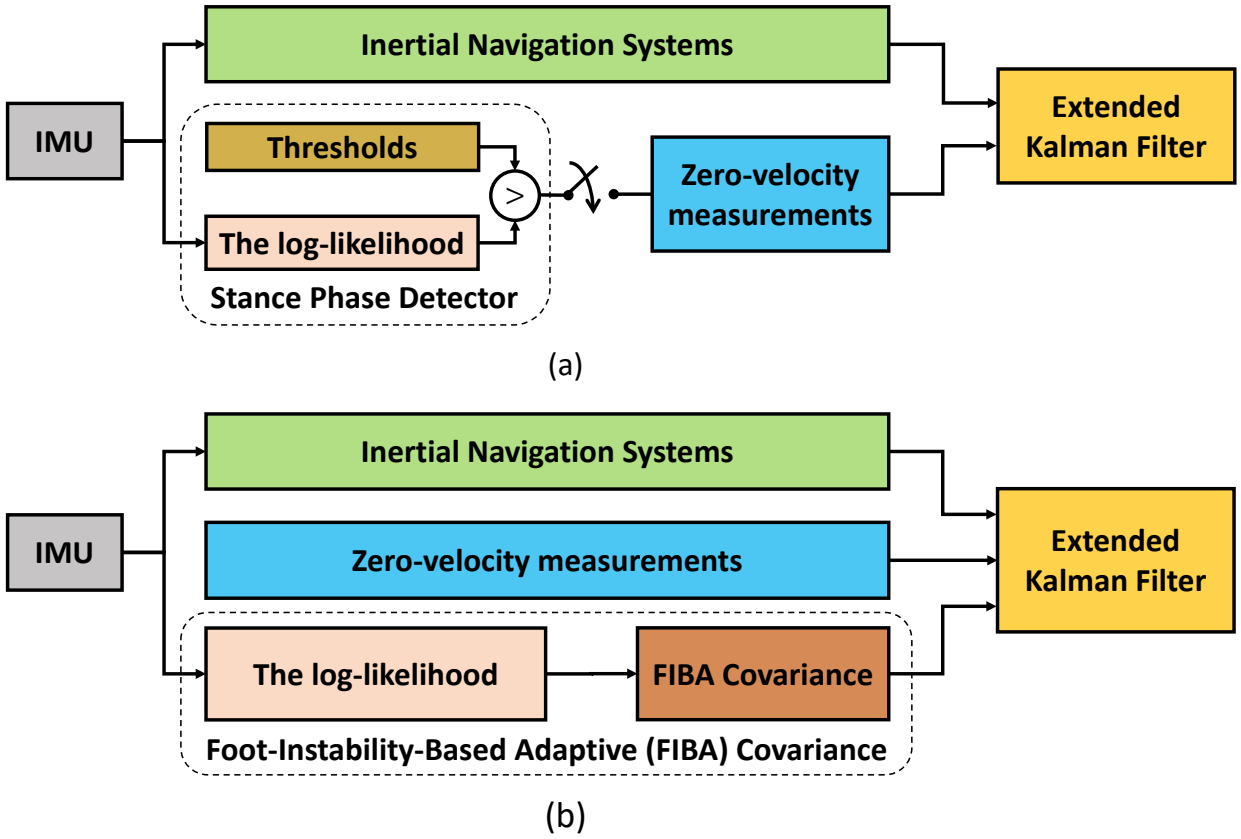


Figure 5.2: (a) A conventional ZUPT-aided INS using the stance phase detector with a threshold.(b) The developed ZUPT-aided INS implemented in an EKF with the developed FIBA covariance. The developed system does not use a stance phase detector and adopts a covariance matrix for zero-velocity measurements that varies in each iteration based on instability level of a foot.

pared to the conventional ZUPT-aided INS, shown in Figure 5.2(a), the developed algorithm feedbacks the pseudo measurements of zero velocity at every time instance in the EKF with the measurement covariance matrix updated by the FIBA covariance that varies according to an instability level of a pedestrian's foot. The foot instability level is quantified in this section with a log-likelihood ratio statistics calculated based on readings from the foot-mounted IMU. The FIBA covariance is designed to have a significantly higher value when an IMU mounted on the foot is unstable, which are the cases when the IMU experiences forces other than the gravity of the Earth. The higher value of the FIBA covariance in these cases causes the feedback of zero-velocity measurements to have a numerically minimal effect on other navigation states. When the sensor unit is stable, the FIBA covariance automatically declines to low values that are sufficient to compensate for residual velocities of a standalone INS. This property of the FIBA covariance not only reduces the modeling error of the ZUPT algorithm but also enables an alternative implementation of the ZUPT-aided INS without using a stance phase detector.

The developed ZUPT-aided INS using the FIBA covariance is implemented in an EKF framework, which is illustrated by the block diagram presented in Figure 5.2(b). The propagation step and the update step of the EKF used in this section are similar to the system discussed in [216]. However, in implementation presented in this section, the zero-velocity measurement covariance matrix $\mathbf{R}_{\text{ZUPT}}$ is time-varying and adjusted according to the developed FIBA covariance in each update step of the EKF.

5.2.2 Modeling Instability of Foot Dynamics

The FIBA covariance for zero-velocity measurements aims to provide a low uncertainty when a sensor unit mounted on a foot is stable, which is the case of the sensors being completely stationary, and a high uncertainty when the sensor is unstable, which is the case of the

sensors experiencing a motion. The moving case and stationary case are denoted as H_0 and H_1, respectively. To model the instability level of the sensor with a metrics, this section derives a Likelihood Ratio (LR) statistics that describes the probabilities of occurrence of any of the two cases based on IMU measurements $\mathbf{z}_n$. In this section, the logarithm of the derived LR statistics, which is the log-likelihood ratio, is considered as a metrics that describes the instability level of a pedestrian's foot.

The derivation of the LR statistics is similar to the derivation of the SHOE detector presented in [181]. The inertial sensor measurements $\mathbf{y}_k$ are modeled using the following notations:

$$\mathbf{y}_k = \begin{bmatrix} \mathbf{y}_k^\alpha \\ \mathbf{y}_k^\omega \end{bmatrix} = \begin{bmatrix} \mathbf{s}_k^\alpha \\ \mathbf{s}_k^\omega \end{bmatrix} + \begin{bmatrix} \mathbf{v}_k^\alpha \\ \mathbf{v}_k^\omega \end{bmatrix} = \mathbf{s}_k + \mathbf{v}_k.$$

Here, $\mathbf{s}_k^\alpha \in \mathbb{R}^3$ and $\mathbf{s}_k^\omega \in \mathbb{R}^3$ denote the IMU-experienced acceleration and angular rate, respectively. Vectors $\mathbf{v}_k^\alpha \in \mathbb{R}^3$ and $\mathbf{v}_k^\omega \in \mathbb{R}^3$ represent the measurement noise of the accelerometer and gyroscope, respectively. This derivation used an assumption that the measurement noise of accelerometers and gyroscopes has two components. One of the components is described by independent and identically-distributed white Gaussian noises. The other component is stochastic biases. It was considered that the stochastic biases could be minimized by subtracting IMU measurements with bias states of the EKF used in ZUPT-aided INS. Therefore, in this derivation, the measurement noise of accelerometers and gyroscopes were modeled as white Gaussian noises with respective variances σ_α^2 and σ_ω^2.

In the case H_0, signal patterns of foot-mounted IMU measurements are unknown. In the case H_1, we hypothesize that the accelerometer experiences only the gravitational acceleration, and the angular rate experienced by the gyroscope is zero. More formally, for the two cases,

the sensor measurements are assumed to satisfy the following conditions:

$$H_0 : \exists k \in \Omega_n, \mathbf{s}_k^\alpha \neq g\mathbf{u}_n, \mathbf{s}_k^\omega \neq 0_{3\times 1},$$
$$H_1 : \forall k \in \Omega_n, \mathbf{s}_k^\alpha = g\mathbf{u}_n, \mathbf{s}_k^\omega = 0_{3\times 1},$$

where $\mathbf{u}_n$ is a 3×1 unit vector, g is the gravitational constant, and $\Omega_n = \{l \in \mathbb{N}, n \leq l < N-1\}$ is a collection of the sensor measurement indexes at time n with a window of length N.

Following the derivation described in [181], the probability density function (pdf) of collected IMU measurements z_n under H_0 is expressed as

$$p(\mathbf{z}_n; H_0) = \frac{1}{(2\pi\sigma_\alpha^2)^{3N/2}(2\pi\sigma_\omega^2)^{3N/2}}, \tag{5.1}$$

where $\mathbf{z}_n = \{\mathbf{y}_k\}_{k=n}^{k=N-1}$. The pdf under H_1 is given by

$$p(\mathbf{z}_n; H_1) = \frac{1}{(2\pi\sigma_\alpha^2)^{3N/2}}\exp(-\frac{1}{2\sigma_\alpha^2}\sum_{k\in\Omega_n}(\parallel \mathbf{y}_k^\alpha - g\frac{\bar{\mathbf{y}}_k^\alpha}{\parallel \bar{\mathbf{y}}_k^\alpha \parallel}\parallel^2)\frac{1}{(2\pi\sigma_\omega^2)^{3N/2}}\exp(-\frac{1}{2\sigma_\omega^2}\sum_{k\in\Omega_n}(\parallel \mathbf{y}_k^\omega \parallel^2), \tag{5.2}$$

where

$$\bar{\mathbf{y}}_k^\alpha = \frac{1}{N}\sum_{k\in\Omega_n}\mathbf{y}_k^\alpha.$$

The LR statistics $L(\mathbf{z}_n)$ is derived from (5.1) and (5.2), having the following form

$$L(\mathbf{z}_n) = \frac{p(\mathbf{z}_n; H_0)}{p(\mathbf{z}_n; H_1)} = \exp(\sum_{k\in\Omega_n}\frac{1}{2\sigma_\alpha^2}\parallel \mathbf{y}_k^\alpha - g\frac{\bar{\mathbf{y}}_k^\alpha}{\parallel \bar{\mathbf{y}}_k^\alpha \parallel}\parallel^2 + \frac{1}{2\sigma_\omega^2}\parallel \mathbf{y}_k^\omega \parallel^2). \tag{5.3}$$

An example of the LR statistics profile collected in an indoor walking-and-running experiment is demonstrated in Figure 5.3(a). It should be noted that the LR statistics is always positive.

This section quantifies the foot instability level with a log-likelihood ratio. The log-likelihood ratio, denoted as $S(\mathbf{z}_n)$, takes a scaled version of the logarithm of (5.3) to increase the numerical discrepancy of small values, which corresponds to the case when a foot is relatively stable. The log-likelihood ratio $S(\mathbf{z}_n)$ is expressed as follows:

$$S(\mathbf{z}_n) = \frac{2}{N}\log(L(\mathbf{z}_n)) = \frac{1}{N}(\sum_{k\in\Omega_n}(\frac{1}{\sigma_\alpha^2} \parallel \mathbf{y}_k^\alpha - g\frac{\bar{\mathbf{y}}_k^\alpha}{\parallel \bar{\mathbf{y}}_k^\alpha \parallel} \parallel^2 + \frac{1}{\sigma_\omega^2} \parallel \mathbf{y}_k^\omega \parallel^2)). \tag{5.4}$$

Figure 5.3(b) gives an example of the log-likelihood ratio profile collected in the same experiment involving indoor walking and running. It can be seen that (5.4) gives a lower value in the case of the sensor being stationary than in the case of the moving sensor. Two phenomena can be noticed in Figure 5.3(b). First, in the case where an IMU is traveling at a constant velocity without any rotation, the value of the log-likelihood ratio would be on the same level as in the case of the IMU being completely stationary. However, based on our observations in pedestrian navigation experiments, the constant velocity scenarios are unlikely to occur when the IMU is mounted on a shoe. Second, the log-likelihood ratio has an identical expression to the statistics metrics used by the SHOE detector. However, the statistics metrics and the log-likelihood ratio play two different roles. In the SHOE detector, the statistics metrics is utilized to compare with a threshold for zero-velocity event detection. In this section, the log-likelihood ratio is considered as an approach to quantify the instability level of a pedestrian's foot based on inertial sensor readings, enabling the covariance matrix of zero-velocity measurements to dynamically adjust its values in different scenarios.

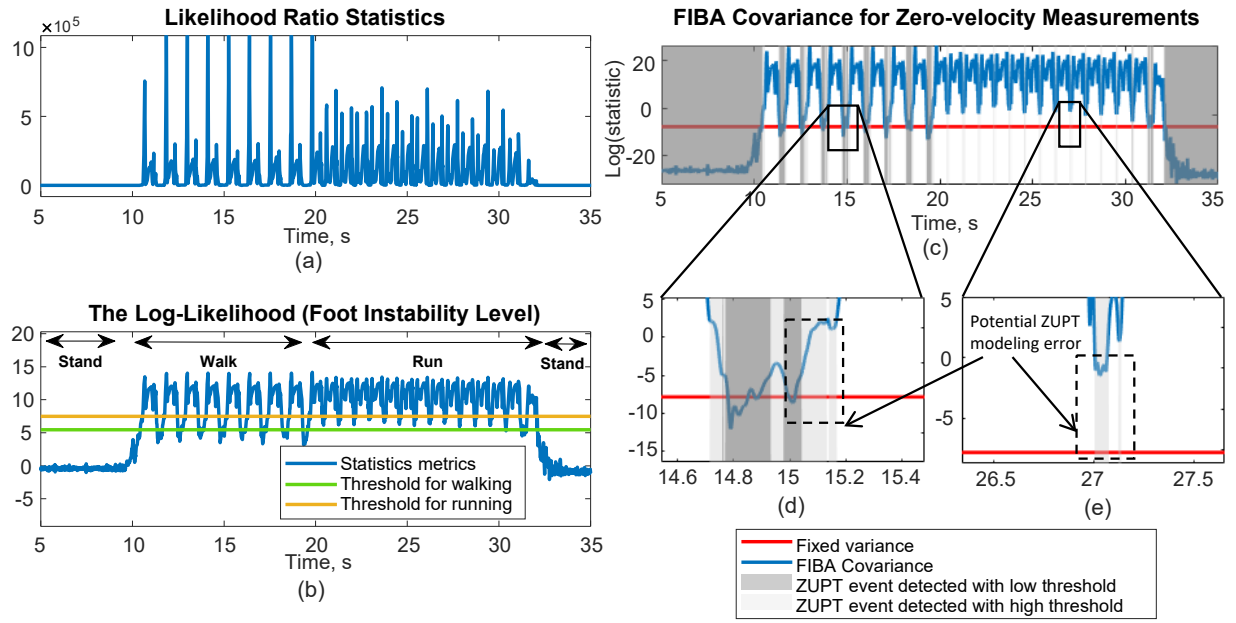


Figure 5.3: (a) A profile of the Likelihood Ratio statistics, expressed in (5.3), in an indoor pedestrian navigation experiment discussed in Section 5.2.4. (b) The blue curve illustrates a profile of the log-likelihood ratio in the same experiment, expressed in (5.4). The green line and the blue line represent the low threshold and the high threshold used for the SHOE detector, respectively, in the experiments discussed in Section 5.3.1. (c) The blue curve shows a profile of the developed Foot-Instability-Based Adaptive (FIBA) covariance. The red horizontal line indicates the value of 0.02. The dark gray areas indicate the stance phase detected by the SHOE detector with the threshold value specified by the green line in (b). The light gray areas mark the stance phase detected by the SHOE detector with the threshold value specified by the yellow line in (b). (d) and (e) display zoomed-in versions of (c), showing two scenarios, marked with the dashed rectangles, that a pedestrian's foot was unstable, but a stance phase was detected.

5.2.3 The Foot-Instability-Based Adaptive (FIBA) Covariance

The developed FIBA covariance is designed to have values that are varying based on the derived log-likelihood ratio in (5.3). The derived log-likelihood ratio demonstrates a desired property where the metrics decreases in the stable case and increases in the unstable case. However, the values might not be suitable for selection as an uncertainty of the zero-velocity measurements. This section hypothesizes that the appropriate uncertainty for zero-velocity measurements can be achieved by scaling the log-likelihood ratio. The FIBA covariance, $\sigma_{\text{FIBA}}(\mathbf{z}_n)$, is expressed as follows:

$$\sigma_{\text{FIBA}}(\mathbf{z}_n) = \beta S(\mathbf{z}_n)^{\gamma},\ \beta \in \mathbb{R}^{+},\ \gamma \in \mathbb{R} \tag{5.5}$$

In (5.5), β and γ are hyper-parameters of the FIBA covariance that need to be selected. Note that

$$\sigma_{\text{FIBA}}(\mathbf{z}_n) > 0, \forall \mathbf{z}_n.$$

The developed FIBA covariance $\sigma_{\text{FIBA}}(\mathbf{z}_n)$ is used to update the covariance matrix $\mathbf{R}_{\text{ZUPT}}(\mathbf{z}_n)$ for the zero-velocity measurements in each iteration in the EKF framework. It is assumed that the zero-velocity measurements are uncorrelated and that uncertainties for the measurements along the 3-axis are identical. $\mathbf{R}_{\text{ZUPT}}(\mathbf{z}_n)$ is presented as follows:

$$\mathbf{R}_{\text{ZUPT}}(\mathbf{z}_n) = \begin{bmatrix} \sigma^2_{\text{FIBA}}(\mathbf{z}_n) & 0 & 0 \\ 0 & \sigma^2_{\text{FIBA}}(\mathbf{z}_n) & 0 \\ 0 & 0 & \sigma^2_{\text{FIBA}}(\mathbf{z}_n) \end{bmatrix} \tag{5.6}$$

The diagonal structure of $\mathbf{R}_{\text{ZUPT}}(\mathbf{z}_n)$ and the positivity property of the term $\sigma^2_{\text{FIBA}}(\mathbf{z}_n)$ guar-

antee $\mathbf{R}_{\text{ZUPT}}(\mathbf{z}_n)$ to be a proper covariance matrix.

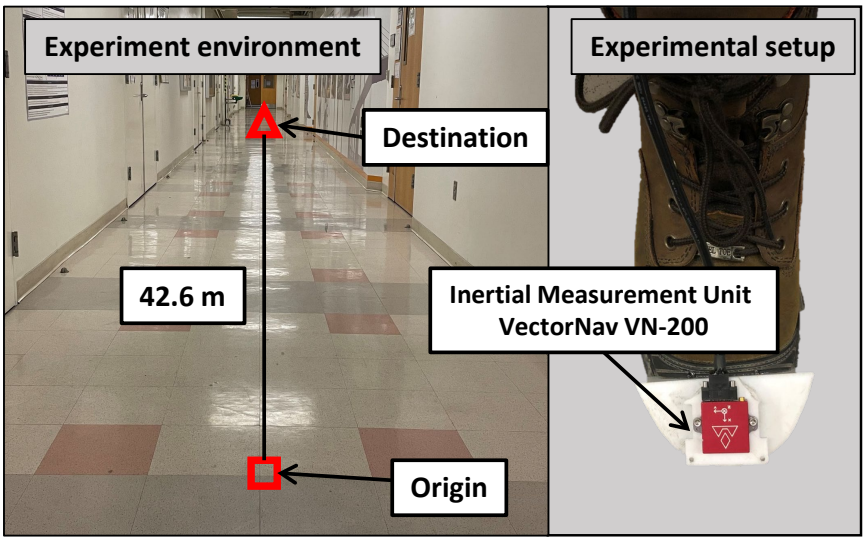


Figure 5.4: The scenario and setup of the experiments discussed in Section 5.2.4. The red square indicates the nominal starting point of the experiment, and the red triangle marks the nominal destination. The distance between the starting point and the destination was 42.6 m, which was measured by a ruler. The IMU used in the experiments was a VectorNav VN−200 IMU. The sensor was mounted on a fixture securely attached to the toe side of the boot. The sampling rate was set to 800 Hz.

5.2.4 Hyper-Parameter Selection

This section uses a data-driven approach to estimate values of the hyper-parameters β and γ.Pedestrian navigation experiments were conducted in the Engineering Gateway Building at the University of California, Irvine. The experimental setup and scenarios are shown in Figure 5.4. The IMU VectorNav VN−200 was mounted on the toe side of the right shoe. In the experiments, a pedestrian first walked at a speed of approximately 60 step/sec on a straight line for 20 m and then ran at a speed of approximately 100 step/sec along the line for 22.6 m. An example of the trajectory inside the building is illustrated in Figure 5.5. The pedestrian repeated the same experiments 10 times. The relative ground truth location of the destination was determined by a ruler.

The selection of hyper-parameters aims to minimize navigation errors. Based on IMU mea-

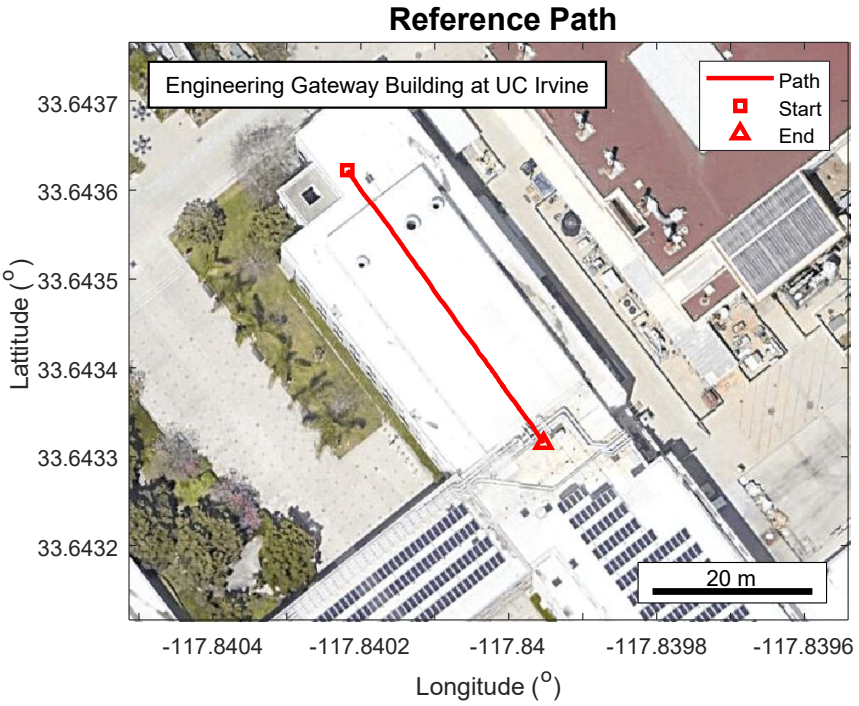


Figure 5.5: A reference path for indoor pedestrian navigation experiments, described in Section 5.2.4.

surements collected in the experiments, we implemented the ZUPT-aided INS with the developed FIBA covariance and swept the values of β with values $\left[e^{-15}, \ e^{-14.5}, \ e^{-14.0}, \ ..., \ e^{15.0}\right]$ and γ with values $\left[-3, \ -2.5, \ -2, \ ..., \ 3\right]$. For each pair of β and γ values, we calculated the RMSEs based on the loop-closure errors of the ten experiments. Figure 5.6 presents the RMSEs when different values of β and γ were used. The hyper-parameters that corresponded to the minimum error are summarized in Table 5.1.

Table 5.1: Hyper-parameters for the FIBA covariance

Hyper-parameter	**Value**
β	$\exp(-4.5)$
γ	1.8

5.2.5 Discussion

With $\beta = e^{-4.5}$ and $\gamma = 1.8$, an example of profile of the FIBA covariance is illustrated in Figure 5.3(c), and two zoomed-in versions are shown in Figure 5.3(d) and Figure 5.3(e), respectively. The blue curves represent the value of the FIBA covariance, varying based on the IMU measurements collected in the experiments described in Section 5.2.4, and the red horizontal line indicates a value of the variance of the zero-velocity measurements that are commonly used in other pedestrian navigation systems using foot-mounted IMUs. The dark and the light gray areas in 5.3(c) illustrate the stance phases detected by the SHOE detector with a threshold represented by the green and the yellow horizontal lines in 5.3(b), respectively. The following three features can be observed in Figure 5.3(c), Figure 5.3(d), and Figure 5.3(e).

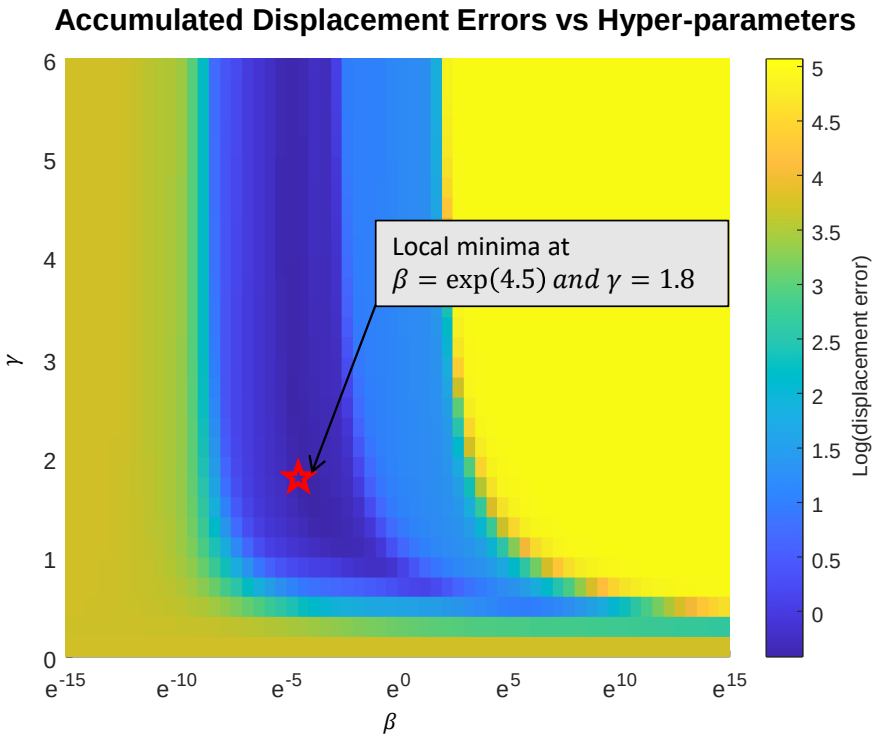


Figure 5.6: The accumulated position errors at the destination estimated by the ZUPT-aided INS using the developed FIBA covariance with different values of the hyper-parameter β and γ in the indoor pedestrian navigation experiments discussed in Section 5.2.4. The minimum displacement error, marked with the red pentagram, occurred at $\beta = e^{-4.5}$ and $\gamma = 1.8$.

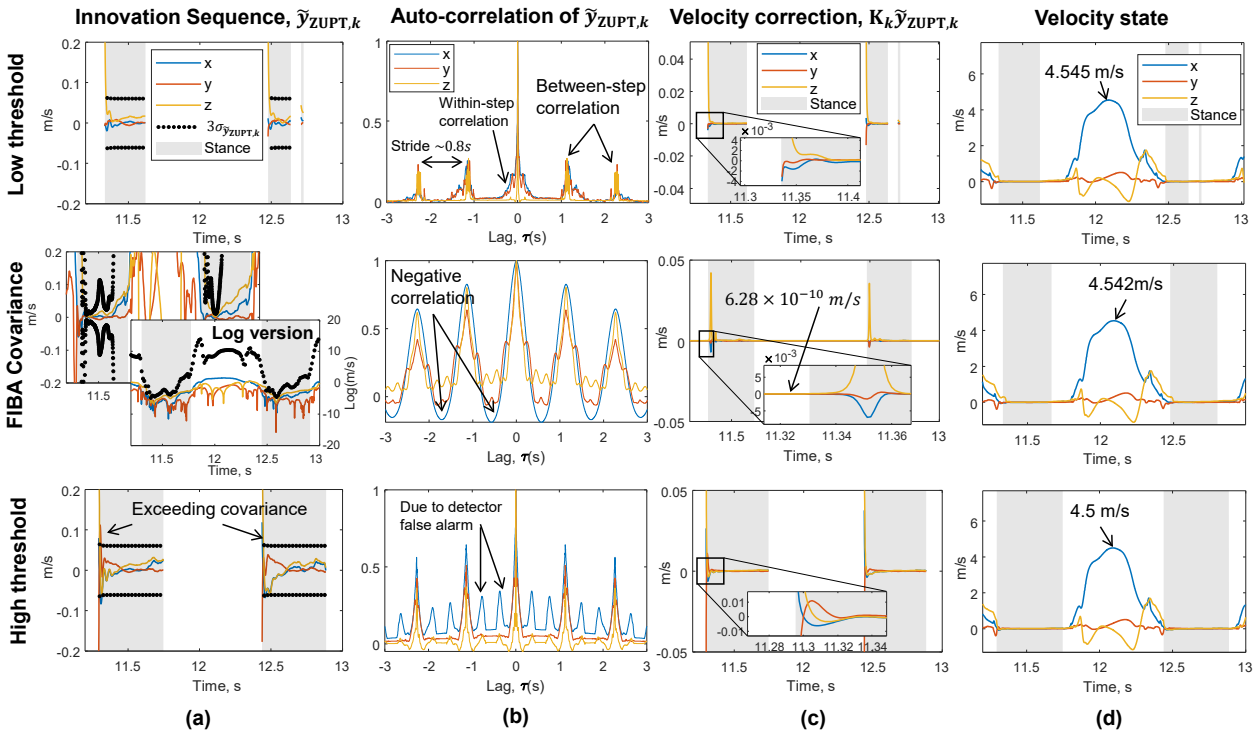


Figure 5.7: (a) examples of the EKF innovation sequences, $\tilde{y}_{\mathrm{ZUPT}}$, and 3× square-root of the EKF innovation covariances, $\sigma_{\tilde{y}_{\mathrm{ZUPT}}}$. It can be seen that the innovation sequences in the cases of using the low threshold and the high threshold do are not continuous because the conventional ZUPT-aided INS performs the update step only during the stance phase. The grey areas represent the stance phases identified by the SHOE detector. A logarithm version of the innovation sequence in the ZUPT-aided INS using the FIBA covariance is presented next to the regular version for better visualization of large values. The developed FIBA covariance does not require the binary stance phase detection, but the stance phase periods are displayed to illustrate the status of the foot. (b) profiles of auto-correlation calculated based on the innovation sequences, $\tilde{y}_{\mathrm{ZUPT}}$, presented in (a). (c) amount of velocity corrected in each iteration of the EKF update step. It can be observed that, even though the ZUPT-aided INS using the FIBA covariance feedbacks the zero-velocity measurements regardless of the stance phase and the swing phase, the corrections applied to the velocity states when the foot was very unstable were minimal, which was 6.28×10^{-10}. (d) the estimated velocities. All profiles shown in this figure corresponded to the IMU measurements collected during one complete gait cycle in the walking part of the experiments discussed in Section 5.2.4.

- In Figure 5.3(c), the FIBA covariance during the stance phase decreases to a comparable level defined by the red line, which marked a value commonly used for the variance of zero-velocity measurements in implementation of conventional ZUPT-aided INS. The decrease indicates that the zero-velocity measurements have low uncertainty, and the velocity error would be reduced during this period. During the swing phase, the FIBA covariance increases sharply, leading to a result that the zero-velocity measurements have high uncertainty, and the velocity state in the navigation solutions would not be numerically affected by the zero-velocity feedback. This property of the developed FIBA covariance eliminates the need to use a stance phase detector for the ZUPT-aided INS.

- In the period pointed by the black arrow in Figure 5.3(d), a stance phase highlighted in light gray is detected, but the instability of the shoe is higher than in other regions of the stance phase. In our opinion, the instability was due to the fact that the foot of a pedestrian could still move slightly during the stance phase, and the movement generated acceleration and angular velocity that contributed to higher values of the log-likelihood ratio. In conventional ZUPT-aided INS using a stance phase and a constant measurement variance, the filter could be over-confident in the zero-velocity measurements, resulting in a modeling error. With the FIBA covariance, the uncertainty was automatically tuned to a higher number, reducing the impact of the modeling error on navigation accuracy.

- Figure 5.3(e) shows a segment of the FIBA covariance collected when the pedestrian was running. In this period, the SHOE detector with a threshold indicated by the green line in Figure 5.3(b) was not able to detect the stance phase. The SHOE detector with the threshold indicated by the yellow line in Figure 5.3(b) could identify a stance phase, but the usage of the constant covariance could introduce a modeling error because instability of the foot when running should be higher than in the case of walking. In

this example, the value of the FIBA covariance in the running case was higher than in the walking case, indicating the modeling error is reduced when the FIBA covariance was used for the ZUPT-aided INS.

5.2.6 The Zero-velocity Measurement Model

A ZUPT-aided INS uses the pseudo-zero-velocity measurements to correct the velocity states in the update step of the EKF. In a traditional implementation of the system, illustrated in Figure 5.2(a), the zero-velocity measurements are applied only when a stance phase is detected. The implementation of our developed ZUPT-aided INS using the FIBA covariance, which is described in Figure 5.2(b), feedbacks the zero-velocity information in every iteration of the EKF, regardless of the stance phase or the swing phase. The different implementations lead to distinct properties of the zero-velocity measurement model, the quality of which can be evaluated by reviewing innovation sequences of the EKF[143]. In this section, we investigate the properties of the measurement model by studying the innovation sequences, $\tilde{y}_{\text{ZUPT}}$, innovation covariances, $\sigma^2_{\tilde{y}_{\text{ZUPT}}}$, auto-correlation of the innovation sequences, velocity state propagation, and velocity state correction $\text{K}_k\tilde{y}_{\text{ZUPT}}$, which is a Kalman gain K_k multiplied by the innovation sequences $\tilde{y}_{\text{ZUPT}}$.

Figure 5.7 compares the properties of zero-velocity measurement models in the developed ZUPT-aided INS using FIBA covariance, the conventional ZUPT-aided INS with the SHOE detector using a low threshold indicated by the green line in Figure 5.3(b), and the conventional ZUPT-aided INS with the SHOE detector using a high threshold described by the yellow line in 5.3(b). It is worth noting that, in this dataset, the low threshold was a preferred value for walking, and the high value was favorable in the case of running. All profiles shown in Figure 5.7 corresponded to the IMU measurements collected during one complete gait cycle in the walking part of the experiments discussed in Section 5.2.4. Interpretations

of the plots shown in Figure 5.7 are discussed in the following paragraphs.

The Zero-velocity Innovation Sequences

A few observations can be made based on the innovation sequences shown in Figure 5.7(a). First, it can be seen that the innovation sequences have different profiles in different implementations of the ZUPT-aided INS. In the cases of the conventional ZUPT-aided INS using both the low and high thresholds, the innovation sequences were not continuous because the update step of the EKF was performed only when a stance phase was detected. Second, in all the cases, the innovation sequences along the z-axis had larger values than those along the x- and the y-axis. This phenomenon was considered a result of accelerometer's bandwidth of the VN−200 IMU, which was 260 Hz, being insufficient to fully reconstruct the forces experienced by the foot-mounted IMU during heel striking phases. The insufficient bandwidth of an IMU in foot-mounted INS has been reported in [101]. Third, the innovation sequences along the x- and the y-axis in the cases of FIBA covariance and the low threshold were bounded by the dotted $3\sigma_{\tilde{y}_{\text{ZUPT}}}$ curves in Figure 5.7(a), while the innovation sequence in the case using the high threshold frequently exceeded the $3\sigma_{\tilde{y}_{\text{ZUPT}}}$ curve. The occurrence of exceeding innovation sequences is considered a sign of inappropriate setting of the threshold in the stance phase detection, which could introduce unmodeled errors into the navigation systems.

Correlation of the Zero-velocity Measurements

Figure 5.7(b) compares statistics of auto-correlations of the innovation sequences $\tilde{y}_{\text{ZUPT}}$. Five observations can be made in Figure 5.7(b):

- The auto-correlations in the three cases show the innovation sequences are correlated

within lags of approximately 0.3 seconds. Since stride periods during walking in this experiment were around 0.8 seconds, this phenomenon suggests that the innovation sequences were correlated within a step, which has been reported and referred to as the within-step correlation in [143].

- The auto-correlations in the cases of the low threshold and the high threshold have peaks at lags of approximately every one second. The presence of the peaks indicates that the zero-velocity measurements between two steps were correlated. In [143], this type of correlation was referred to as the between-step correlation. Although the within-step and the between-step correlations are in contrast to the assumptions of EKF and can lead to navigation errors, it is typically assumed that the correlations would die out within a short period of time.

- The correlation of the innovation sequence in the case of using the high threshold was increased, as compared to the low threshold case. In our opinion, the increase can be interpreted as a presence of zero-velocity measurements biases, which was introduced by an inappropriate setting of thresholds in the stance phase detection. In implementation of the conventional ZUPT-aided INS, the additional navigation errors brought by the high correlation can be reduced by improving performance of stance phase detection[88].

- The auto-correlation in the case of the FIBA covariance had increased within-step and between-step correlations, as compared to those in the other two cases. Moreover, the auto-correlation along the x-axis shows negative values at lags of around every 0.5 seconds. In this dataset, 0.5 seconds was approximately the period between the middle points of a stance phase and a swing phase that were adjacent to each other. Our explanation for the increase in correlations was that in the developed ZUPT-aided INS using the FIBA covariance, the zero-velocity measurements during the entire experiment were utilized in the EKF update step, and the innovation sequences during

the swing phases had large biases. While developing the ZUPT-aided INS using the FIBA covariance, this phenomenon was taken into account, and the FIBA covariance handles the highly correlated zero-velocity measurements during the swing phases by increasing the measurement covariance matrix to significantly larger values.

Velocity Correction During the Swing Phase

In Figure 5.7(c), it can be seen that, in cases of the low threshold and the high threshold, velocity corrections had the maximum value at the beginning of each detected step, and the magnitude of the corrections in the latter case was larger than the former case. Since the high threshold was an inappropriate setting for walking, the large correction in this case can lead to reduced accuracy in velocity estimation. The inaccuracy velocity estimation can be seen in Figure 5.7(d), where the difference in the peaks of velocity states, in the case of the low and the high thresholds, was 0.041 m/s. In the case of the FIBA covariance, the maximum correction did not occur at the beginning of a stance phase but at the moment when the foot had the highest stability, which is illustrated in Figure 5.7(c). In Figure 5.7(c), it could also be observed that, even though the ZUPT-aided INS using the FIBA covariance feedbacks the zero-velocity measurements regardless of the stance phase and the swing phase, the corrections applied to the velocity states when the foot was very unstable were minimal, which was 6.28×10^{-10} in the case shown in Figure 5.7(c). This number can be considered to have an insignificant impact to the estimation accuracy of the velocity states during the swing phases in a short- to mid-term navigation mission. As presented in Figure 5.7(d), the difference in the peaks of velocity states in the case of the low threshold and the FIBA covariance was less than 0.003 m/s. In this dataset, it was unclear about whether the case using FIBA covariance had a better velocity estimation accuracy or the one with the low threshold. Alternative localization systems, for example, the camera system described in [15], are needed to verify this information.

Based on properties of the innovation sequences demonstrated in Figure 5.7, we concluded that even though the ZUPT-aided INS using the FIBA applied the zero-velocity measurements, which were highly correlated, regardless of the stance phase and the swing phase, the developed system only gave an influential update to the velocity state when the foot was stable. During the swing phases when the foot was unstable, the velocity correction could be considered insignificant and would not impact the velocity estimation accuracy in this experiment.

5.3 Experimental Validation

To investigate the navigation performance of the developed ZUPT-aided INS using the FIBA covariance, two series of experiments were conducted in the Engineering Gateway building at the University of California, Irvine. The first series investigated the navigation performance in the case of traveling at two different speeds. The second series evaluated the performance in the case of traveling on different terrains. The experimental setup used for the two series of experiments is shown in Figure 5.4. A VectorNav IMU VN−200 was mounted on a customized fixture which was firmly attached to the toe side of the pedestrian's right shoe. The IMU was connected to a laptop held by the pedestrian for data recording. The sampling rate of the IMU was set to 800 Hz. The EKF noise parameters, including VRW σ_{VRW}, ARW σ_{ARW}, RRW σ_{RRW}, and AcRW σ_{AcRW}, had values listed in Table 5.2.

Table 5.2: EKF Parameter settings for the ZUPT-aided INS

EKF parameter	**Value**
σ_{ARW}	2.1597×10^{-5}
σ_{VRW}	4.8557×10^{-4}
σ_{RRW}	1.7141×10^{-6}
σ_{AcRW}	1.3873×10^{-6}

5.3.1 Different Traveling Speeds

The first series of experiments was the same as the experiments conducted for parameter selection of the FIBA covariance, which is described in Section 5.2.4. A reference trajectory is shown in Figure 5.5. In this experiment, we evaluated the navigation accuracy of the ZUPT-aided INS using the SHOE detector with a low threshold specified by the green line in 5.3(b), the SHOE detector with a high threshold illustrated by the yellow line in 5.3(b), and the FIBA covariance. The zero-velocity variance used for the conventional ZUPT-aided INS was set to 0.02 m/s. For the horizontal displacement error, CEP, which is a circle centered at the ground truth location with a radius enclosing 50% of the data, was used. For the vertical displacement error, the RMSE based on the estimated destination was calculated.

The navigation solutions are presented in Figure 5.8. It can be observed that the horizontal displacement errors of destinations estimated by solutions that used the low threshold and the FIBA covariance had similar values, which were 0.62 m and 0.64 m, respectively. The horizontal CEP in the case of using the high threshold was increased to 1 m. The vertical RMSEs in the case of the FIBA covariance was 0.34, which was smaller than the other two localization solutions. Ratios of position errors and trajectory lengths in this series of experiments are summarized in Table 5.3.

Table 5.3: Percentage of position error in trajectory length for the 1st series of experiments

	Low threshold	**FIBA covariance**	**High threshold**
Horizontal	1.46%	1.51%	2.35%
Vertical	1.01%	0.8%	2.24%

Two observations can be made in this series of experiments. First, the ZUPT-aided INS that was using a high threshold had larger displacement errors because the high threshold led to feeding the zero velocity measurements to the EKF when the foot was not stable, resulting in additional modeling errors. The developed ZUPT-aided INS using the FIBA covariance also feedbacked zero-velocity measurements during this period, but the modeling error was

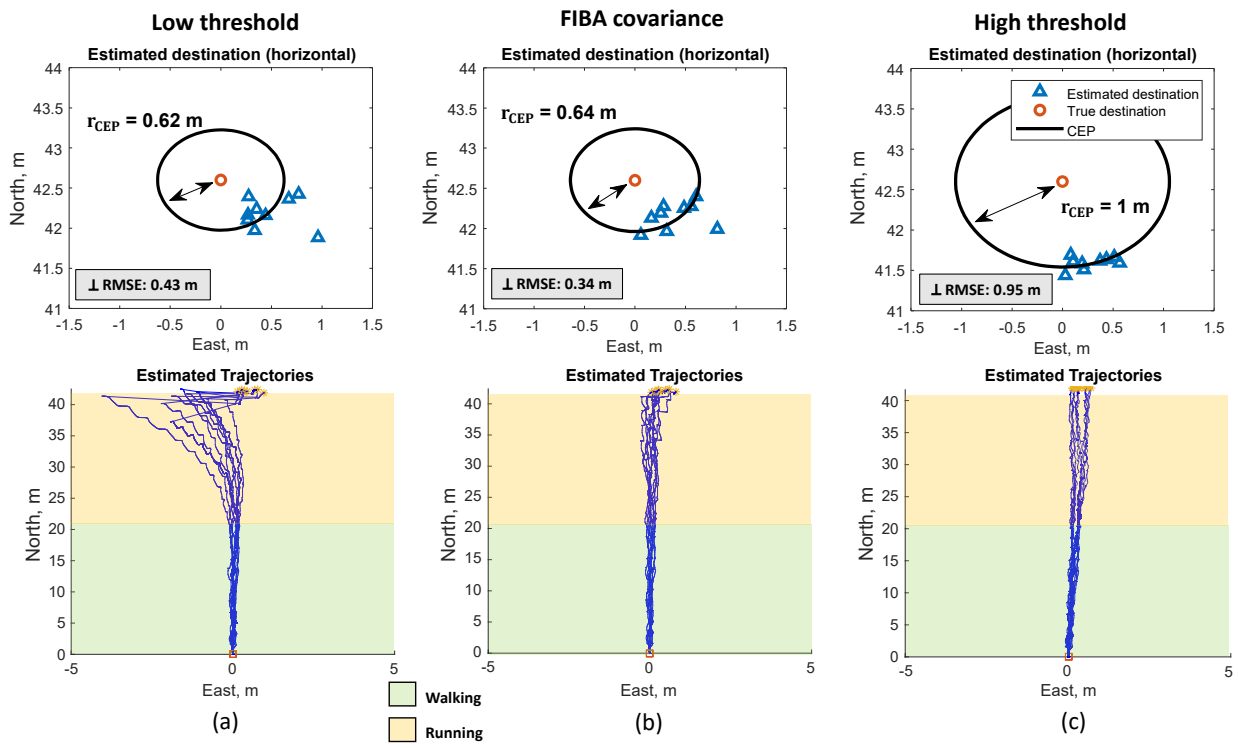


Figure 5.8: (a) Trajectories and destinations estimated by the ZUPT-aided INS using the SHOE detector with a low threshold in the first series of experiments described in Section 5.3.1. The value of the low threshold is indicated by the green line in Figure 5.3(b). (b) Trajectories and destinations estimated by the ZUPT-aided INS using the developed FIBA covariance in the same experiments. The hyper-parameters used for the FIBA covariance are summarized in Table 5.1. (c) Trajectories and destinations estimated by the ZUPT-aided INS using the SHOE detector with a high threshold in the experiments. The value of the high threshold is represented by the yellow line in Figure 5.3(b). In these experiments, the ZUPT-aided INS using the FIBA covariance reduced the maximum displacement errors from 4m to less than 1 m, as compared to the case using the low thresholds. When compared with solution using the high threshold, the FIBA covariance improved navigation performance by of 36% horizontally and 64% vertically.

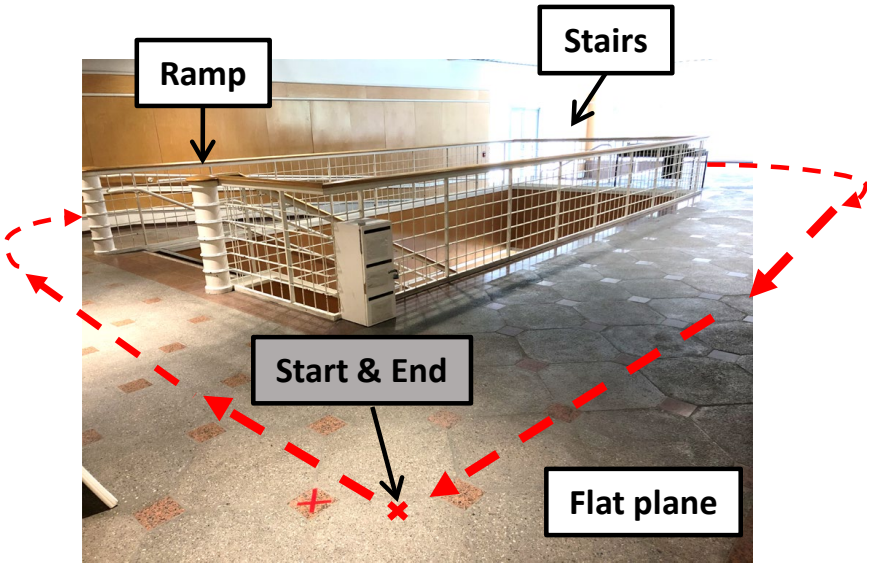


Figure 5.9: The scenario of the second series of experiments discussed in Section 5.3.2. The trajectories in the experiments included the segments of terrains of flat planes, stairs, and a ramp.

reduced because the uncertainty of the zero-velocity measurements automatically adjusted to a higher value. Second, although the displacement errors at the destinations estimated by the ZUPT-aided INS using the low threshold had errors less than one meter, it can be seen that the estimated trajectory in Figure 5.8(a) drifted away when the pedestrian started to run and the maximum error was 4 m along the east direction. The drift was observed because the low threshold was not able to detect the stance phase, and hence no ZUPT algorithm was triggered during this period. In this series of experiments, the developed ZUPT-aided INS using the FIBA covariance demonstrated an improvement in the localization accuracy, as compared to the conventional ZUPT-aided INS using the SHOE detector with a low threshold and a high threshold.

5.3.2 Different Terrains

The second series of experiments investigated the navigation performance of the developed ZUPT-aided INS using FIBA covariance when operating on different terrains. The exper-

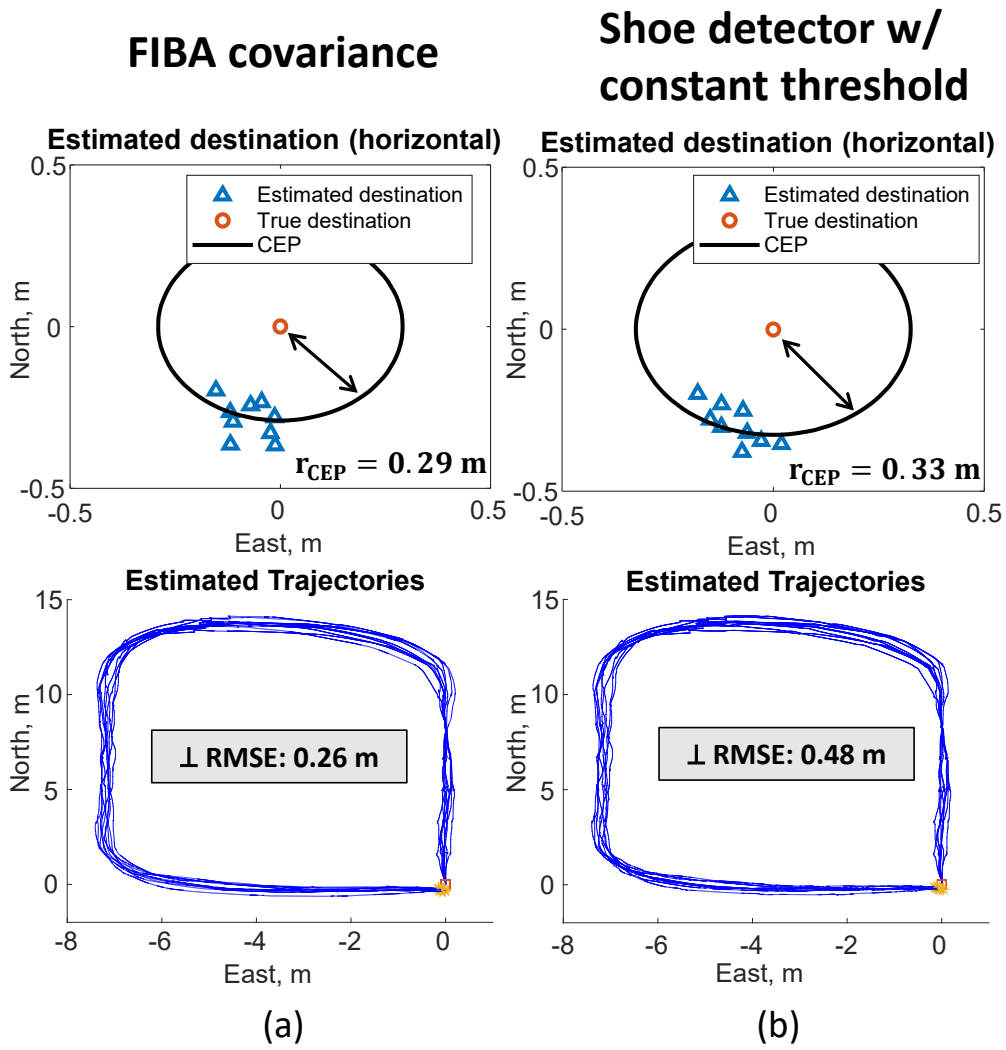


Figure 5.10: (a) Trajectories and destinations estimated by the ZUPT-aided INS using the developed FIBA covariance in the pedestrian navigation experiments described in Section 5.3.2. The hyper-parameters used for the FIBA covariance are summarized in Table 5.1. (b) Trajectories and destinations estimated by the ZUPT-aided INS using the SHOE detector with the threshold in the same experiments. The value of the threshold is indicated by the green line in Figure 5.3(b). In this series of experiments, the system using the FIBA covariance outperformed the conventional ZUPT-aided INS by 12% in terms of horizontal CEP and 45% in terms of vertical RMSE.

iments were conducted in an indoor environment illustrated in Figure 5.9. In this series of experiments, a pedestrian walked a closed-loop trajectory for a length of 50 m in 40 s. The path included a flat surface, a ramp, and stairs. The pedestrian repeated the same experiments 10 times.

We compared the navigation performance of the developed ZUPT-aided INS using the FIBA covariance and the conventional ZUPT-aided INS using the SHOE detector with a constant threshold. The experimental results of the two systems are shown in Figure 5.10. The solution with FIBA covariance improved horizontal CEP and the vertical RMSE by 12% and 45%, respectively, as compared to the conventional ZUPT-aided INS. Table 5.4 summarizes percentage of position errors in trajectory lengths in this series of experiments. We concluded the improvement in the navigation accuracy to be a direct result of using the FIBA covariance, which reduced the modeling error in the ZUPT-aided INS. This series of experiments demonstrated that the navigation performance of the ZUPT-aided INS using the developed FIBA covariance outperformed the case of conventional ZUPT-aided INS when traveling on terrains of flat planes, stairs, and slopes.

Table 5.4: Percentage of position error in trajectory length for the 2nd series of experiments

	FIBA covariance	**Constant threshold**
Horizontal	0.58%	0.66%
Vertical	0.52%	0.96%

5.4 Conclusion

This chapter introduced the FIBA covariance to dynamically adjust uncertainties of the zero-velocity measurements in the ZUPT-aided INS, allowing to reduce the modeling error of the ZUPT algorithm. The developed ZUPT-aided INS with the FIBA covariance was implemented in the EKF framework, where the measurements covariance matrix for the

zero-velocity measurements was updated in each iteration according to the FIBA covariance, which varied based on the instability metrics derived from foot-mounted IMU measurements. The developed FIBA covariance was demonstrated to exhibit a property that it gives a value sufficiently lower when the foot was stable, such that the zero-velocity measurements can effectively reset the velocity states. When the foot was experiencing a motion, the statistics of the FIBA covariance increased sharply, and as such the zero-velocity measurements would not have a significant numerical impact on the velocity state. This property of the FIBA covariance allows to reduce the ZUPT modeling error and eliminates the requirement to have a stance phase detector in the ZUPT-aided INS. Two series of indoor pedestrian navigation experiments were conducted in this section to evaluate the ZUPT-aided INS using the developed FIBA covariance. In the first series, including both walking and running activities, the solution using the FIBA covariance showed a maximum improvement in navigation accuracy of 36% horizontally and 64% vertically, as compared to the conventional ZUPT-aided INS using the SHOE detector with a constant threshold. In the second series of experiments, which included walking on different terrains of flat planes, stairs, and slopes, the ZUPT-aided INS using the FIBA covariance reduced horizontal CEP by 12% and vertical RMSE by 45%, as compared to the conventional ZUPT-aided INS. We concluded that using the FIBA covariance in the ZUPT-aided INS can eliminate the need to use a stance phase detector and could be beneficial for navigation accuracy. The result of this chapter has been published in [86]. Future research directions based on the developed FIBA covariance mechanism are suggested in Section 10.2 of Chapter 10.

Chapter 6

On Estimation Filter – Increasing Yaw Angle Observability

6.1 Introduction

This chapter presents the development of an augmentation approach to enhance the observability of a ZUPT-aided INS implemented in an EKF framework. The section is organized as follows. Section 6.2 presents the developed approach, Section 6.3 evaluates the navigation performance of the developed approach with numerical simulation and real-world indoor pedestrian navigation experiments, Section 6.4 concludes this chapter with a highlight of the results.

6.2 ZUPT-aided INS Augmented by Self-contained Vision-based Foot-to-foot Measurements

This section presents a vision-aided pedestrian inertial navigation system that includes two sets of shoe-mounted IMUs, cameras, and feature patterns. The system uses shoe-mounted feature points to estimate the relative positions between the shoes and compensates the position and yaw angle drifts in a ZUPT-aided INS. The number of feature points to be used in the system is a fixed quantity. As a result, the computational complexity is constant in any context. Our developed system uses foot-mounted IMUs and implements the ZUPT to limit the error growth in velocity estimation from IMU measurements between each camera frame. Additionally, the system does not use historical measurements for compensation. Thus, it is possible to achieve a real-time implementation.

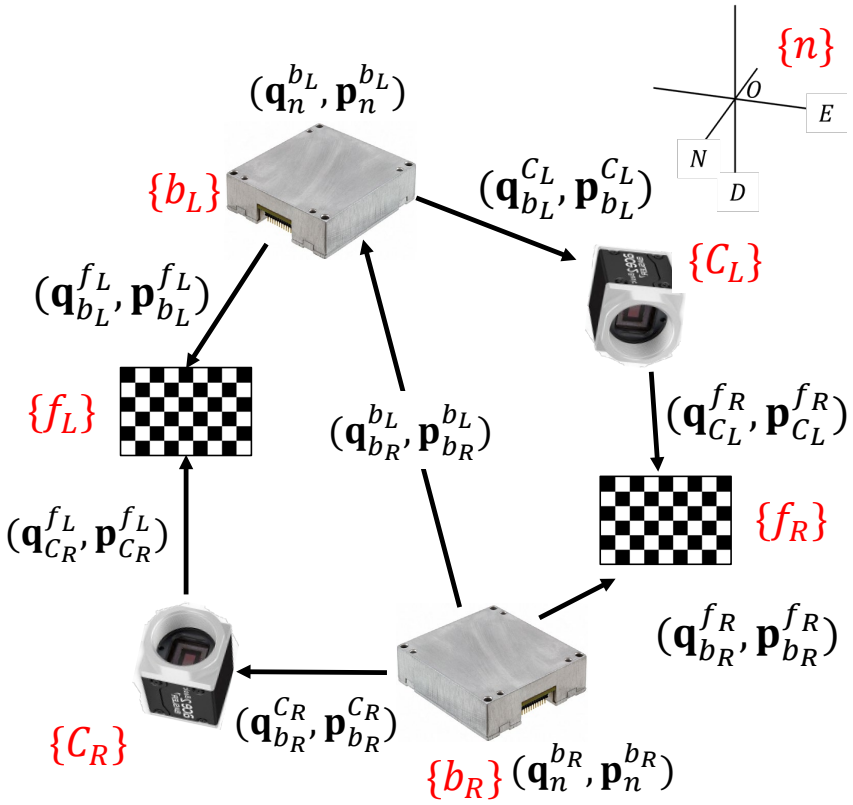


Figure 6.1: Relationship between the coordinate frames of different objects in the developed system.

6.2.1 Foot-to-foot Kinematics

This section derives a measurement model that uses self-contained vision measurements and ZUPT algorithm, provides a mechanism to simulate the foot-to-foot relative position measurements obtained by shoe-mounted cameras, and verifies the developed system with real-world experiments by comparing the results with a standalone ZUPT-aided INS and the ZUPT-aided INS augmented by foot-to-foot relative distance measurements.

The developed system aims to simultaneously track positions and orientations of the two shoe-mounted IMUs $\{b_L\}$ and $\{b_R\}$ in the navigation frame $\{n\}$ in an EKF. Two cameras $\{C_R\}$ and $\{C_L\}$ and two feature patterns $\{f_L\}$ are used to extract measurements for the update step of the EKF. The relationships between all the coordinate frame used in this section are shown in Figure 6.1. Earth rotation effect on the navigation frame is also included in our model. The system has two steps. The first step estimates the position and the orientation of the two IMUs based on the strap-down inertial navigation algorithm [198]. In the second step, the system estimates the pose of the IMUs based on images captured by the cameras and determines the status of each foot with a ZUPT detector. Then, these two measurements are fed to the EKF update step.

The setup of the system includes two IMUs, two feature patterns, and two cameras. All these components are mounted on the heel side of both shoes, shown in Figure 6.2. Returning to Figure 6.1, the positions and orientations of the two cameras in the two IMU frames are expressed by two vectors $(\mathbf{q}_{b_L}^{C_L}, \mathbf{p}_{b_L}^{C_L})$ and $(\mathbf{q}_{b_R}^{C_R}, \mathbf{p}_{b_R}^{C_R})$. The positions and orientations of the feature patterns in the two IMU frames are expressed by $(\mathbf{q}_{b_L}^{f_L}, \mathbf{p}_{b_L}^{f_L})$ and $(\mathbf{q}_{b_R}^{f_R}, \mathbf{p}_{b_R}^{f_R})$. $(\mathbf{q}_{C_R}^{f_L}, \mathbf{p}_{C_R}^{f_L})$ is the position of the feature pattern from the left shoe in the right camera frame and $(\mathbf{q}_{C_L}^{f_R}, \mathbf{p}_{C_L}^{f_R})$ is the position of the feature pattern from the right shoe in the left camera frame.

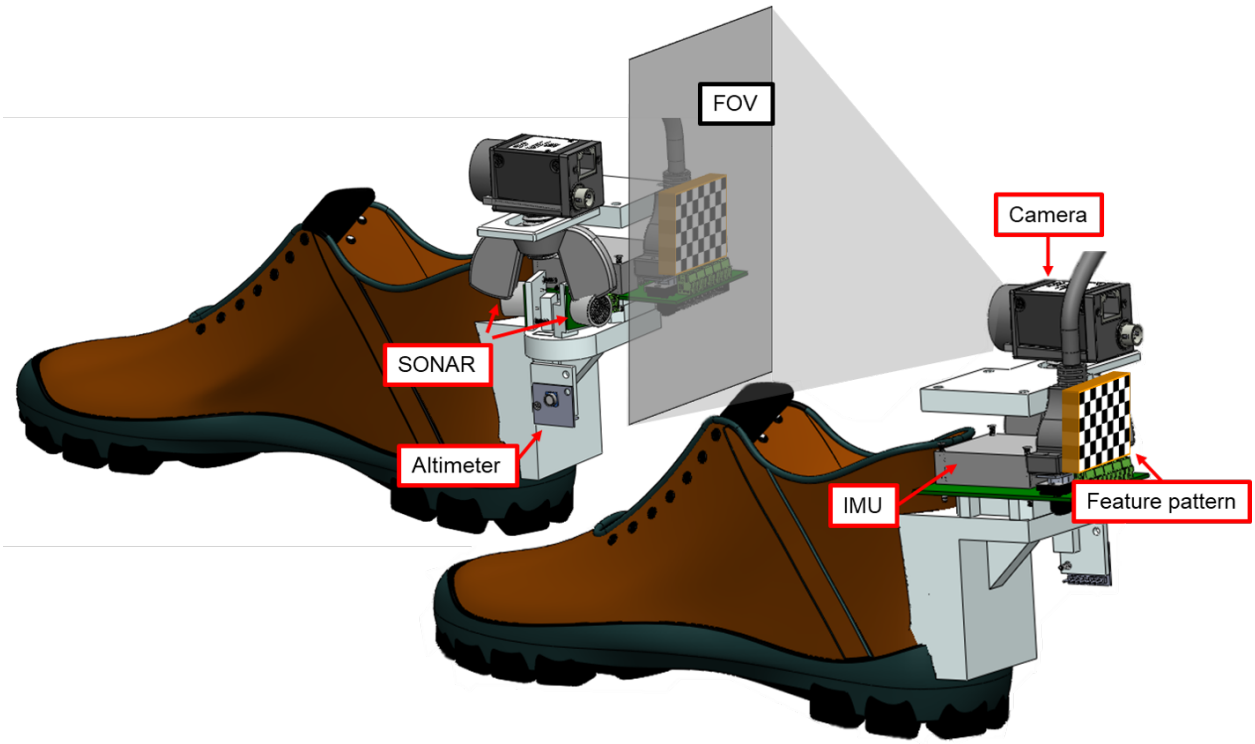


Figure 6.2: Lab-on-Shoe system for investigation of self-contained navigation.

6.2.2 Structure of the EKF States

The system keeps track of the states of the two shoe-mounted IMUs with an EKF. The EKF state is a 30×1 vector, described as follows:

$$\mathbf{x} = [\mathbf{q}_n^{b_L}, \mathbf{v}_n^{b_L}, \mathbf{p}_n^{b_L}, \mathbf{b}_a^{b_L}, \mathbf{b}_g^{b_L}, \mathbf{q}_n^{b_R}, \mathbf{v}_n^{b_R}, \mathbf{p}_n^{b_R}, \mathbf{b}_a^{b_R}, \mathbf{b}_g^{b_R}],$$

where $\mathbf{q}_n^{b_L}, \mathbf{v}_n^{b_L}, \mathbf{p}_n^{b_L}$ are the attitudes, velocities, and positions of the IMU mounted on the left shoe expressed in navigation frame, respectively. $\mathbf{b}_a^{b_L}$ and $\mathbf{b}_g^{b_L}$ are the bias of the accelerometers and gyroscopes of the left IMU. $\mathbf{q}_n^{b_R}, \mathbf{v}_n^{b_R}, \mathbf{p}_n^{b_R}, \mathbf{b}_a^{b_R}$, and $\mathbf{b}_g^{b_R}$ indicate the attitudes, velocities, positions, and the bias of the IMU mounted on the right shoe. The error state that are used in the EKF update step is expressed as

$$\boldsymbol{\delta}\mathbf{x} = [\boldsymbol{\delta\theta}_n^{b_L}, \boldsymbol{\delta}\mathbf{v}_n^{b_L}, \boldsymbol{\delta}\mathbf{p}_n^{b_L}, \boldsymbol{\delta}\mathbf{b}_a^{b_L}, \boldsymbol{\delta}\mathbf{b}_g^{b_L}, \boldsymbol{\delta\theta}_n^{b_R}, \boldsymbol{\delta}\mathbf{v}_n^{b_R}, \boldsymbol{\delta}\mathbf{p}_n^{b_R}, \boldsymbol{\delta}\mathbf{b}_a^{b_R}, \boldsymbol{\delta}\mathbf{b}_g^{b_R}],$$

Note that the attitude states are expressed in terms of the Euler angle (roll, pitch, yaw) in the error state because the true attitudes and the estimated attitudes are assumed to only differ by a small amount. Therefore, according to [136], the error quaternions $\boldsymbol{\delta}q$ can be approximated by

$$\boldsymbol{\delta}\mathbf{q} = [\frac{1}{2}\boldsymbol{\delta}\boldsymbol{\theta}^\top, 1]^\top.$$

6.2.3 Prediction Model: Strapdown Inertial Navigation using Dual IMUs

In the prediction step of the EKF, the states of each IMU are propagated according to the standard Strapdown Inertial Navigation [198] and the motions of the two feet are considered to be independent of each other. The linearized continuous-time model of the system state is expressed as follows:

$$\dot{\mathbf{x}} \triangleq \mathbf{A}\mathbf{x} + \mathbf{B} = \mathbf{A} = \begin{bmatrix} \mathbf{A}_{\mathrm{L}} & \mathbf{0}_{15\times15} \\ \mathbf{0}_{15\times15} & \mathbf{A}_{\mathrm{R}} \end{bmatrix} \mathbf{x} + \mathbf{B} = \begin{bmatrix} \mathbf{B}_{\mathrm{L}} \\ \mathbf{B}_{\mathrm{R}} \end{bmatrix},$$

where

$$\mathbf{A}_L(t) = \begin{bmatrix} \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & -\mathbf{C}(\mathbf{q}_n^{b_L}) & \mathbf{0}_{3\times3} \\ \overrightarrow{f_L}^n\times & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{C}(\mathbf{q}_n^{b_L}) \\ \mathbf{0}_{3\times3} & \mathbf{I}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \end{bmatrix},$$

$$\mathbf{A}_R(t) = \begin{bmatrix} \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & -\mathbf{C}(\mathbf{q}_n^{b_R}) & \mathbf{0}_{3\times3} \\ \overrightarrow{f}_R^{\,n}\times & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{C}(\mathbf{q}_n^{b_R}) \\ \mathbf{0}_{3\times3} & \mathbf{I}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \end{bmatrix},$$

$$\mathbf{B}_L = \begin{bmatrix} \mathbf{C}(\mathbf{q}_n^{b_L})\boldsymbol{\sigma}_{\text{ARW}} \\ \mathbf{C}(\mathbf{q}_n^{b_L})\boldsymbol{\sigma}_{\text{VRW}} \\ \mathbf{0}_{3\times1} \\ \boldsymbol{\sigma}_{\text{RRW}} \\ \boldsymbol{\sigma}_{\text{AcRW}} \end{bmatrix}, \mathbf{B}_R = \begin{bmatrix} \mathbf{C}(\mathbf{q}_n^{b_R})\boldsymbol{\sigma}_{\text{ARW}} \\ \mathbf{C}(\mathbf{q}_n^{b_R})\boldsymbol{\sigma}_{\text{VRW}} \\ \mathbf{0}_{3\times3} \\ \boldsymbol{\sigma}_{\text{RRW}} \\ \boldsymbol{\sigma}_{\text{AcRW}} \end{bmatrix},$$

where $\overrightarrow{f}_L^{\,n}\times$ and $\overrightarrow{f}_R^{\,n}\times$ are the skew-symmetric cross-product-operator of the accelerometer outputs of the left IMU and the right IMU, expressed in the navigation frame, respectively, $\boldsymbol{\sigma}_{\text{ARW}}$ is ARW of the gyroscopes, $\boldsymbol{\sigma}_{\text{VRW}}$ is the VRW of the accelerometers, $\boldsymbol{\sigma}_{\text{RRW}}$ is RRW of the gyroscope, and $\boldsymbol{\sigma}_{\text{AcRW}}$ is the AcRW of the accelerometers. $\mathbf{C}(\mathbf{q})$ is the DCM corresponding to the quaternion $\mathbf{q}$.

6.2.4 Measurement Model

The model consists of measurements from the ZUPT and the relative position between the two shoes obtained by the camera capturing the feature pattern mounted on the other shoe.

Zero-velocity Update

A stance phase detector is used to determine if a zero velocity measurement should be fed to the system. Different mechanisms to build a stance phase detector have been explored,

including using IMU with a pre-determined threshold, IMU and additional sensors, or IMU with adaptive threshold [223]. In this section, a stance phase is detected if

$$T(z_n) = \frac{1}{N} \sum_{k \in \Omega_n} (\frac{1}{\sigma_\alpha^2} \parallel y_k^\alpha - \bar{y}_n^\alpha \parallel^2 + \frac{1}{\sigma_\omega^2} \parallel y_k^\omega - \bar{y}_n^\omega \parallel^2) < \gamma, \tag{6.1}$$

where $\Omega_n = l \in \mathbb{N}, n \leq l \leq N-1$ is a collection of the IMU measurement indexes at time n with a window of length N, $z_n = \{[y_k^{\alpha\top}, y_k^{\omega\top}]\}_{k=N}^{k=N-1}$ is a sequence of the IMU measurements in the window, y_k^α are the accelerometer measurements at k, σ_a^2 is the noise variance of the accelerometer, y_k^ω are the gyroscope measurements at k, σ_a^2 is the noise variance of the accelerometer, σ_ω^2 is the noise variance of the gyroscope, and γ are user-defined thresholds.

Foot-to-Foot Relative Position Measurement

Images containing the feature pattern are used to estimate the pose of the camera mounted on the opposite shoe. This method considers a valid measurement when the feature pattern is fully present inside the FOV of the camera. For each frame taken by the camera, the position of the feature pattern relative to the camera can be estimated based on the features detected by computer vision-based methods and deduce the relative position between the camera and the feature pattern $(\mathbf{q}_{C_R}^{f_L}, \mathbf{p}_{C_R}^{f_L})$ and $(\mathbf{q}_{C_L}^{f_R}, \mathbf{p}_{C_L}^{f_R})$.

The positions of the two IMUs, the cameras, and the feature patterns have a relation shown in Figure 6.1. The foot-to-foot relative position measurements, which is the difference in positions of the left IMU and the right IMU expressed in the navigation frame $\mathbf{p}_n^{b_L} - \mathbf{p}_n^{b_R}$, are derived as follows.

When the left feature pattern is presented in the FOV of the right camera, the position of

the left IMU in the navigation frame can be rewritten as

$$\mathbf{p}_n^{b_L} = \mathbf{C}(\mathbf{q}_n^{b_R}) p_{b_R}^{b_L} + \mathbf{p}_n^{b_R} = \mathbf{C}(\mathbf{q}_n^{b_R})\{\mathbf{p}_{b_R}^{C_R} + \mathbf{C}(\mathbf{q}_{b_R}^{C_R})[\mathbf{p}_{C_R}^{f_L} - \mathbf{C}(\mathbf{q}_{C_R}^{f_L}) C^{\top}(\mathbf{q}_{b_L}^{f_L}) \mathbf{p}_{b_L}^{f_L}]\} + \mathbf{p}_n^{b_R}$$

Similarly, when the right feature pattern is presented in the FOV of the left camera, the position of the right IMU in the navigation frame can be expressed as

$$\mathbf{p}_n^{b_R} = \mathbf{C}(\mathbf{q}_n^{b_L}) p_{b_L}^{b_R} + \mathbf{p}_n^{b_L} = \mathbf{C}(\mathbf{q}_n^{b_L})\{\mathbf{p}_{b_L}^{C_L} + \mathbf{C}(\mathbf{q}_{b_L}^{C_L})[\mathbf{p}_{C_L}^{f_R} - \mathbf{C}(\mathbf{q}_{C_L}^{f_R}) C^{\top}(\mathbf{q}_{b_R}^{f_R}) \mathbf{p}_{b_R}^{f_R}]\} + \mathbf{p}_n^{b_L}$$

Thus, the foot-to-foot relative position in the navigation frame is described as

$$\begin{aligned}
\mathbf{p}_n^{b_L} - \mathbf{p}_n^{b_R} &= \mathbf{C}(\mathbf{q}_n^{b_R})\{\mathbf{p}_{b_R}^{C_R} + \mathbf{C}(\mathbf{q}_{b_R}^{C_R})[\mathbf{p}_{C_R}^{f_L} - \mathbf{C}(\mathbf{q}_{C_R}^{f_L}) C^{\top}(\mathbf{q}_{b_L}^{f_L}) \mathbf{p}_{b_L}^{f_L}]\} \\
&= -\mathbf{C}(\mathbf{q}_n^{b_L})\{\mathbf{p}_{b_L}^{C_L} + \mathbf{C}(\mathbf{q}_{b_L}^{C_L})[\mathbf{p}_{C_L}^{f_R} - \mathbf{C}(\mathbf{q}_{C_L}^{f_R}) C^{\top}(\mathbf{q}_{b_R}^{f_R}) \mathbf{p}_{b_R}^{f_R}]\}
\end{aligned}$$

Note that the above relation includes two equations. The first equation is used when the left feature pattern is present in the FOV of the right camera, and the second equation is used when the right feature pattern is present in the FOV of the left camera.

The measurement model for the relative position measurements and the ZUPT detector for both IMUs are described as follows:

$$\mathbf{z}_{\text{F2F}} = \mathbf{p}_n^{b_L} - \mathbf{p}_n^{b_R}, \mathbf{z}_{\text{ZUPT}_L} = \mathbf{v}_n^{b_L}, \mathbf{z}_{\text{ZUPT}_R} = \mathbf{v}_n^{b_R},$$

and the corresponding measurement matrices are

$$\begin{aligned}
\mathbf{H}_{\text{F2F}} &= \begin{bmatrix} \mathbf{0}_{3\times6} & \mathbf{I}_{3\times3} & \mathbf{0}_{3\times6} & \mathbf{0}_{3\times6} & -\mathbf{I}_{3\times3} & \mathbf{0}_{3\times6} \end{bmatrix}, \\
\mathbf{H}_{\text{ZUPT}_L} &= \begin{bmatrix} \mathbf{0}_{3\times3} & \mathbf{I}_{3\times3} & \mathbf{0}_{3\times9} & \mathbf{0}_{3\times15} \end{bmatrix}, \\
\mathbf{H}_{\text{ZUPT}_R} &= \begin{bmatrix} \mathbf{0}_{3\times15} & \mathbf{0}_{3\times3} & \mathbf{I}_{3\times3} & \mathbf{0}_{3\times9} \end{bmatrix}.
\end{aligned}$$

The form of the EKF measurement model employed depends on what measurements are available. When the measurements are available, they are stacked in one measurement vector $\mathbf{z}$ to form a single batch-form update equation. Similarly, the batch measurement matrix is formed by stacking the measurement matrices corresponding to the available measurements. For example, in the case where relative position and ZUPT of both feet have measurements, the measurement model is

$$\mathbf{z} = \begin{bmatrix} \mathbf{p}_n^{b_L} - \mathbf{p}_n^{b_L} \\ \mathbf{v}_n^{b_L} \\ \mathbf{v}_n^{b_R} \end{bmatrix}, \mathbf{H} = \begin{bmatrix} \mathbf{H}_{\text{F2F}} \\ \mathbf{H}_{\text{ZUPT}_L} \\ \mathbf{H}_{\text{ZUPT}_R} \end{bmatrix}.$$

6.3 Simulation and Experimental Results

6.3.1 Simulation Results

To validate the developed visual-aided pedestrian INS, a series of numerical simulations was performed and the results were compared with those using standalone ZUPT and ZUPT aided by foot-to-foot relative distance. The foot-to-foot relative distance measurements were assumed to be obtained from shoe-mounted SONAR sensors. In the simulation setup, two hypothetical foot trajectories were generated based on a foot motion model of a pedestrian walking at regular speed straight toward the North for 100 steps, resulting in a total length of 154 m in 107 s. Four assumptions were made for the foot motion: 1) each of the steps for the two feet was utterly identical, 2) the left foot started first, 3) the initial separation distance between the two shoes were 20 cm, and 4) the foot velocities were zero during the entire stance phase. The simulation model also included mismatches of g-sensitivity in the IMUs, which leads to the effect that the trajectory of left shoe drifted towards the west, and the trajectory of the right shoe drifted towards the east. This phenomenon was observed in

the experiments reported in [222].

The nominal final locations of the two shoes in the navigation frame were $[154.3, 0, 0]^\top$ m for the left shoe and $[153.5, 0.2, 30]^\top$ m for the right shoe. These position measurements were used to produced simulated IMU readouts for both shoes based on the generated paths. The IMU noise characteristics were the same as those of Analog Device ADIS−16485 IMUs used in the experiments (see subsection 6.3.2). The relative position measurements were derived by converting the difference in positions of the two shoes in the navigation frame to the body frame of each IMU. The generated relative position measurements were considered valid only if the position was within the FOV region. This simulation assumed that the cameras have a FOV of 75° and a frame rate of 60 Hz. These two parameters directly affect the amount of valid relative position measurements. The adopted standard deviation of the measurements was 1 mm. For relative distance measurements, it was assumed that the SONARs have the same characteristics as those of Devantech SRF08 Ultrasonic Sensors.

We collected 30 sets of simulations and compared the estimated results from the developed visual-aided INS with those estimated by standalone ZUPT algorithm and by ZUPT aided by foot-to-foot relative distance. Figure 6.3 shows the results. The average accumulated errors $\bar{e}_L$ and $\bar{e}_R$ and the covariances along the east direction $\sigma_{x,L}$ and $\sigma_{x,R}$ and the north direction $\sigma_{y,L}$ and $\sigma_{y,R}$ resulting from different methods are summarized in Table 6.1. The developed system showed an average improvement in accumulated errors of more than 90% for both feet compared to the relative distance aided ZUPT method and standalone ZUPT method.

6.3.2 Experimental Results

To demonstrate the validity of our system in realistic situations, experiments were conducted with a flexible system integrated with cameras and feature patterns [18]. The performance

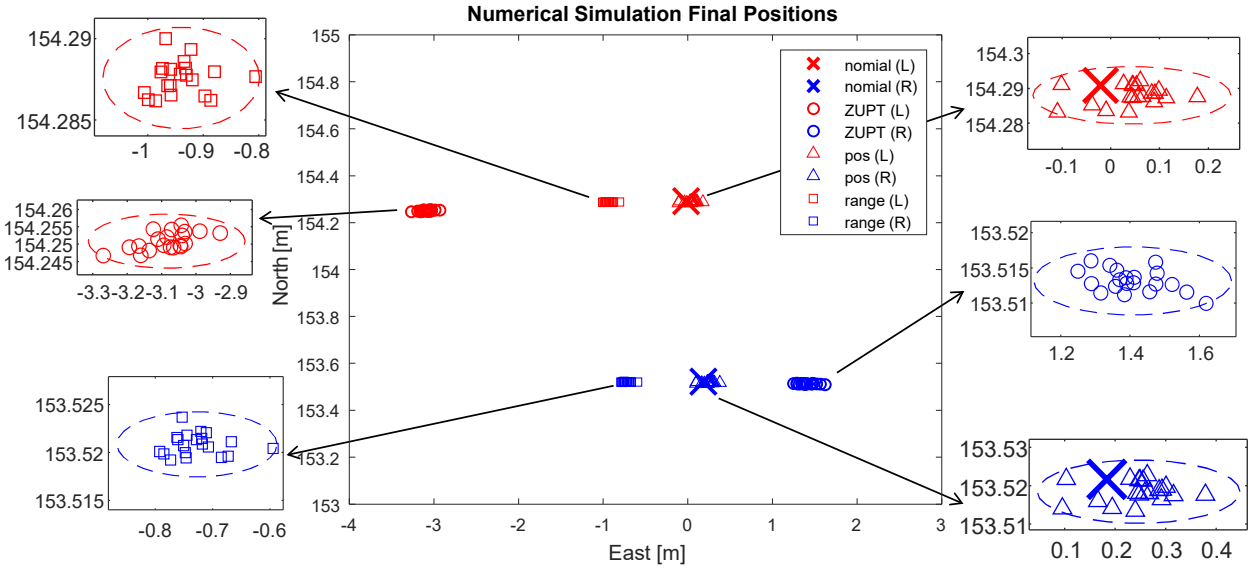


Figure 6.3: Simulated results correspond to standalone ZUPT (ZUPT), ZUPT aided by relative distance (range), and ZUPT aided by relative position (pos). Data in red were the estimated final positions of the left shoe, and those in blue were the estimated final position of the right shoe. Zoomed-in views of the data set corresponding to each of the methods are shown next to the data set. The dashed circle around each data set indicates the 3σ limit.

Table 6.1: Accumulated errors and covariances of the simulation dataset.

Unit [m]	ZUPT-aided INS Enhancement		
	Standalone	**Relative distance**	**Relative position**
$\bar{e}_L$	2.9687	0.8687	0.0653
$\bar{e}_R$	1.2354	0.8598	0.0646
$\sigma_{x,L}$	0.0962	0.0654	0.0817
$\sigma_{y,L}$	0.0027	0.0012	0.0014
$\sigma_{x,R}$	0.0841	0.0647	0.0809
$\sigma_{y,R}$	0.0016	0.0012	0.0014

of our system was evaluated against the standalone ZUPT method and the INS using both ZUPT and foot-to-foot relative distance measurements. The flexible system adopted Analog Device ADIS16485 IMUs and Devantech SRF08 SONARs. The cameras employed in this section were Gigabit Ethernet (GigE) camera acA800−200gc from the Basler Camera, and the lens of choice had a 4 mm focal length and a FOV of 73°. The cameras were calibrated by the standard camera calibration method provided by the *MATLAB Toolbox* [259, 77, 25]. The resolution of the images was 800×600 pixels. Images were recorded at a frame rate of 60 Hz while the IMU provided measurements at 120 Hz. The relative positions between cameras and IMUs, and between feature patterns and IMUs, were determined by Computer-Aided Design (CAD). The SONARs had a sampling rate of 25 Hz and a line-of-sight of 60°.

The feature pattern employed in the experiments was a 6×9 scaled-version checkerboard, which is often used in the standard camera calibration process. It should be pointed out that other reference geometries or features could also be used for detection. The physical size of each grid on the checkerboard was 5×5 mm. The features used for detection were the 40 intersubsection points on the checkerboard. The adopted feature detection method was described in [64]. For the 40 points detected on each frame, the camera extrinsic matrix was estimated with the method presented in [259], and from the extrinsic matrix, the relative positions between the two shoes were deduced. A slice of the sequence of images recorded during one of our indoor walking experiments is shown in Figure 6.4.

Two sets of experiments with different nominal trajectories were conducted on the second floor of the Engineering Gateway Building at the University of California, Irvine. In the first set of experiments, we conducted 5 runs of indoor experiments of walking straight toward the north for 53 meters. At the end of the experiment, the nominal distance between the two shoes was 30 cm and the nominal final locations were $[53, -0.15, 0]^\top$ m for the left shoe and $[53, 0.15, 0]^\top$ m for the right shoe. We compared this result to the case of using relative-distance-aided ZUPT algorithms and that of using standalone ZUPT, shown in Figure 6.5.

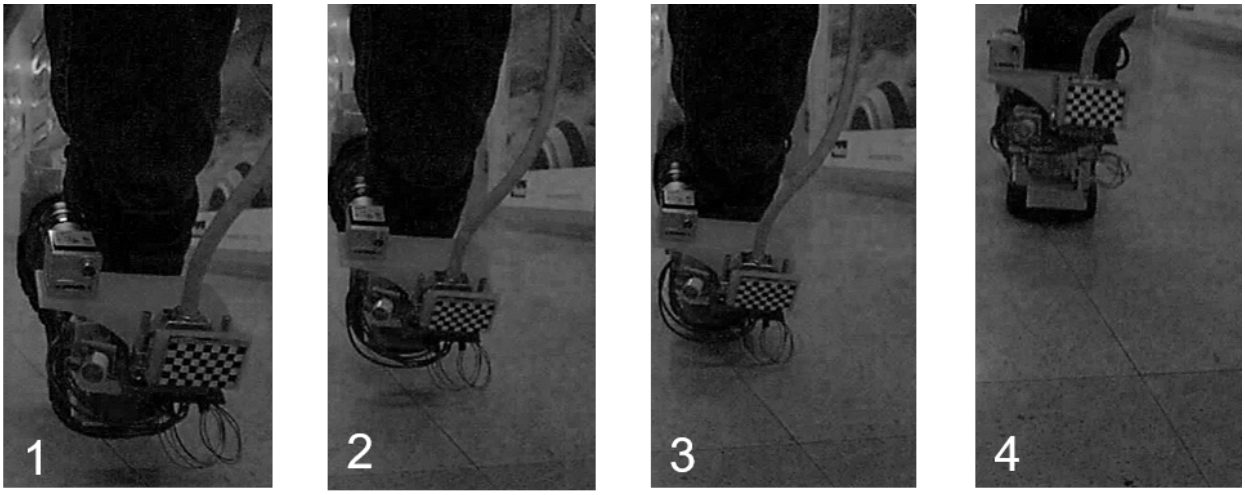


Figure 6.4: An example of consecutive images captured by the camera during a walking experiment.

The accumulated navigation errors $\bar{e}_L$ and $\bar{e}_R$ for each case are shown in Table 6.2. The developed system showed improvements in the accumulated navigation error of 55% and 22% for left and right foot when compared with standalone ZUPT; 30% and 31%, when compared with ZUPT aided by relative distance.

Table 6.2: Accumulated errors and covariances of the first set of the experiments

Unit [m]	**ZUPT-aided INS Enhancement**		
	Standalone	**Relative distance**	**Relative position**
$\bar{e}_L$	5.2113	3.3565	2.3589
$\bar{e}_R$	3.2046	3.6173	2.4852

Three things can be noted from the experiments. First, in the plot corresponding to the standalone ZUPT algorithm, we observed that the estimated trajectories of the left shoe drifted toward the west and those of the right shoe to the east. The main factor for this phenomenon was considered to be the mismatch of the g-sensitivity of the IMUs [222]. We did not calibrate the g-sensitivity of the IMUs in this section. Nevertheless, we observed that the trajectories estimated by our developed system had been shown to mitigate the errors caused by this phenomenon, while the method using foot-to-foot relative distance

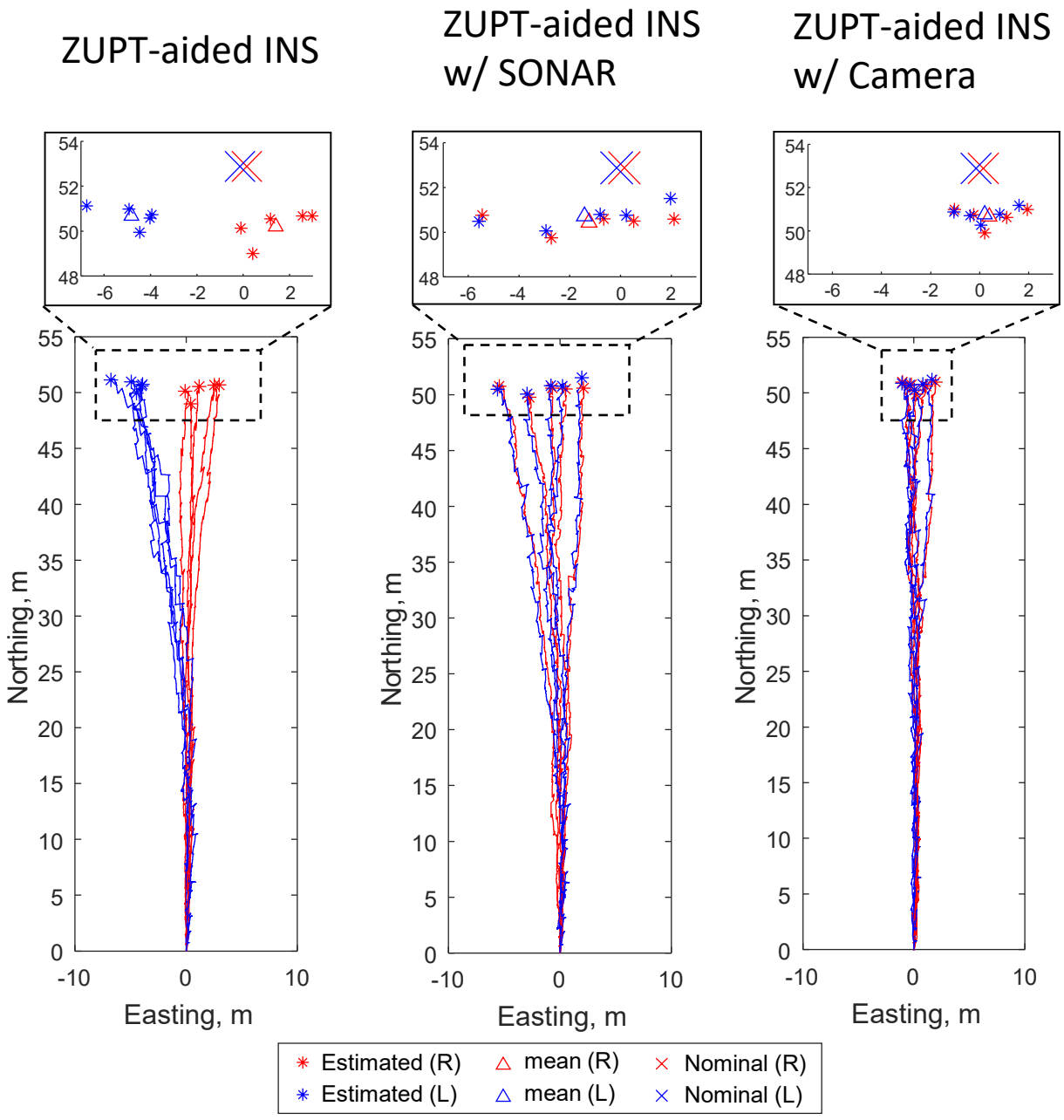


Figure 6.5: Estimated results of the first set of experiments from the standalone ZUPT method (ZUPT), ZUPT aided by relative distance measurements (ZUPT + Relative distance), and our developed system (ZUPT + Relative position). The lower plots show the estimated trajectories, and the upper plots present the corresponding final positions. The triangles in the upper plots indicate the statistical means of each data set.

measurements did not show much improvement. The observation can be explained by the fact that the measurement model of the relative distance in the EKF only optimizes the relative distance between the two shoes. Thus, the effect of such a method is to bring close together the trajectories resulted from the left and the right IMUs. This effect was also observed in Figure 6.3. Second, we perceived that all estimated trajectories by the three methods had lengths shorter than 53 m. This perception was mainly due to the systematic error in ZUPT, where the velocity of the foot was assumed zero during the stance phase in the gait cycle while in reality, the foot velocity was not absolutely zero during the stance phase. The false assumption that it is zero led to shorter estimated trajectories. Third, in the experiments, the amount of foot-to-foot relative position measurements obtained in each gait cycle was not consistent. The inconsistency was also another contribution to the difference between experiments and simulations.

In the second set of experiments, we conducted a close loop trajectory, which included more complicated walking motions of four right turns, a ramp, and a short stair. In this set of experiments, whenever we made a turn, we walked in an arched shape, instead of a direct 90° turn, so that the checkerboard could still be inside the FOV of the camera. The starting point of the left shoe overlapped with the ending point of the right shoe, and vice versa. The nominal total length of the trajectory was 126 m, and the navigation time was 140 s. The estimated results obtained by the three methods were compared and shown in Figure 6.6. The accumulated navigation errors e_L and e_R for each case are summarized in Table 6.3. The developed system showed improvements in the accumulated navigation error of 83% and 85% for each foot when compared to standalone ZUPT and 23% and 54%, when compared with ZUPT aided by relative distance.

We would like to point out that the lighting condition in the environment and the exposure time of the camera were key factors in implementation of the developed system. The lighting condition directly affected the performance of the feature detector. As a result, it is important

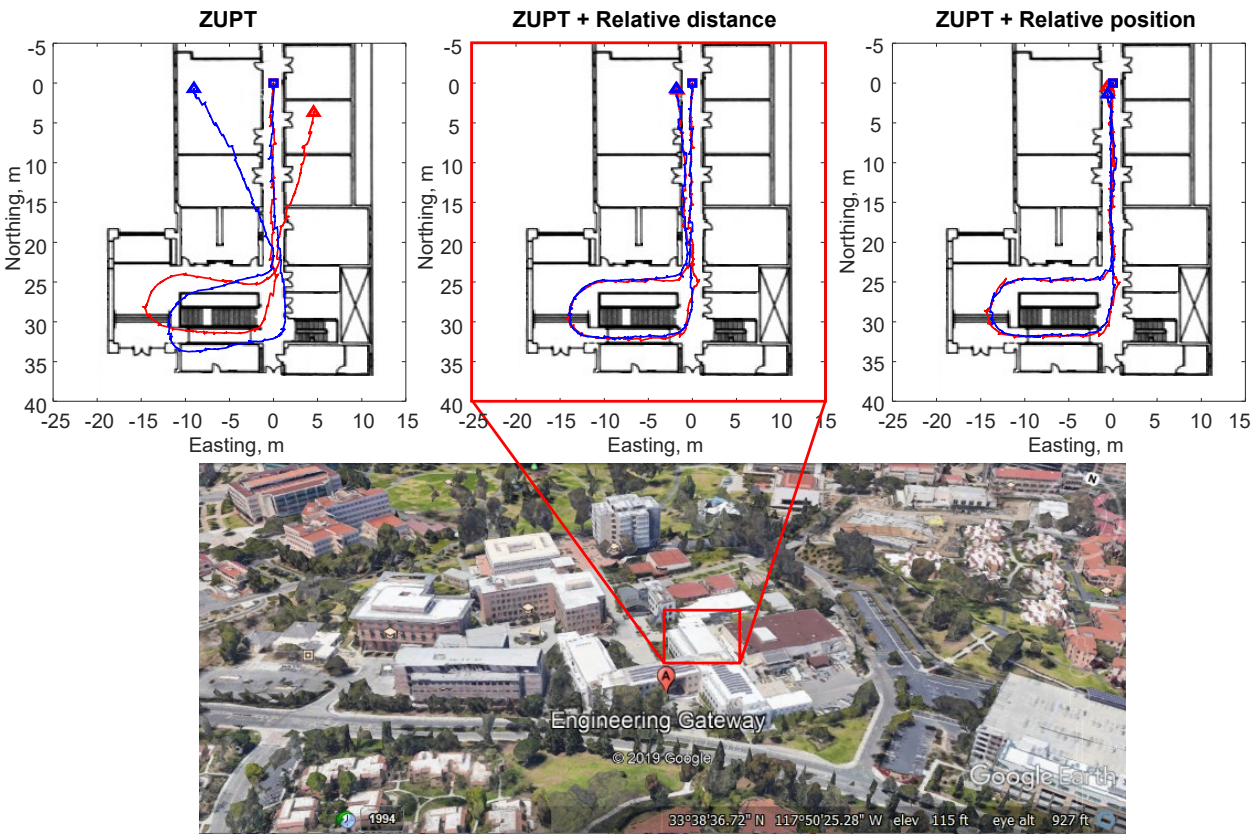


Figure 6.6: Estimated results of the second set of experiments from the standalone ZUPT method (ZUPT), ZUPT aided by relative distance measurements (ZUPT + Relative distance), and our developed system (ZUPT + Relative position).

Table 6.3: Accumulated errors and covariances of the second set of the experiments.

Unit [m]	**ZUPT-aided INS Enhancement**		
	Standalone	**Relative distance**	**Relative position**
e_L	9.0606	1.9836	1.5337
e_R	5.8524	1.9156	0.8743

to have sufficient light sources when the camera is used as a part of the navigator. The exposure time of the camera should also be set appropriately. If the exposure time set too short, the resulting images would not have enough brightness for the feature detection to work well. It cannot be set too long either; since the foot velocity can go up to as fast as 3 m/s, a long exposure time would lead to a quite blurry image. In our experiments, the value was set to 2000 μs.

6.4 Conclusion

In this chapter, a visual-aided Pedestrian INS using foot-to-foot relative position measurements was presented. The main contribution was the measurement model that blends ZUPT and foot-to-foot relative position measurements. The relative position measurements between the two shoes were obtained from shoe-mounted feature patterns and cameras. This measurement model directly outputs compensation measurements for the three position states and three velocity states, and does not need linearization. The developed system has constant computational complexity in any environment. The simulation results showed an improvement in accumulated navigation errors of over 90%. Experiments were also conducted, where the shoe-mounted feature pattern was rendered to a scaled version of a checkerboard. Experimental results showed a maximum improvement of 85% in accumulated errors, verifying the validity of the developed system in real-world environments. The results presented in this chapter were published in [94].

Chapter 7

On Estimation Filter – Compensating Vertical Position

7.1 Introduction

This chapter focuses on the development of approaches for enhancing vertical position accuracy of the ZUPT-aided INS. The rest of this section is organized as follows. Section 7.2 presents the EKF used to fuse altimeter measurements with the ZUPT-aided INS, Section 7.3 derives a close-form analytical expression that predicts the covariance of the estimated vertical displacement, Section 7.4 develops a hybrid ultrasonic/barometric altimeter that utilizes a downward-facing ultrasonic sensor to improve the reliability of a barometric altimeter, Section 7.5 demonstrates a real-time system architecture of the ZUPT-aided INS augmented with an altimeter, and Section 7.6 concludes the chapter.

7.2 ZUPT-aided INS Augmented With an Altimeter

This section presents the EKF used to realize the ZUPT-aided INS augmented with an altimeter.

7.2.1 EKF Prediction Step

A standard strap-down inertial navigation system in the navigation frame was implemented [198]. The output from the INS is corrected in the EKF by keeping track of the states $\boldsymbol{\delta}\mathbf{x}_k = [\boldsymbol{\delta}\boldsymbol{\theta}_k^\top, \boldsymbol{\delta}\mathbf{v}_k^\top, \boldsymbol{\delta}\mathbf{s}_k^\top]^\top$, where $\boldsymbol{\delta}\boldsymbol{\theta}_k$, $\boldsymbol{\delta}\mathbf{v}_k$, and $\boldsymbol{\delta}\mathbf{s}_k$ are the altitude, velocity, and position errors along the north, east, and down directions of the navigation coordinate frame at timestamp k [198].

The linearized continuous-time dynamic model of EKF is expressed as follows:

$$\boldsymbol{\delta}\dot{\mathbf{x}}_t = \begin{bmatrix} \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ \overrightarrow{f}^n\times & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{I}_{3\times3} & \mathbf{0}_{3\times3} \end{bmatrix} \boldsymbol{\delta}\mathbf{x}_t + \begin{bmatrix} \mathbf{C}_b^n\tilde{n}_{\text{ARW}} \\ \mathbf{C}_b^n\tilde{n}_{\text{VRW}} \\ \mathbf{0}_{3\times1} \end{bmatrix} \triangleq \mathbf{A}_t\boldsymbol{\delta}\mathbf{x}_t + \mathbf{B}_t,$$

where $\overrightarrow{f}^n\times$ is the skew-symmetric cross-product-operator of the accelerometer output in the navigation frame; $\tilde{n}_{\text{ARW}}$ is ARW of the gyroscopes; $\tilde{n}_{\text{VRW}}$ is the VRW of the accelerometers; $\mathbf{C}_b^n$ is the DCM from the body frame to the navigation frame;$\mathbf{0}_{n\times m}$ is a $n \times m$ zero matrix; and $\mathbf{I}_{m\times m}$ is a $m \times m$ identity matrix. $\mathbf{A}_t$ and $\mathbf{B}_t$ are time varying matrices.

This continuous-time model is discretized by the Euler's equation since the sampling period

Δt of the IMU is small, and the discrete-time dynamic matrix can be expressed as

$$\mathbf{F}_k = \exp(\mathbf{A}_{t_k}\Delta t) \approx \mathbf{I} + \mathbf{A}_{t_k}\Delta t,$$
$$\mathbf{B}_k = \mathbf{B}_{t_k}\Delta t,$$
$$\mathbf{Q}_k = \mathrm{diag}(\mathbf{B}_k)\Delta t.$$

The EKF prediction equations are

$$\boldsymbol{\delta}\hat{\mathbf{x}}_{k+1} = \mathbf{F}_k\boldsymbol{\delta}\mathbf{x}_k,$$
$$\hat{\mathbf{P}}_{k+1} = \mathbf{F}_k\mathbf{P}_k\mathbf{F}_k^\top + \mathbf{Q}_k,$$

where $\boldsymbol{\delta}\hat{\mathbf{x}}_{k+1}$ is the predicted states and $\hat{\mathbf{P}}_{k+1}$ is the *a priori* covariance matrix at the $(k+1)^{th}$ step.

7.2.2 EKF Update Step

On the updated step, both altimeter and pseudo-zero-velocity measurements were used to correct the INS output. The altimeter measurement updates the system states when a measurement is available, and the ZUPT measurements comes in when a stance phase is detected. Pressure-based altimeters measure the absolute air pressure, denoted as p. In an environment where temperature is static at 15 °C and gravity constant is 9.80665 m/s^2, this measurement can be converted to the estimated displacement along the down direction $d_\perp$ with respect to an initial location [253] based on the Earth's atmosphere model by the equation

$$d_\perp = 44330 \times \left(1 - \left(\frac{p}{p_0}\right)^{\frac{1}{5.255}}\right),$$

where p_0 is the absolute pressure at the sea level. The measurement matrix for altimeter $\mathbf{H}_{\text{ALT}}$ is expressed as follows:

$$\mathbf{H}_{\text{ALT}} = \begin{bmatrix} \mathbf{0}_{1\times3} & \mathbf{0}_{1\times3} & 0 & 0 & 1 \end{bmatrix}.$$

For the ZUPT algorithm, a stance phase is detected if

$$T(z_n) = \frac{1}{N}\sum_{k\in\Omega_n}(\frac{1}{\sigma_\alpha^2} \parallel y_k^\alpha - \bar{y}_n^\alpha \parallel^2 + \frac{1}{\sigma_\omega^2} \parallel y_k^\omega - \bar{y}_n^\omega \parallel^2) < \gamma, \tag{7.1}$$

where $\Omega_n = l \in \mathbb{N}, n \leq l \leq N-1$ is a collection of the IMU measurement indexes at time n with a window of length N, $z_n = \{[y_k^{\alpha\top}, y_k^{\omega\top}]\}_{k=N}^{k=N-1}$ is a sequence of the IMU measurements in the window, y_k^α are the accelerometer measurements at k, σ_a^2 is the noise variance of the accelerometer, y_k^ω are the gyroscope measurements at k, σ_a^2 is the noise variance of the accelerometer, σ_ω^2 is the noise variance of the gyroscope, and γ are user-defined thresholds. The measurement matrix $\mathbf{H}_{\text{ZUPT}}$ for ZUPT algorithm is expressed as follows:

$$\mathbf{H}_{\text{ZUPT}} = \begin{bmatrix} \mathbf{0}_{3\times3} & \mathbf{I}_{3\times3} & \mathbf{0}_{3\times3}. \end{bmatrix},$$

The measurement equation of the EKF has different forms for each of the following three cases: 1) A stance phase is detected (ZUPT ON) and the altimeter has a null measurement, 2) A swing phase is detected and the altimeter acquires a measurement (altimeter ON), and 3) A stance phase is detected and the altimeter obtains a measurement (Both ON). The altimeter can provide null measurement because in the case of this project, an IMU with 100Hz sampling rate and an altimeter with 10Hz sampling rate are used. The altimeter acquires an effective measurement at the first IMU sample. Then in the following 9 IMU samples, the altimeter gives null measurement. The measurement equation of the EKF is

expressed as follows:

$$z_k = \mathbf{H}\boldsymbol{\delta}\mathbf{x}_k + \tilde{\omega}_k,$$

where $\mathbf{H} = \mathbf{H}_{\text{ZUPT}}$ in the ZUPT ON case, $\mathbf{H} = \mathbf{H}_{\text{ALT}}$ in the altimeter ON case, and $\mathbf{H} = \left[\mathbf{H}_{\text{ZUPT}}^{\top}, \mathbf{H}_{\text{ALT}}^{\top}\right]^{\top}$ in the both ON case. $\tilde{\omega}_k$ is the measurement noise associated with altimeter and ZUPT measurements, modeled as Gaussian distribution, with variances σ^2_{ALT} and σ^2_{ZUPT}, respectively.

The EKF update equations are described as follows.

$$\mathbf{K} = \hat{\mathbf{P}}_{k+1}\mathbf{H}^{\top}(\mathbf{H}\hat{\mathbf{P}}_{k+1}\mathbf{H}^{\top} + \mathbf{R})^{-1},$$
$$\mathbf{v}_{k+1} = \mathbf{z}_k - \mathbf{H}\boldsymbol{\delta}\hat{\mathbf{x}}_{k+1},$$
$$\boldsymbol{\delta}\mathbf{x}_{k+1} = \boldsymbol{\delta}\hat{\mathbf{x}}_{k+1} + \mathbf{K}\mathbf{v}_{k+1},$$
$$\mathbf{P}_{k+1} = (\mathbf{I} - \mathbf{K}\mathbf{H})\hat{\mathbf{P}}_{k+1},$$

where $\mathbf{K}$ is the Kalman gain, $\mathbf{v}_{k+1}$ is the innovation sequence, $\mathbf{x}_{k+1}$ is the *a posteriori* state, and $\mathbf{P}_{k+1}$ is the covariance matrix at the $(k+1)^{th}$ step.

7.3 Analytically Predicting Vertical Displacement Error

Pedestrian INS implementing ZUPT-aided INS based on a foot-mounted IMU has been pointed out that vertical displacement drifts faster than the horizontal ones [143]. As discussed in Section 1.4.3, altimeters can provide vertical position compensation for the ZUPT-aided INS. When developing a ZUPT/altimeter-aided INS, there are a variety of types of altimeters, each of which has different noise density, resolution, measurement ranges, SWaP+C budget. An accurate analytical approach that estimates vertical displacement accuracy of an

ZUPT/Altimeter-aided INS can greatly lower the development cost and duration and allows for quickly determining a balance between SWaP+C and navigation accuracy.

The analytical estimations of velocity errors in the north, east, and down directions were previously derived in [216], and the position error standard deviations in the north and east directions were analytically estimated in [226] for a ZUPT-augmented INS. This section discusses a closed-form analytical estimation of the displacement error in the vertical direction in a ZUPT/altimeter-aided INS.

7.3.1 Estimation of Error Covariance in the Down Direction

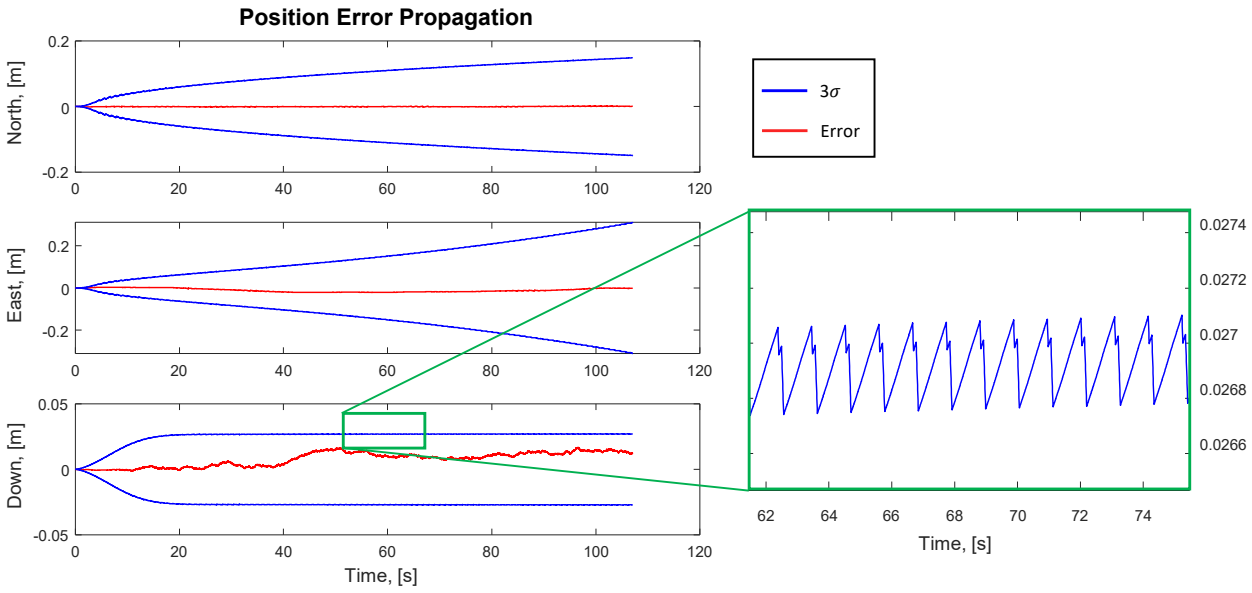


Figure 7.1: A typical propagation of errors in displacement estimations in the INS aided by ZUPT and altimeter. N, E, and D are the displacements along the north, east, and down directions, respectively. The red curve in each plot is the error profile, and the blue curve indicates the 3σ limit of errors.

The analytical estimation of the displacement error in the down direction was motivated by an observation that the altimeter measurements are able to reduce the navigation error by restricting the error growth of displacement along the down direction. A typical propagation of errors in displacements along the north, the east, and the down directions and their covari-

ances are presented in Figure 7.1. It could be observed that while covariances of the errors in displacement estimation of the north and the east directions keep growing over time, the covariance along the down direction reaches a stable level with a small range of fluctuation. The fluctuation of the covariance follows a pattern that is reduced when measurements from an altimeter are acquired, and it increases when the measurements from the altimeter are not available. This observation inspired us to combine the altimeter parameters that determine the altimeter performance and the IMU parameters that are dominated in the free navigation. The combination enables us to fully analyze the system behavior and extract the covariance of the errors in the system's state estimation.

Since there are 9 states in the EKF implementation, both the *a priori* and the *a posteriori* covariance matrices $\mathbf{P}_k$ and $\hat{\mathbf{P}}_k$ are 9×9. These two matrices are divided into nine 3×3 sub-matrices as follows:

$$\mathbf{P}_k = \begin{bmatrix} \mathbf{P}_{11} & \mathbf{P}_{12} & \mathbf{P}_{13} \\ \mathbf{P}_{21} & \mathbf{P}_{22} & \mathbf{P}_{23} \\ \mathbf{P}_{31} & \mathbf{P}_{32} & \mathbf{P}_{33} \end{bmatrix}, \hat{\mathbf{P}}_k = \begin{bmatrix} \hat{\mathbf{P}}_{11} & \hat{\mathbf{P}}_{12} & \hat{\mathbf{P}}_{13} \\ \hat{\mathbf{P}}_{21} & \hat{\mathbf{P}}_{22} & \hat{\mathbf{P}}_{23} \\ \hat{\mathbf{P}}_{31} & \hat{\mathbf{P}}_{32} & \hat{\mathbf{P}}_{33} \end{bmatrix}.$$

From the EKF propagation equations,

$$\begin{aligned} \hat{\mathbf{P}}_{22}(3,3) &= \mathbf{P}_{22}(3,3) + (2\mathbf{P}_{21}(3,2)a_{D,t} + \sigma^2_{\text{VRW}})\Delta t \\ &= \mathbf{P}_{22}(3,3) + 2\mathbf{P}_{21}(3,2)v_{D,t} + \sigma^2_{\text{VRW}}\Delta t, \end{aligned}$$

where $a_{D,t}$ and $v_{D,t}$ are the acceleration and the velocity estimates along the down direction in the navigation frame. Since the term with $\mathbf{P}_{21}(3,2) \ll \sigma^2_{\text{VRW}}$, an increase in $\mathbf{P}_{22}(3,3)$ during the prediction step, denoted as $\Delta_{\text{pred}}\mathbf{P}_{22}(3,3)$ can be approximated as

$$\Delta_{\text{pred}}\mathbf{P}_{22}(3,3) = \hat{\mathbf{P}}_{22}(3,3) - \mathbf{P}_{22}(3,3) = \sigma^2_{\text{VRW}}\Delta t$$

From the update step,

$$\mathbf{P}_{22}(3,3) = \hat{\mathbf{P}}_{22}(3,3) - \frac{\hat{\mathbf{P}}_{22}(3,3)^2}{\sigma^2_{\text{ZUPT}}}$$

Consequently, the decrease in $P_{22}(3,3)$ during the update step, denoted as $\Delta_{\text{update}}\mathbf{P}_{22}(3,3)$, is

$$\Delta_{\text{update}}\mathbf{P}_{22}(3,3) = -\frac{\hat{\mathbf{P}}_{22}(3,3)^2}{\sigma^2_{\text{ZUPT}}}$$

Because the ZUPT algorithm limits the error covariance growth in velocities, the increase $\Delta_{\text{pred}}\mathbf{P}_{22}(3,3)$ during the prediction step is equal to decrease $\Delta_{\text{update}}\mathbf{P}_{22}(3,3)$ in the update step,

$$\int_{t_{\text{stride}}} \Delta_{\text{pred}}\mathbf{P}_{22}(3,3)dt = -\int_{t_{\text{stance}}} \Delta_{\text{update}}\mathbf{P}_{22}(3,3)dt, \tag{7.2}$$

where t_{stride} is a time duration of the swing phase in a gait cycle, N_{stance} is a number of samples being updated by the ZUPT algorithm, and dt is a sampling rate of IMU. Rearranging (7.2), the covariance of the velocity along the down direction is

$$\mathbf{P}_{22}(3,3) = \sqrt{\frac{\sigma^2_{\text{VRW}} t_{\text{stride}} \sigma^2_{\text{ZUPT}}}{N_{\text{stance}}}}$$

To find $\mathbf{P}_{23}(3,3)$, the derivation starts from the propagation equation,

$$\hat{\mathbf{P}}_{23}(3,3) = \mathbf{P}_{23}(3,3) + (\mathbf{P}_{22}(3,3) + \alpha_{D,t}\mathbf{P}_{13}(2,3))\Delta t$$

Since the term with $\mathbf{P}_{13}(2,3)$ is much smaller than the term with $\mathbf{P}_{22}(3,3)$, the equation can

be rewritten and the increase of $\mathbf{P}_{23}(3,3)$ during the swing phase is approximated as

$$\Delta_{\text{pred}}\mathbf{P}_{23}(3,3) = \hat{\mathbf{P}}_{23}(3,3) - \mathbf{P}_{23}(3,3) = \mathbf{P}_{22}(3,3)\Delta t = \sqrt{\frac{\sigma^2_{\text{VRW}} t_{\text{stride}} \sigma^2_{\text{ZUPT}}}{N_{\text{stance}}}}\Delta t$$

After taking the integral in (7.2) from 0 to t, $\mathbf{P}_{23}(3,3)$ can be expressed as follows:

$$\mathbf{P}_{23}(3,3) = \sqrt{\frac{\sigma^2_{\text{VRW}} t_{\text{stride}} \sigma^2_{\text{ZUPT}}}{N_{\text{stance}}}}t$$

Now, it is ready to derive $P_{33}(3,3)$. From the propagation equation, the increase of displacement covariance along the down direction, when the altimeter measurement is not available, is

$$\Delta_{\text{pred}}\mathbf{P}_{33}(3,3) = \hat{\mathbf{P}}_{33}(3,3) - \mathbf{P}_{33}(3,3) \approx 2\mathbf{P}_{23}(3,3)\Delta t.$$

From the update equation discussed in Section 7.2.1, the decrease in covariance when altimeter data is obtained is

$$\Delta_{\text{update}}\mathbf{P}_{33}(3,3) = \mathbf{P}_{33}(3,3) - \hat{\mathbf{P}}_{33}(3,3) = -\frac{\mathbf{P}_{23}(3,3)^2}{\sigma^2_{\text{ZUPT}}} - \frac{\mathbf{P}_{33}(3,3)^2}{\sigma^2_{\text{ALT}}},$$

where σ_{ALT} is the altimeter measurement noise, which is considered as identical to the altimeter resolution in this case.

Since the displacement covariance along the down direction is bounded by altimeter measurements,

$$\int_{t_{\text{ALT OFF}}} \Delta_{\text{pred}}\mathbf{P}_{33}(3,3) = -\int_{t_{\text{ALT ON}}} \Delta_{\text{update}}\mathbf{P}_{33}(3,3), \tag{7.3}$$

where $t_{\text{ALT ON}}$ is the total amount of time that the altimeter is obtaining data within a gait cycle, and $t_{\text{ALT OFF}}$ is the rest of the time in the gait cycle. Taking the integration, the

estimate of the error covariance of displacement along the down direction is

$$\mathbf{P}_{33}(3,3) = (\frac{t_{\text{ALT OFF}}}{N_{\text{ALT ON}}}\sqrt{\frac{\sigma^2_{\text{VRW}} t_{\text{stride}} \sigma^2_{\text{ZUPT}}}{N_{\text{stance}}}} - \frac{\sigma^2_{\text{VRW}} t_{\text{stride}} N_{\text{stance}} t_{\text{ALT ON}}}{3N_{\text{ALT ON}}})^{\frac{1}{2}} \frac{\sigma_{\text{ALT}}}{\sqrt{\Delta t}}.$$

A few observations can be made from the analytical solution:

1. The variance of the vertical displacement estimate in the EKF is affected by σ_{VRW}, but is independent of σ_{ARW}.
2. Altimeter sampling rate and resolution are key factors in estimation of variance.
3. The ratio of the swing phase and the stance phase during the gait cycle would affect the estimate of variance.

7.3.2 Simulation and Experiment

Two sets of numerical simulations and experiments were conducted to verify the derived analytical expression. This section presents how the simulations and experiments were conducted and discuss their results.

Investigation of Altimeter Resolutions on Vertical Accuracy

In the first set of simulations and experiments, the effect of altimeter resolution on displacement error along the down direction was investigated. For simulation, a trajectory of foot toward north and the corresponding IMU and altimeter readouts were generated based on a human gait analysis [216]. The stride length was set to 0.6 [m], and each step was considered identical. VRW of the IMU was set to 0.023 mg/$\sqrt{\text{Hz}}$, and ARW to 0.3 °/$\sqrt{\text{h}}$. The altimeter sampling rate was set to 20 Hz. The total navigation time was 107 s. Then, dif-

ferent levels of altimeter noises, from 10^{-3} [m] to 10 [m], were added to the readouts. Next, the ZUPT-augmented inertial navigation algorithm aided by altimeter was applied to the IMU and altimeter readouts. A flexible laboratory testbed [18] was used to perform the experiments. Five additional pressure-based altimeters, MS−5803−01BA, MS−5803−02BA, MS−5803−05BA, MS−5803−14BA, and MS−5803−30BA were considered. Their resolutions were experimentally determined, which were found to be 0.29 [m], 0.6 [m], 1.5 [m], 5 [m], and 10 [m], respectively. For each altimeter, a set of 6 similar experiments was conducted. In each experiment, the total navigation time was 90 s.

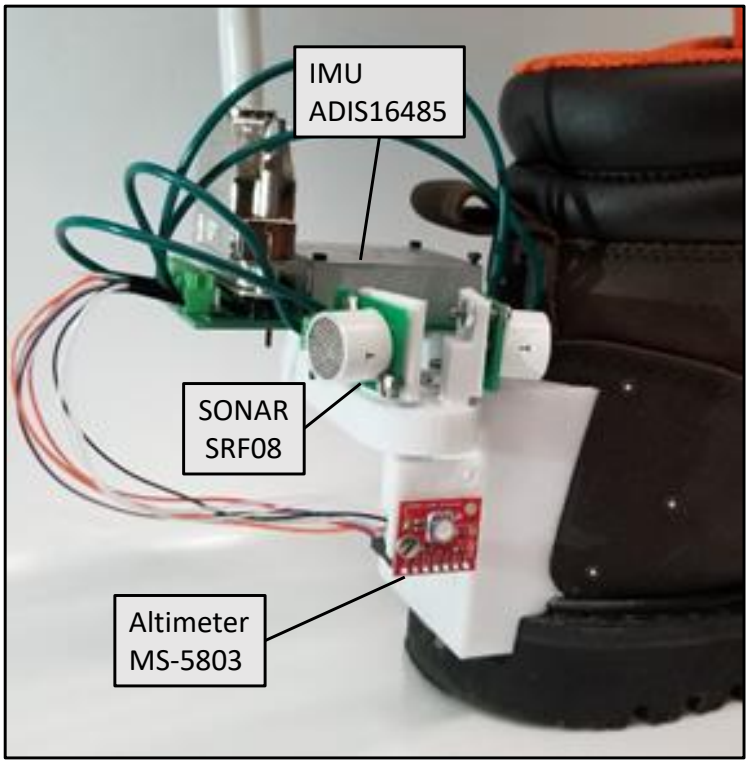


Figure 7.2: Illustrated is a test platform integrated with an MS−5803 altimeter.

Figure 7.3 shows a relation between altimeter resolution to the estimated position error's standard deviation along the down direction. The blue curve corresponds to the analytical estimation of the error displacement covariance; the red circles indicate the results of the simulation, and the blue solid triangles represent the statistical means of the results from the 6 experiments using the same altimeter. The results show that the analytical and the

simulation estimates were closely matched within 15% for altimeter resolutions higher than 0.05 [m]. This mismatch came mainly from omitting the terms that have relatively small values in the derivation of the analytical estimates. For the case of lower altimeter resolutions, the term with $\mathbf{P}_{13}(2,3)$ was no longer 10 times smaller than $\mathbf{P}_{33}(3,3)$, and a larger discrepancy percentage was expected. Figure 7.3 also shows a difference within 16% between the analytical estimation and experimental results. The discrepancy between the analytical estimation and experimental results was contributed not only by omission of the small-value terms but also by the following two factors: 1) foot dynamics considered in the derivation was not exactly the same as the actual walking experiments and 2) the altimeter measurements could be affected by abrupt changes in air pressure during the experiments.

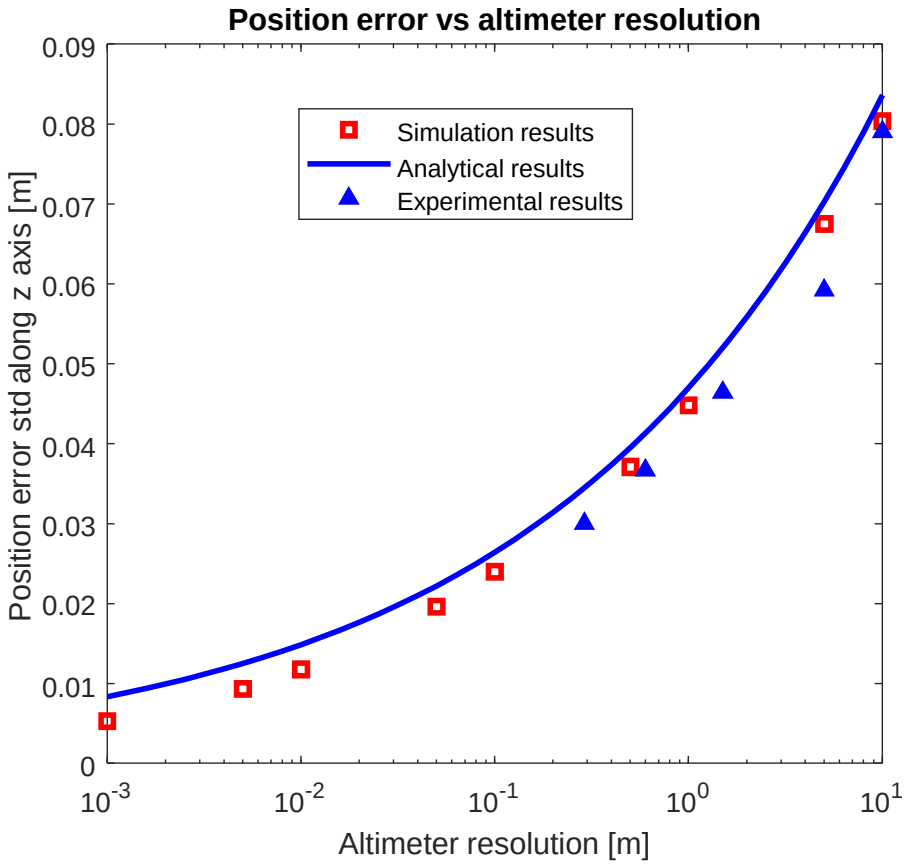


Figure 7.3: The relation of altimeter resolution and the displacement error standard deviation along the down direction.

Investigation of Altimeter Sampling Rate on Vertical Accuracy

In the second set of simulation and experiment, we investigated a relationship between the altimeter sampling rate and the displacement covariance. The simulation setup was the same as in the first set, except that in this case, the altimeter resolution was set to 0.1 [m] and its sampling rate was swept from 2 Hz to 100 Hz. In the experiment, the altimeter MS−5803−01BA was used with our custom navigation platform, and the sampling rate was swept from 1 Hz to 20 Hz. Nominal trajectories were the same as in the first experiment, and 6 experiments were conducted for each sampling rate.

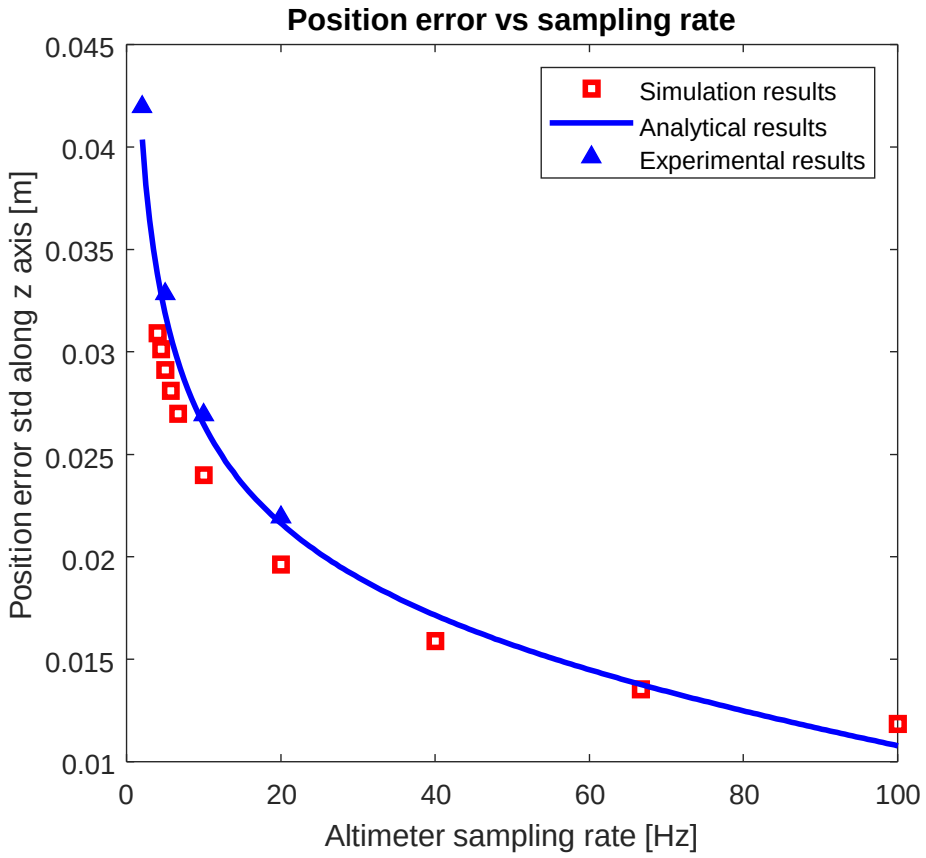


Figure 7.4: The relation of altimeter sampling rate and the displacement error standard deviation along the down direction.

Figure 7.4 shows the effect of altimeter sampling rate on the estimated position error standard deviation along the down direction. The blue curve corresponds to the analytical estimation

of the error displacement covariance; the red circles indicate the results of the simulation, and the blue solid triangles represent statistical means of the results from the 6 experiments using the same altimeter. The results show that the numerical simulation results differed from the analytical results by less than 20%. It could be seen that the discrepancy between them tended to increase as the altimeter sampling rate was increased. This was anticipated because an increase in the altimeter sampling rate led to a longer integration intervals for the second integral in (7.3), in which the term $\mathbf{P}_{21}(3,2)$ was neglected because of the assumption $\mathbf{P}_{21}(3,2) \ll \sigma^2_{\text{VRW}}$. The difference between experimental results and analytical results was within 5%. Note that only altimeter sampling rates lower than 20 Hz was shown due to the limitation of the sampling rate of altimeter MS−5308 in data acquisition communication protocol.

This section derived an analytical solution for position variance estimation in the case of inertial navigation aided by ZUPT and altimeter measurements. The solution showed that the displacement error variance along the down direction was directly related to the altimeter resolution, altimeter sampling rate, IMU VRW, IMU sampling rate, and the swing phase and the stance phase percentage of the gait cycle. The analytical expression was verified by numerical simulation and experimental, showing that the analytical estimation had an uncertainty of less than 20%. This section provides an analytical expression to estimate the displacement accuracy along the down direction of INS aided by ZUPT and altimeter measurements. The results discussed in this section were published in [91].

7.4 A Hybrid Barometric/Ultrasonic Altimeter

In this section, a hybrid altimeter that uses a shoe-mounted ultrasonic altimeter and a shoe-mounted barometer for aiding ZUPT-based INS was developed. One of the goals of this approach is to minimize the usage of barometers during indoor navigation, as barometers are

subject to variations in ambient temperature and air pressure. The shoe-mounted ultrasonic sensor was placed on a toe side of pedestrian's shoe, facing downward to the ground. The measurement is the height of the shoe relative to the ground. The KF is used to convert ultrasonic readouts to altitude data relative to the initial location and refer to this type of altimeter as the ultrasonic altimeter. The developed ultrasonic altimeter is experimentally demonstrated to estimate pedestrian's elevation when walking on a flat plane.

To extend the usage of the ultrasonic altimeter when operating on other terrains, a hybrid framework that fuses a barometer and an ultrasonic altimeter was developed. In the fusion process, the ultrasonic altimeter receives more weights when the hybrid system detects that the pedestrian is walking on a flat plane or stairs. For the case of slopes or inside elevators, the hybrid system would prioritize the barometer measurements. The detection of flat planes, slopes, and elevators would be achieved with information obtained instantaneously from an IMU. This configuration not only limits the error growth of an INS in the vertical direction but also enables capturing the foot motions, which is subsequently used to aid ZUPT-augmented INS.

7.4.1 Ultrasonic Altimeter

A shoe-mounted downward-facing ultrasonic sensor is capable of finding relative distances between the shoe and the ground. To use the sensor as an altimeter, we convert the relative distance to the height of the shoe in the navigation frame by simultaneously estimating the vertical position of the floor in the navigation frame. In the case of walking on flat surfaces and stairs, the estimation of floors can be achieved based on two phenomena: 1) ultrasonic measurements are smooth in the case of flat surfaces, and 2) a discontinuity in ultrasonic measurements can be observed when the sensor is passing through edges of stairs. These phenomena were observed in indoor walking experiments with the Lab-On-Shoe platform

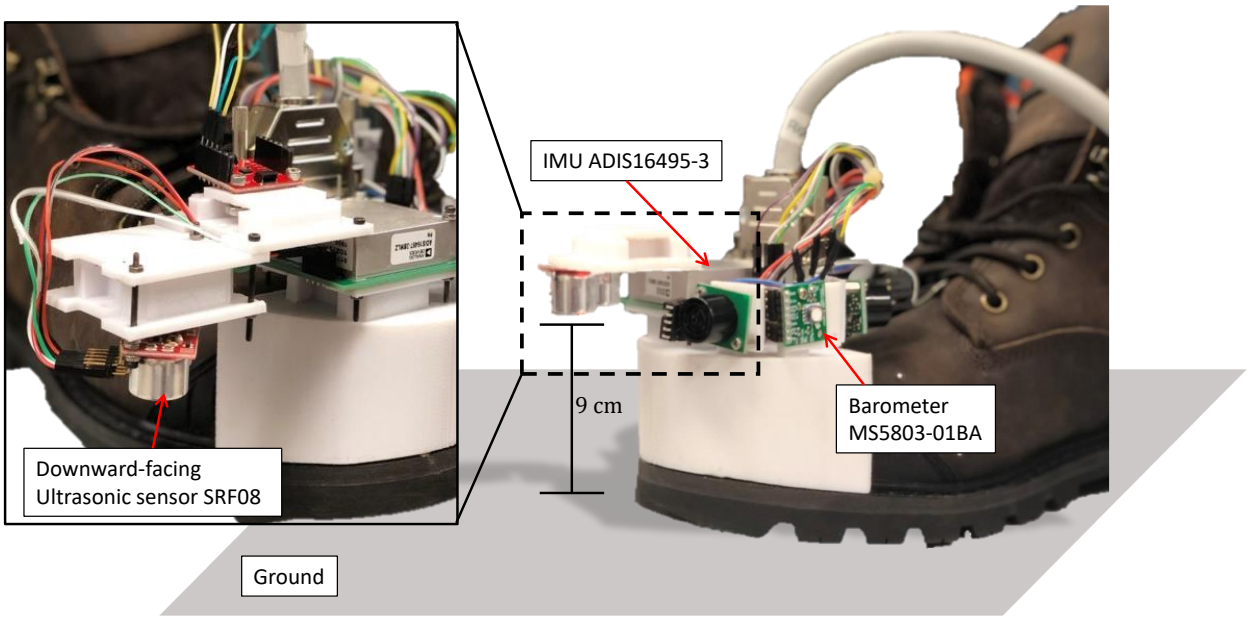


Figure 7.5: The Lab-On-Shoe platform integrated with a downward-facing ultrasonic sensor SRF08 and a barometric altimeter MS5803–01BA.

[31] integrated with a downward-facing ultrasonic sensor SRF08 and a barometric altimeter MS5803–01BA, shown in Figure 7.5. The sampling rates of the ultrasonic sensor and the barometer were set to 25 Hz and 5 Hz, respectively.

Since distance measurements of the ultrasonic sensor are discretized, two consecutive distinct measurements always have a discontinuity. In the case of walking at a speed of 40 steps per minute on a flat plane, the maximum discontinuity, α, can be approximated as follows:

$$\alpha = \frac{\max(\text{Foot relative elevation})}{F_s \times \frac{T_{\text{swing phase}}}{2}}$$

where F_s is the ultrasonic sampling rate and $T_{swing\ phase}$ is the swing phase duration. Then, the maximum discontinuity α was used as a threshold to determine smoothness of the measurements. If two consecutive measurements have a difference smaller than the threshold α, then the measurements during this period are considered to be smooth, and vice versa. In this section, the threshold α is set to 6 cm. Figure 7.6(a) presents an example of the

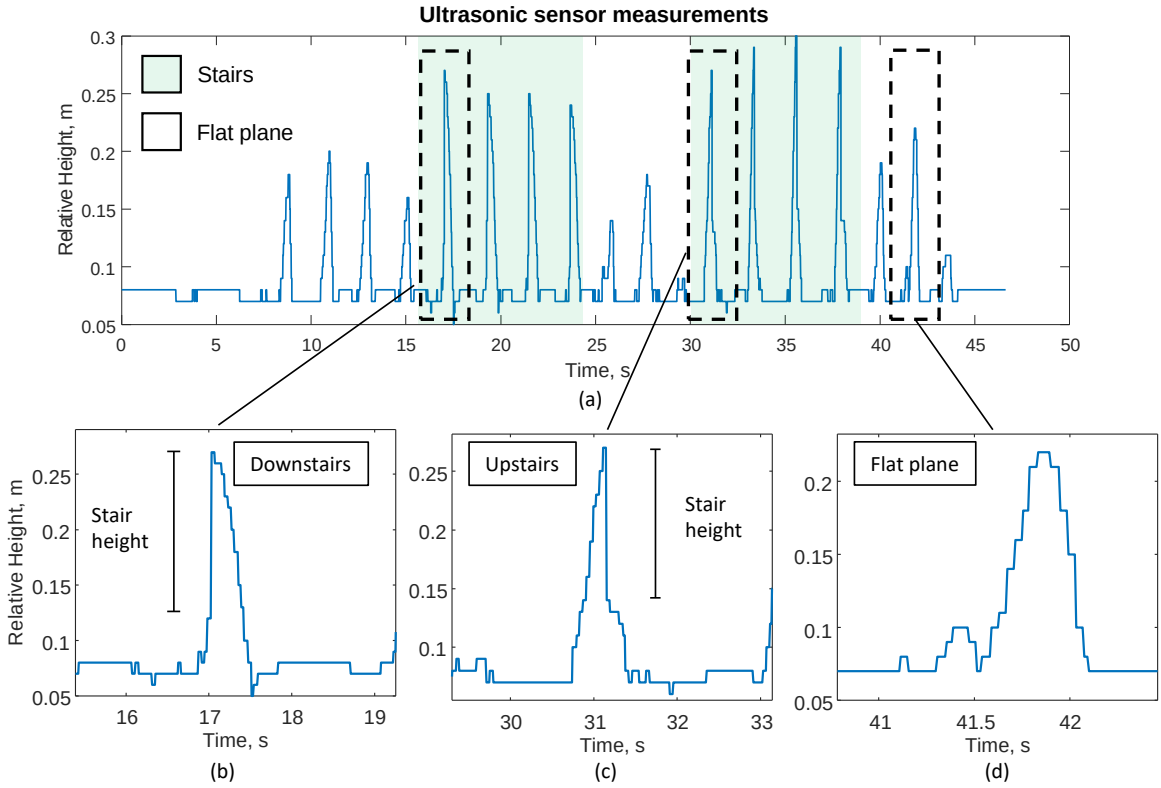


Figure 7.6: (a) The ultrasonic measurements collected by a shoe-mounted downward-facing ultrasonic sensor SRF08 during the experiment of walking indoor on flat surfaces, upstairs, and downstairs. The height of each stair was assumed to be nominally identical and was around 15 cm. The total elapsed time in this experiment was 46.5 s. In the period of the first 16 s, 24.5 s to 30.5 s, and 38.5 s to the end, a subject walked on a flat plane. From 16 s to 24.5 s, the subject went down four stairs. From 30.5 s to 38.5 s, the subject went up four stairs to the original height level. (b) ultrasonic profile in the case of downstairs. (c) ultrasonic profile in the case of upstairs. (d) ultrasonic profile in the case of flat plane.

ultrasonic measurements collected during an experiment of walking indoor at a speed of 40 steps per minute on flat surfaces, upstairs, and downstairs. In Figure 7.6(a), multiple humps in the ultrasonic measurements can be observed. Each hump corresponded to the swing phase in a gait cycle. In Figure 7.6(b), the first half of the hump had a discontinuity when going downstairs. In Figure 7.6(c), when going upstairs, a discontinuity appeared on the second half of the hump. The ultrasonic measurements in the case of walking on flat surfaces, shown in Figure 7.6(d), were smooth because there were not any two consecutive measurements that had a difference larger than α.

Kalman Filter for Ultrasonic Altimeter

A standard KF to realize the estimation of the vertical position of the shoe and the floor was used. The KF has the following states:

$$\mathbf{x}_k = [h_k, V_k, L_k]^\top$$

where h_k, V_k, and L_k are the shoe height, the shoe vertical velocity, and the floor height in the navigation frame at time k, respectively. The propagation step of the KF uses the relation $h_{k+1} = h_k + V_k dt$, where dt is the sampling rate of the ultrasonic sensor. Furthermore, the EKF uses an assumption that floor does not move and the shoe velocity in the vertical direction does not change in the propagation step. The discrete propagation matrices $\mathbf{F}_k$ and $\mathbf{Q}_k$ of the KF are formulated as follows:

$$\mathbf{F}_k = \begin{bmatrix} 1 & dt & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \mathbf{Q}_k = \begin{bmatrix} \sigma_{h_k}^2 & 0 & 0 \\ 0 & \sigma_{V_k}^2 & 0 \\ 0 & 0 & \sigma_{L_k}^2 \end{bmatrix}$$

where $\sigma^2_{h_k}$, $\sigma^2_{V_k}$, and $\sigma^2_{L_k}$ are the variances of noise corresponding to shoe height, shoe vertical velocity, and floor elevation, modeled as zero-mean Gaussian.

In the update step of the KF, the ultrasonic sensor provides three types of data: 1) relative distance between the shoe and the floor $d_{k,\text{SONAR}}$, which are the raw readouts from the sensor, 2) vertical velocity of the shoe $V_{k,\text{SONAR}}$, obtained by subtracting two consecutive ultrasonic measurements, and 3) floor elevation in navigation frame $L_{k,\text{SONAR}}$. The ultrasonic measurement of the floor elevation is achieved by observation that a discontinuity Δh is displayed in ultrasonic measurements when the sensor is passing through edges of stairs. The value of the discontinuity Δh, as the ultrasonic sensor scans through edges of stairs, is considered to be the change of floor elevation when both the pitch angle and the vertical velocity of the shoe are very close to zero. A consequence of this configuration is that the value of the floor elevation measurement is equal to the current floor height when no discontinuity is observed, which is the case for flat surfaces. The measurement vector z_k and update matrices H and R of the KF are formulated as follows:

$$\mathbf{z}_k = \begin{bmatrix} V_{k,\text{SONAR}} \\ L_{k,\text{SONAR}} \\ d_{k,\text{SONAR}} \end{bmatrix} = \begin{bmatrix} \frac{d_{k,\text{SONAR}} - d_{k-1,\text{SONAR}}}{dt} \\ L_{k-1,\text{SONAR}} - \Delta h \\ d_{k,\text{SONAR}} \end{bmatrix} = \begin{bmatrix} V_k + n_{V_{k,\text{SONAR}}} \\ L_{k,\text{SONAR}} + n_{L_{k,\text{SONAR}}} \\ h_k - L_k + n_{d_{k,\text{SONAR}}} \end{bmatrix}$$

$$\mathbf{H}_k = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & -1 \end{bmatrix}, \mathbf{R}_k = \begin{bmatrix} \sigma^2_{h_{k,\text{SONAR}}} & 0 & 0 \\ 0 & \sigma^2_{V_{k,\text{SONAR}}} & 0 \\ 0 & 0 & \sigma^2_{L_{k,\text{SONAR}}} \end{bmatrix}$$

where $n_{d_{k,\text{SONAR}}}$, $n_{V_{k,\text{SONAR}}}$, and $n_{L_{k,\text{SONAR}}}$ are the noises corresponding to ultrasonic measurements of shoe relative height, shoe vertical velocity, and floor elevation, modeled as zero-mean Gaussian with respective variances $\sigma^2_{h_{k,\text{SONAR}}}$, $\sigma^2_{V_{k,\text{SONAR}}}$, and $\sigma^2_{L_{k,\text{SONAR}}}$.

KF Parameter	Values
n_{h_k}	0.01
n_{V_k}	0.01
n_{L_k}	0.1
$n_{h_{k,\mathrm{SONAR}}}$	0.01
$n_{V_{k,\mathrm{SONAR}}}$	0.05
$n_{L_{k,\mathrm{SONAR}}}$	1

Table 7.1: Noise Characteristics of the Kalman Filter for the Ultrasonic Altimeter.

Ultrasonic Altimeter Performance Evaluation

This section presents an example of measurements of an ultrasonic altimeter. We tested the ultrasonic altimeter with the indoor walking experiment discussed in Figure 7.6. The initial height was assumed to be 17 [m] above the sea level. The noise characteristics, determined according to the nominal specification described in the datasheet of the ultrasonic sensor SRF08, were summarized in Table 7.1. In this experiment, the threshold α for determining the smoothness of the ultrasonic measurements was set to 6 cm. Figure 7.7(a) shows the estimated heights of the shoe and the floor, represented by a red curve and a blue curve, respectively. The black curve in Figure 7.7(a) illustrates the vertical positions estimated by the standalone ZUPT-aided INS based on IMU measurements collected in the same experiment. Details of the implementation of the standalone ZUPT-aided INS can be found in [91]. In this experiment, the accumulated error along the vertical direction of the standalone ZUPT-aided INS was 14 cm, while the error of the ultrasonic altimeter decreased to 1 cm. The remaining error sources of the ultrasonic altimeter could be due to 1) limited to resolution of the ultrasonic sensor, and 2) non-zero pitch angle of the shoe when discontinuities were observed, resulting in a shorter estimated floor level change because the sound waves might hit walls of the stairs, instead of ground. Filtering of these multi-path events would require an additional signal processing element.

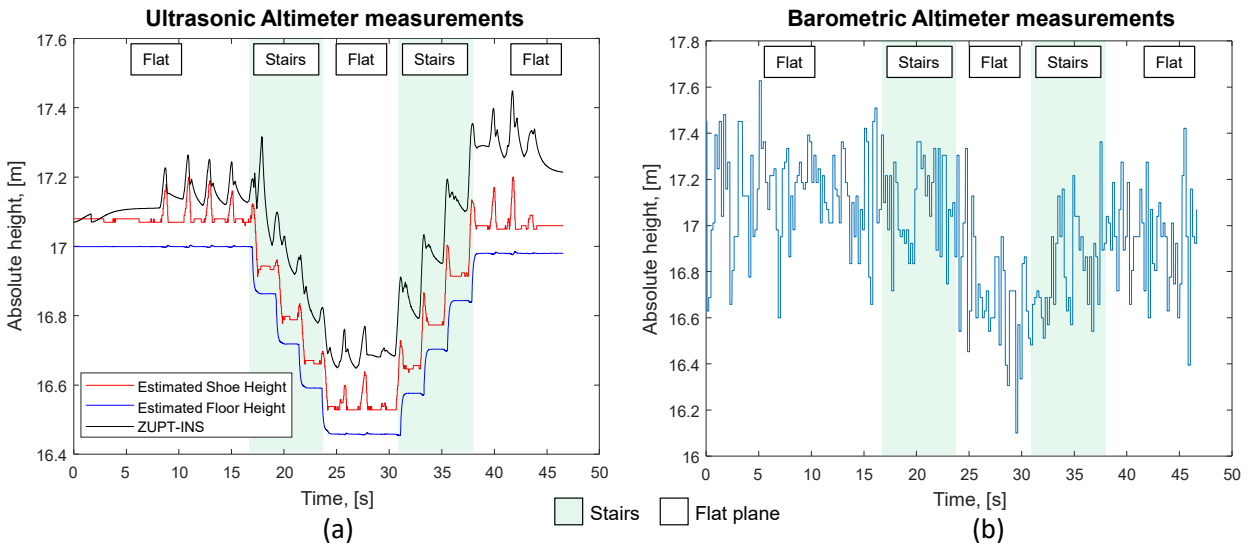


Figure 7.7: (a) Shoe height and floor height estimated by the proposed method, and shoe height determined by ZUPT-augmented INS. (b) Barometer readouts collected during the indoor walking experiment.

Figure 7.7(b) shows the readouts of the barometer MS5803−01BA collected in the same experiments discussed in Figure 7.7(a). Two observations can be made in Figure 7.7(a) and (b). First, the measurements of ultrasonic altimeter demonstrated a higher resolution than the barometer, which has the lowest resolution among Commercial-Off-The-Shelf sensors. Second, the ultrasonic altimeter captured subtle foot motions, which were not observed in the barometric measurements. The subtle foot motion could benefit the overall navigation results of a pedestrian INS.

7.4.2 Hybrid Altimeter

The hybrid altimeter presented in this section uses both measurements from the ultrasonic altimeter and the barometer. The main reason why the ultrasonic altimeter is not sufficient in its characteristics to replace barometers is that it does not account for other possible terrains that appear in indoor environments, such as ramp or elevator. It fails to operate under such conditions because, in the KF for the ultrasonic altimeter discussed in Section

7.4.1, the floor height state is updated only when a discontinuity (or a stair) is detected. When the ultrasonic altimeter operates under a ramp or inside an elevator, the estimated floor elevation will remain the same, which leads to incorrect estimation of the shoe height.

The hybrid altimeter aims to include the ultrasonic altimeter's advantages of stability and high resolution, while also leveraging barometer's capability of operation in the cases of ramp and elevator. Thus, the hybrid altimeter is designed to adaptively select the weight to put on the two types of altimeters based on the terrain under operation and to fuse the two measurements. The adaptive property is achieved with a Multi-Model KF, and the detection of different indoor terrains is realized with a foot-mounted IMU. Figure 7.8(a) presents the framework for the hybrid altimeter. In the case of flat surfaces or stairs, the measurements from the ultrasonic altimeter receive more weights in the fusion process. When operating inside an elevator or on a ramp, the hybrid altimeter increases the weights of the barometer.

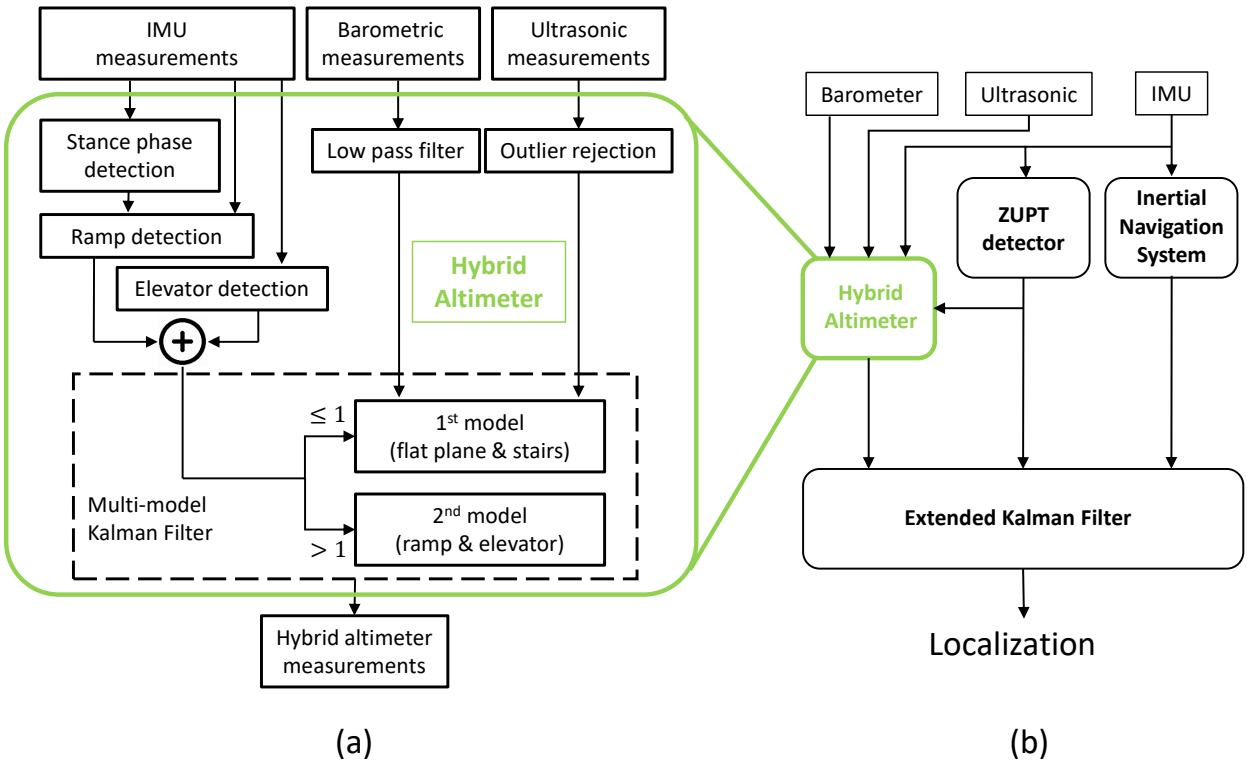


Figure 7.8: (a) The framework for the hybrid ultrasonic/barometric altimeter. (b) ZUPT-aided INS augmented by the hybrid altimeter.

Next, the principles of the elevator detection and the ramp detection will be discussed. The configuration of the Multi-Model KF will also be explained in detail.

Ramp Detection

The detection of ramps in the hybrid altimeter framework is designed to be accomplished by a foot-mounted IMU. An example of using a foot-mounted IMU to detect indoor ramps was introduced in [97]. The detection principle used in this section is that during the stance phase in a gait cycle, when the motion experienced by an IMU is minimal, the pitch angle of the IMU in the case of ramps is different from the case of flat planes. For example, if the pitch angle of a foot-mounted IMU is zero when the shoe contacts a flat plane, it can be determined that the contacting ground is a ramp when the pitch angle of the IMU is not zero. To account for swing phases, the developed approach made an assumption that if the foot in the current stance phase rests on an incline, then the foot was traveling over the incline during the entire period of the previous swing phase.

The pitch angle θ of the foot at time k can be directly calculated by accelerometers measurements y_k^a when the foot motion is minimum and can be expressed as follows:

$$\theta(y_k^a) = \tan^{-1}(\frac{y_k^{a,x}}{\sqrt{(y_k^{a,y})^2 + (y_k^{a,z})^2}})$$

where $y_k^a = [y_k^{a,x\top}, y_k^{a,y\top}, y_k^{a,z\top}]^\top$ and $y_k^{a,x}, y_k^{a,y}$, and $y_k^{a,z}$ are the readouts of the accelerometers along the x, y, and z axis at time k, respectively.

The status of the foot can be determined by a stance phase detector. This section uses the

SHOE detector, which determines a stance phase if

$$T(z_n) = \frac{1}{N}\sum_{k\in\Omega_n}(\frac{1}{\sigma_\alpha^2} \parallel y_k^\alpha - g\frac{\bar{y}_k^\alpha}{\parallel \bar{y}_k^\alpha \parallel} \parallel^2 + \frac{1}{\sigma_\omega^2} \parallel y_k^\omega \parallel^2) < \gamma, \tag{7.4}$$

where $\Omega_n = l \in \mathbb{N}, n \leq l \leq N-1$ is a collection of the IMU measurement indexes at time n with a window of length N, $z_n = \{[y_k^{\alpha\top}, y_k^{\omega\top}]\}_{k=N}^{k=N-1}$ is a sequence of the IMU measurements in the window, $y_k^{\omega\top}$ are the gyroscope measurements at k, σ_a^2 is the noise variance of the accelerometer, σ_ω^2 is the noise variance of the gyroscope, and γ are user-defined thresholds.

Combining the pitch angle calculated from IMU measurements and stance phase detection, the ramp detector T_{ramp} can be formulated as follows:

Algorithm 1 Ramp detection

```
l= The last stance phase
if T(z_n) < γ then
    if |θ(y_n^α) − θ_0| < ϵ then T_ramp(z_n) = 1
        if T_ramp(z_l) then k = l + 1
            while k ≠ n do T_ramp(z_k) = 1 k = k + 1
            end while
        end if
    else if N is odd then T_ramp(z_n) = 0
    end if
else if N is odd then T_ramp(z_n) = 0
end ifn = n + 1
```

where θ_0 is the pitch angle obtained at the beginning of the experiment when the foot was the on the ground. The parameter ϵ is a threshold used here to improve the robustness of the detector since the estimated pitch angle in practice is rarely exactly equal to zero. $T_{ramp}(z_k) = 1$ indicates that a ramp is detected at time k.

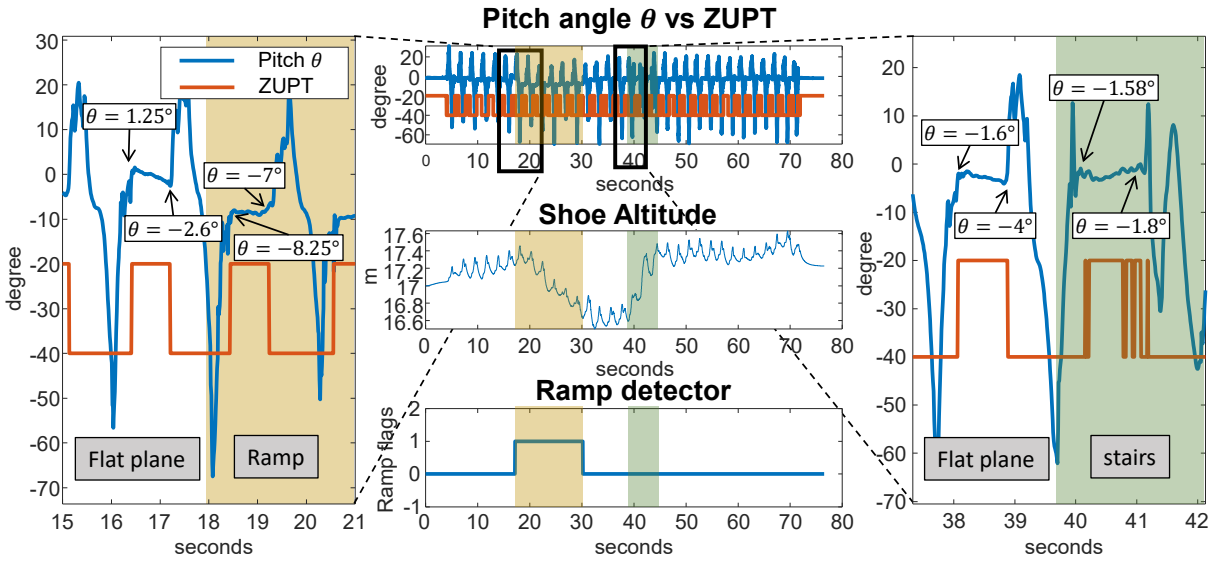
To test the performance of the developed ramp detector, a close-loop experiment with trajectory that included flat surfaces, stairs, and a ramp was conducted. Figure 7.9(b) shows the reference trajectory obtained from the standalone ZUPT-aided INS. In the experiment, the agent first walked towards the North for 7 meters on a flat surface in 16 seconds, turned

90° to the East and walked for 13 meters in 12 seconds on a ramp, turned 90° to the South and walked upstairs for three steps in 5 seconds, proceeded with walking on a flat surface for 6 meters in 10 seconds, turned 90° to the West and walked back to the starting point on a flat surface.

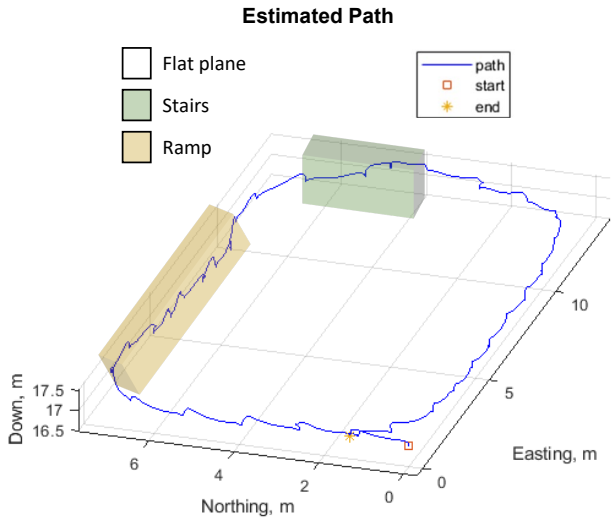
Figure 7.9(a) shows the pitch angle measurements, the stance phase status, the detected ramp flags, and the reference shoe height estimated by ZUPT-aided INS. The left-hand side of Figure 7.9(a) presents a zoomed-in view that includes two gait cycles in the experiments. The gait during 16s to 18s was on a flat surface, and the one during 18 s to 21.2 s was on a ramp. A clear difference between the pitch angles during the two stance phases can be observed. The pitch angles in the case of a flat plane ranged from $-2.6°$ to $1.25°$, while in the case of ramp, the range was from $-7°$ to $-8.25°$. The right-hand side of Figure 7.9(a) presents another zoomed-in view that includes two other gait cycles in the experiments. The gait during 38 s to 39.5 s was on a flat surface, and the one during 39.5 s to 41 s was on a stair. The detected pitch angles when foot rested on the flat plane and the stair had similar values, both had the range between $-4°$ to $-1.6°$. The discussed ramp detector with a $\epsilon = 4°$ achieved a 100% accuracy rate and no false alarm in this experiment.

Elevator Detection

Elevator detection can be achieved with the z-axis accelerometer of a foot-mounted IMU. An example of elevator detection using the z-axis accelerometer was presented in [248]. The detection concept is that when a person with a foot-mounted IMU is standing inside a moving elevator the direction of the force experienced by the IMU within a period of time is consistent, while in the same length of time, the direction of the force generated by foot dynamics is inconsistent. Figure 7.10(c), (d), and (f) illustrate examples of vertical accelerations measured by the Lab-On-Shoe platform in the cases of walking slowly, walking fast, and standing inside a moving elevator. In Figure 7.10(d) and (f), it can be seen that the



(a)



(b)

Figure 7.9: (a) The pitch angle measurements estimated by accelerometers, the stance phase status, the detected ramp flags, and the reference shoe height estimated by ZUPT-aided INS in the experiment for ramp detection. (b) The reference trajectory, obtained by the ZUPT-aided INS, of the experiment for ramp detection.

period of time when the IMU is experiencing the same direction of acceleration was 0.3s in the case of walking slowly and 0.2s in the case of walking fast. In Figure 7.10(c), the period of time that the IMU experienced acceleration from the same direction was 1.5s. Thus, the elevator detection can be achieved by observing a window with length N of the z-axis accelerometer measurements. If all the measurements within the window are consistently larger or consistently smaller than the gravity, then the motion is generated by the elevator instead of the foot. The detection mechanism can be described as follows:

Algorithm 2 Elevator detection

1: **if** $y_k^{a,z} < g \forall k \in \Omega_n$ or $y_k^{a,z} > g \forall k \in \Omega_n$ **then** $T_{Elevator}(z_n) = 1$
2: **else if** N is odd **then** $T_{Elevator}(z_n) = 0$
3: **end if**

where $\Omega_n = l \in N, n \leq l \leq N-1$ is a collection of the IMU measurement indexes at time n with a window of length N, $z_n = \{[y_k^{aT}, y_k^{\omega T}]^T\}_{k=n}^{k=N-1}$ is a sequence of the IMU measurements in the window, and $y_k^{a,z}$ is the z-axis accelerometer measurement at time k.

To test the performance of the elevator detector, an indoor experiment including walking and standing in a moving elevator was conducted. In the experiment, the subject started from the fourth floor of a four-story building and walked into the elevator. During 17s to 51s, 86s to 92s, 126s to 140s, 185s to 191s, and 208s to 218s, the subject was standing inside the moving elevator. The periods between 51s and 86s,92s and 126s, 140s and 185s, the subject moved out of the elevator, walked around on a flat surface, and went back into the elevator. During 191s and 208s, the subject stood inside of the still elevator. The total time duration of the experiment was 232s. The vertical trajectory of the subject was illustrated by a barometer, shown in Figure 7.10(a). Figure 7.10(b) demonstrates the accelerometer measurements collected during the experiment and Figure 7.10(e) presents the detection results. In this experiment, we set the window of length to be 1s. The detection of elevator motion achieved 100% detection rate with no false alarm.

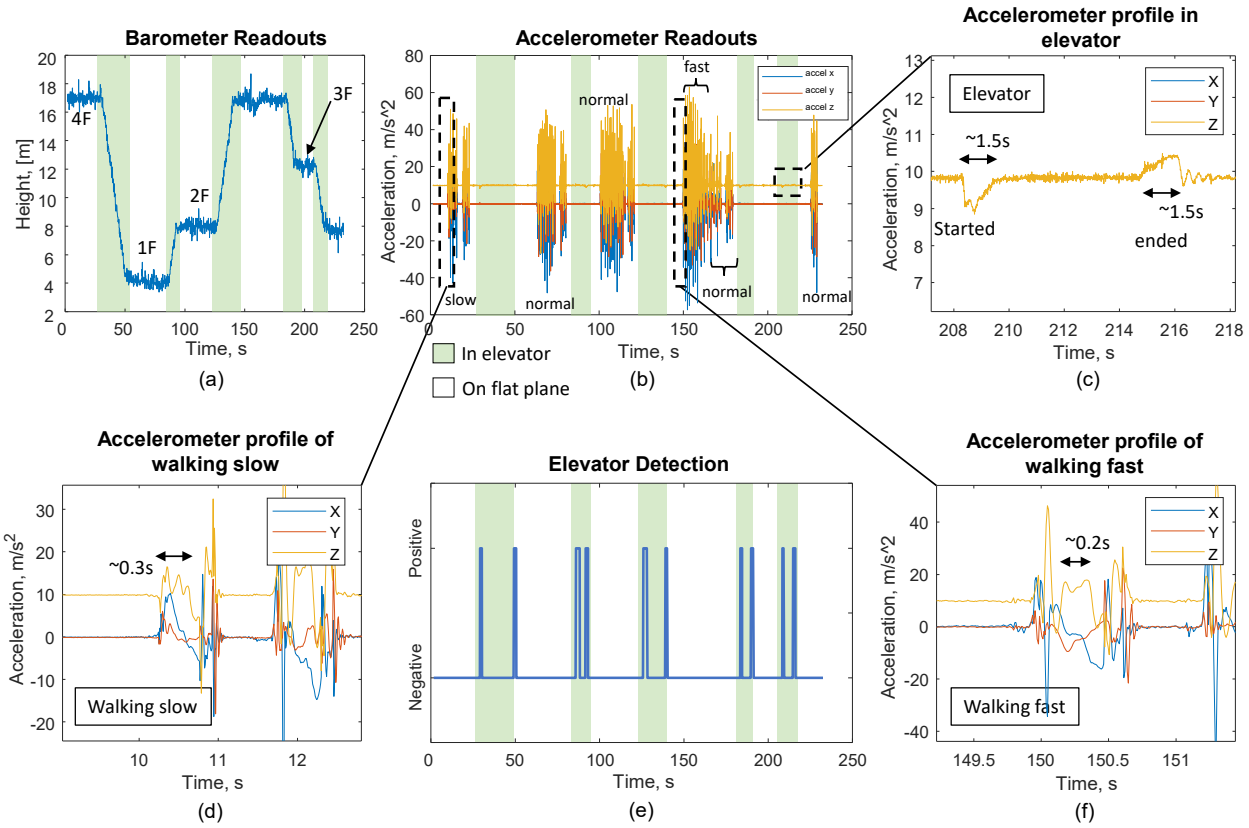


Figure 7.10: (a) The reference vertical trajectory, estimated by the barometer, of the experiment for elevator detection. (b) The accelerometer measurements collected during the experiment. (c) Acceleration profiles of the start and the end of the elevator motion. (d) Acceleration profiles of walking slowly. (e) The elevator detection results of the experiment. (d) Acceleration profiles of walking fast.

Multi-Model Kalman Filter for the Hybrid Altimeter

The hybrid altimeter uses both ultrasonic and barometric altimeters. Since barometric measurements have a bias that comes from ambient weather changes, the bias also needs to be estimated. The KF state discussed in Section 7.4.1 is augmented with a bias state b_k for the barometer. The augmented KF states are formulated as follows:

$$\mathbf{x}_k = [h_k, V_k, L_k, b_k]^\top.$$

In the propagation step of the Multi-Model KF, we assumed that the bias state b_k, whose noise n_{b_k} is modeled as a Gaussian distribution, maintains the same expected value. The dynamics of the other states is inherited from the KF for the ultrasonic altimeter. Thus, the propagation matrices $\mathbf{F}$ and $\mathbf{Q}$ of the KF can be expressed as follows:

$$\mathbf{F}_k = \begin{bmatrix} 1 & dt & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \mathbf{Q}_k = \begin{bmatrix} \sigma_{h_k}^2 & 0 & 0 & 0 \\ 0 & \sigma_{V_k}^2 & 0 & 0 \\ 0 & 0 & \sigma_{L_k}^2 & 0 \\ 0 & 0 & 0 & \sigma_{b_k}^2 \end{bmatrix}$$

The barometer readout $h_{k,\text{baro}}$ is considered as the summation of true heights h_k, current sensor bias b_k, and a Gaussian noise $n_{h_{k,\text{baro}}}$ with variance $\sigma_{h_{k,\text{baro}}}^2$, which can be expressed as:

$$h_{k,\text{baro}} = h_k + b_k + n_{h_{k,\text{baro}}}.$$

In the update step of the Multi-Model KF for the proposed hybrid altimeter, the measurement vector z_k of the KF for the ultrasonic altimeter was augmented with the height measurement $h_{k,\text{baro}}$ from the barometer. The measurement vector z_k of the Multi-model

KF is expressed as:

$$\mathbf{z}_k = \begin{bmatrix} h_{k,\text{baro}} \\ V_{k,\text{SONAR}} \\ L_{k,\text{SONAR}} \\ d_{k,\text{SONAR}} \end{bmatrix}$$

and the corresponding measurement matrices $\mathbf{H}$ and $\mathbf{R}$ are formulated as follows:

$$\mathbf{H}_k = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & -1 \end{bmatrix}, \mathbf{R}_k = \begin{bmatrix} \sigma^2_{h_{k,\text{baro}}} & 0 & 0 & 0 \\ 0 & \sigma^2_{h_{k,\text{SONAR}}} & 0 & 0 \\ 0 & 0 & \sigma^2_{V_{k,\text{SONAR}}} & 0 \\ 0 & 0 & 0 & \sigma^2_{L_{k,\text{SONAR}}} \end{bmatrix}$$

The Multi-Model KF has two models. The two models differ in the noise characteristics of the states h_k, V_k, L_k, and b_k and the measurements $V_{k,\text{SONAR}}$ and $L_{k,\text{SONAR}}$. In the first model, where the altimeter is assumed to operate over flat surfaces and ramps, the noise variance configuration, except for the states b_k, is the same as in the case of the ultrasonic altimeter. As for the state b_k, since it is equivalent to $h_{k,\text{baro}} - h_k$ and the state h_k in the first model is mainly affected by the ultrasonic altimeter, the noise variance is set to a value comparable to the maximum distance measured by the ultrasonic sensor. In the second model, which is the case of elevators and ramps, the noise characteristics of the states h_k, V_k, and L_k and the measurements $V_{k,\text{SONAR}}$ and $L_{k,\text{SONAR}}$ are increased because under these terrains, the ultrasonic altimeter does not provide reliable measurements. The noise variance of the state b_k, on the other hand, is set to a value that was experimentally determined. This value is usually much lower than the value in the first model. Table 7.2 summarizes the values used in this section for the noise characteristics of the Multi-Model KF.

The framework presented in Figure 7.8(a) also includes a low-pass filter for barometer mea-

Table 7.2: Noise Characteristics of the Multi-Model Kalman Filter for the Hybrid Altimeter

KF Parameter	1^{st} model value	2^{nd} value
n_{h_k}	0.01	100
n_{V_k}	0.01	10
n_{L_k}	0.1	1
$n_{h_{k,\text{SONAR}}}$	0.01	10
$n_{V_{k,\text{SONAR}}}$	0.05	10
$n_{L_{k,\text{SONAR}}}$	1	0.01
$n_{h_{k,\text{baro}}}$	1	1

surements and an outlier rejector module for ultrasonic measurements. The low-pass filter is included because raw measurements of the barometer contain high-frequency thermal and electronic noises. The outlier rejector module is designed to improve the false alarm rate for the stair detection. Machine learning techniques have the potential to classify different gait motion phases based on the ultrasonic altimeter readouts and can be expected to improve the stair detection further.

7.4.3 Experimental Verification For Hybrid Altimeter

To validate the proposed hybrid altimeter, three series of experiments using the Lab-On-Shoe platform, shown in Figure 7.5, were conducted. In every experiment, the sampling rates of the IMU, barometer, and the downward-facing ultrasonic sensor were set to 120Hz,5Hz, and 25Hz, respectively. The first series of experiments illustrated that the hybrid altimeter is more robust to temperature and air pressure variations than a conventional barometer. In the second series of experiments of walking on a flat plane, we demonstrated that when estimating a vertical displacement, the accuracy of the ZUPT-based INS aided by a hybrid altimeter outperformed both ZUPT-aided INS cases, standalone and barometer-aided. The

third series of experiments investigated the performance of the ZUPT-based INS aided by the hybrid altimeter when walking on different terrains, such as flat surfaces, stairs, ramps, and elevators.

Effect of Temperature And Air Pressure Changes

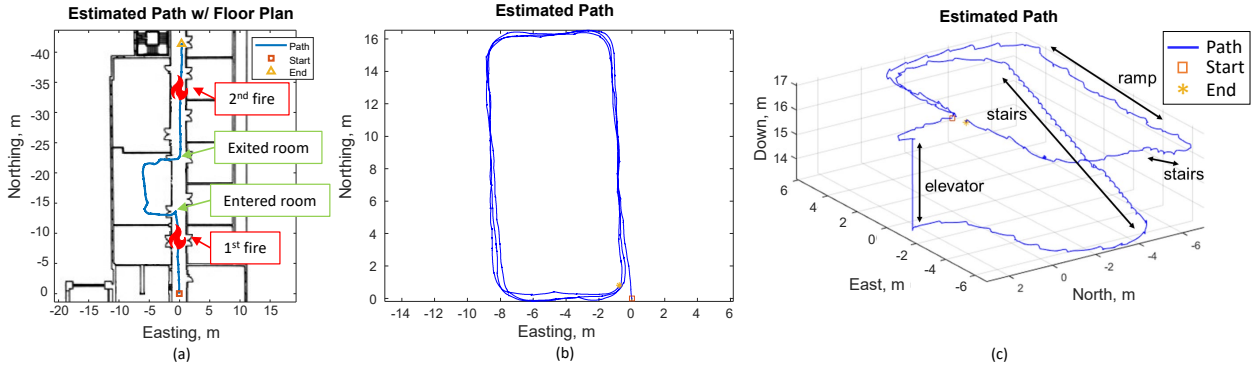


Figure 7.11: (a) Reference trajectory for the experiments when barometer is subject to temperature and air pressure changes. (b) Reference trajectory for the indoor experiment walking on a flat plane. (c) Reference trajectory for the indoor experiment walking on different terrains.

To investigate robustness of the hybrid altimeter measurements when the barometer is subjected to variations due weather changes, an experiment simulating an environment where surrounding temperature and air pressure are unstable was conducted. The nominal trajectory of the experiment is shown in Figure 7.11(a), which was obtained by a standalone ZUPT-aided INS. A subject started the navigation from a reference position in the hallway and the trajectory was recorded by the Lab-On-Shoe platform. The subject started to walk 10 [m] towards the South. At the 60 s mark, the subject stopped and during this time a fire was simulated by heating up the barometer for 0.1 s with a commercial lighter. The subject continued the navigation and entered a room at 78 s, walked for 8 [m], and left the room at 100 s. Then, the subject walked for another 10 [m] towards the South and stopped at 120 s. At 140 s, another heat source lasting 0.1 s was triggered next to the barometer. The subject moved toward the South for another 7 [m] to reach the destination. The time duration of

this experiment was 180 s, and the nominal heights of the starting point and the ending point were the same. In the experiment, the subject walked at a speed of approximately 40 steps per minute. The results were analyzed in the next paragraph.

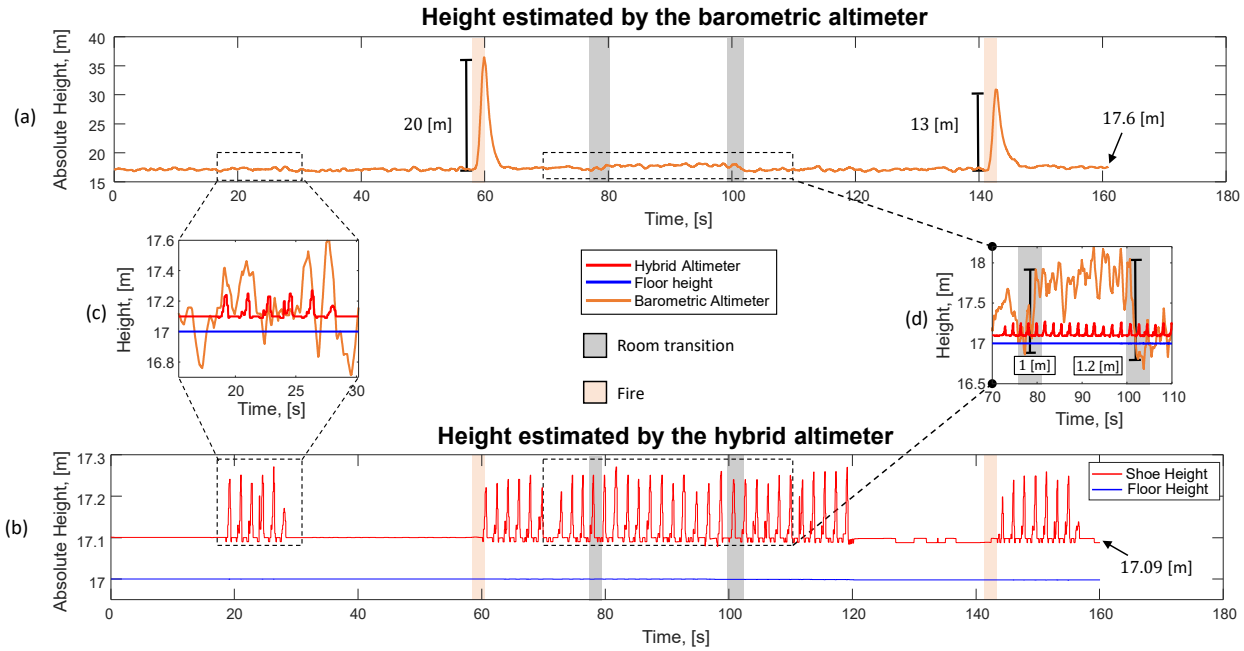


Figure 7.12: (a) The height, measured by the barometer, of the experiment discussed in Section IVA. (b) The height measured by the hybrid altimeter of the experiment. (c) Comparison of the two altimeters in walking in an environment with stable air pressure and temperature. The hybrid altimeter could capture the subtle foot motion while the barometer failed to do so. (d) Comparison of the two altimeters in the case of air pressure changed due to the room transition. The height measured by the barometer was affected by the transition while the measurement of the hybrid altimeter maintained stable.

Figure 8(a) shows the shoe height estimated by the barometer and Figure 8(b) demonstrates the shoe height and the floor height estimated by the hybrid altimeter. Figure 8(a) illustrates that the first fire and the second fire caused vertical errors of 20 [m] and 13 [m] for the barometer, respectively. Figure 8(c) is a zoomed-in view of the sensor measurements during 70 s and 110 s, showing that the transitions between the hallway to the room led to the vertical errors of 1 [m] and 1.2 [m], as judged from measurement of the barometer. Figure 8(b) demonstrates that the two events of fire and the transitions between different rooms had a minimal effect on the hybrid altimeter measurements. Figure 8(b) depicts that the hybrid

altimeter can capture subtle foot motions while the barometer is incapable of capturing the motions due to insufficient resolution.

This experiment investigated the effect of air pressure and temperature changes on readings of the barometers. The operational principle of the ultrasonic sensor involves transmission of the sound wave in air, which is affected by the changes in ambient air pressure and temperature. This effect was not accounted in these experiments.

Experiments of walking on flat planes

To derive the performance of the hybrid altimeter, when used to assist the ZUPT-aided INS, we first performed experiments of walking indoor at a speed of approximately 40 steps per minute in a square shape pattern for three full circles. we repeated the same experiment 10 times. In the experiment, the starting position and the ending position were the same. The total trajectory length was 150 [m], and the navigation time was 232 s. The experiment was conducted on the same floor, so there was no floor height changes during the entire experiment. we assumed that the initial height was 17m above the sea level. Figure 7.11(b) shows a nominal horizontal trajectory obtained by the ZUPT-aided INS. Since the altimeter measurements do not have significant impacts on the displacement errors along the horizontal directions, we only focused on errors along the vertical direction. The ZUPT-aided INS augmented by an altimeter was used for localization, as described in [91]. Note, that the altimeter used in [91] was a barometer but the same analytical framework can be applied when the barometer is replaced by the proposed hybrid altimeter. Figure 4(b) illustrates the configuration of the ZUPT-aided INS augmented by the hybrid altimeter.

Figure 7.13(a) shows the estimated height measurements from 1) standalone ZUPT-based INS, 2) ZUPT-based INS aided by the barometer, and 3) ZUPT-based INS aided by the hybrid altimeter. Figure 7.13(b) shows the final vertical displacements of the 10 sets of

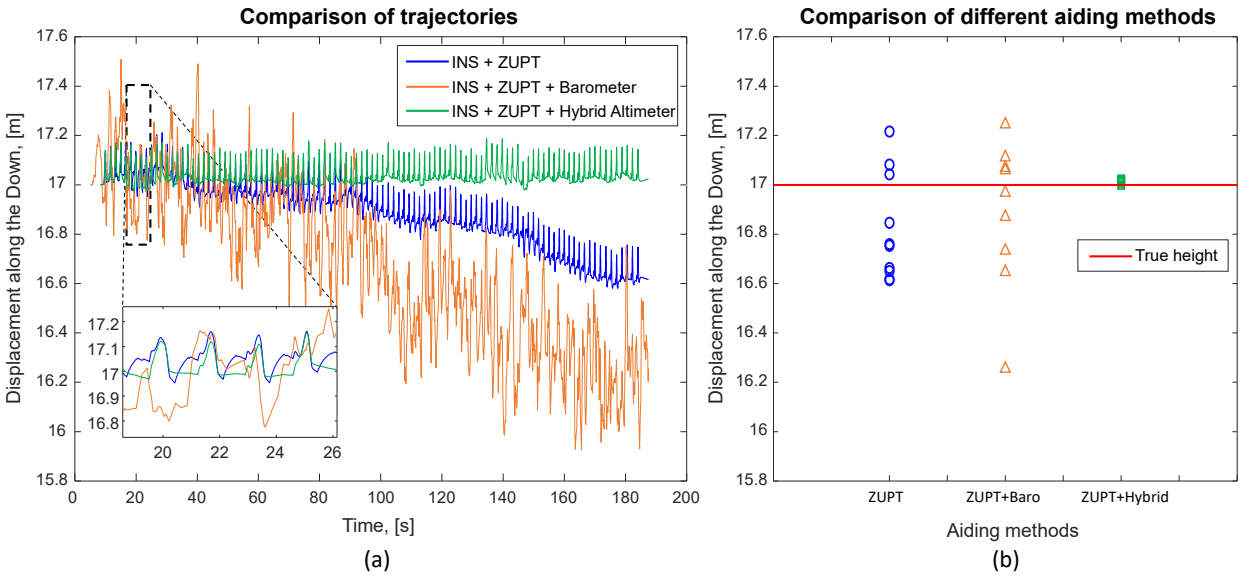


Figure 7.13: (a) An example of the height estimated by a standalone ZUPT-based INS, ZUPT-based INS aided by a barometer, and ZUPT-based INS aided by a hybrid altimeter in the experiments where the subject walked on a flat plan. (b) Vertical displacement accuracy of all three navigation solutions.

Table 7.3: RMSEs of a standalone ZUPT-based INS, ZUPT-based INS aided by the barometer, and ZUPT-based INS aided by the hybrid altimeter in the experiments of walking on a flat plane.

INS aiding method	Height RMSE [m]
ZUPT	0.272
ZUPT + Barometer	0.453
ZUPT + Hybrid Altimeter	0.01

experiments estimated by the three different navigation solutions. The red horizontal line in Figure 7.13(b) indicates the true height. we calculated RMSE based on the 10 sets of the experiment, summarized in Table 7.3, and found that the RMSEs of the standalone ZUPT-based INS, the ZUPT-based INS aided by the barometer, and the ZUPT-based INS aided by the hybrid altimeter were 0.27 [m], 0.453 [m], and 0.01 [m], respectively. In this series of experiments, the ZUPT-based INS aided by the hybrid altimeter reduced the error by 96%, as compared to the standalone ZUPT-based INS, and by 97%, as compare to the ZUPT-based INS aided by the barometer.

Experiments Of Walking On Different Terrains

To demonstrate the hybrid altimeter in a more realistic situations, we conducted a series of experiments with a nominal trajectory that included an elevator, a ramp, flat surfaces, and stairs. The total navigation time in each experiment was 210s, and the length was 92m. We repeated 10 sets of the identical experiment. Figure 7.11(c) demonstrates a reference trajectory of the experiment obtained by the ZUPT-aided INS augmented by the hybrid altimeter. At the beginning of each experiment, an agent who wore the Lab-On-Shoe platform would start the experiment on the second floor of a building. In the first 33 s, the subject moved from the starting point to an elevator. Between 33 s and 41 s, the elevator moved down one floor, whose nominal height was measured to be 3.8 [m]. From 41 s to 70 s, the subject moved out of the elevator and walked on a flat surface to stairs. Between 70 s to 135 s, the subject walked upstairs for 26 stairs. The nominal height of each of the stairs was 15 cm. After the first eight stairs, there was a small area of a flat surface, where the subject walked one step during 100 s to 102 s time interval. Between 135 s and 145 s, the subject walked on a flat surface to a ramp. From 145 s to 165 s, the subject walked down along the ramp. The nominal height difference between the two ends of the ramp was 60 cm. From 165 s to 175 s, the subject moved on a flat surface to other stairs. From 175 s to 182 s, the subject walked upstairs for four stairs. The nominal height of each of the stairs was measured to be 15 cm. From 182 s to the end of the experiment, the subject walked back to the starting point on a flat surface. In this series of experiments, the subject walked at a speed of approximately 40 steps per minute. The results of this experiment is discussed in the next paragraph.

We compared vertical displacements estimated by the ZUPT-augmented INS, the ZUPT-augmented INS aided by the barometer, and the ZUPT-augmented INS aided by the hybrid altimeter. It should be pointed out that because of the stair height estimation error of the ultrasonic altimeter discussed in Section 7.4.1.B, we applied a constant to compensate stair height estimation. In this series of experiments, we found that the average value of the

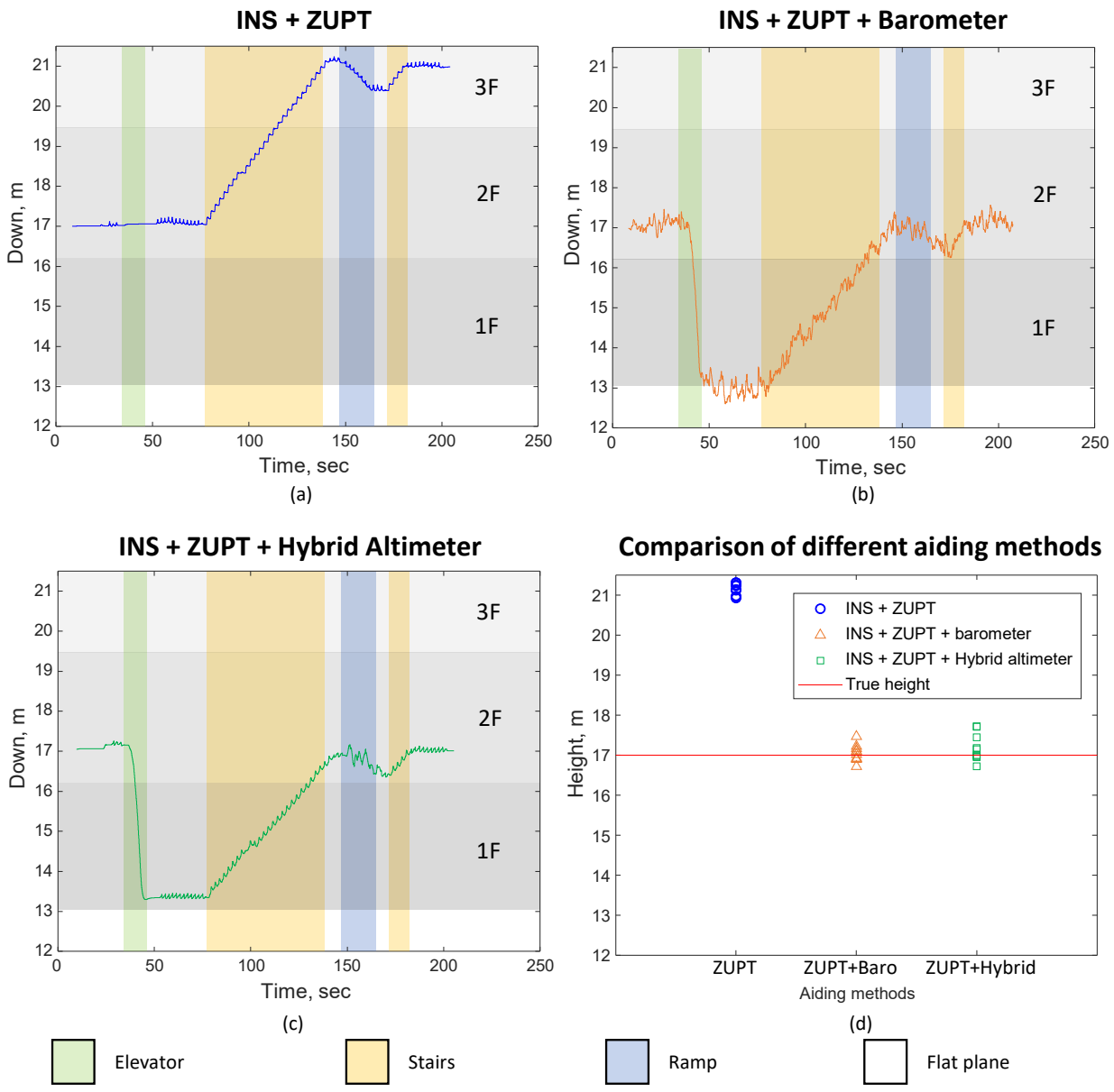


Figure 7.14: (a), (b), and (c) examples of the height estimated by standalone ZUPT-based INS, ZUPT-based INS aided by the barometer, and ZUPT-based INS aided by the hybrid altimeter in the experiments presented in Section 7.4.3.C. (d) The final vertical displacements of the three navigation solutions.

estimated stair heights was 11.5 cm, which was 3.5 cm less than the nominal stair height. Thus, the compensation value was selected to be 3.5 cm in this series of experiments. The results of three navigation solutions are shown in Figure 7.14(a), (b), and (c), respectively. The final vertical displacements of the 10 sets of the experiments for the three navigation

Table 7.4: RMSEs of the standalone ZUPT-based INS, ZUPT-based INS aided by the barometer, and ZUPT-based INS aided by the hybrid altimeter in the experiments of walking on different terrains, including flat surfaces, a ramp, stairs, and an elevator.

INS aiding method	Height RMSE [m]
ZUPT	4.15
ZUPT + Barometer	0.21
ZUPT + Hybrid Altimeter	0.36

solutions are shown in Figure 7.14(d). The RMSEs of the three methods summarized in Table 7.4. Figure 7.14(a) illustrates that standalone ZUPT-augmented INS could not account for the elevator motion, leading to identification of a wrong floor. Although in this series of experiments, the RMSE of the ZUPT-augmented INS aided by the barometer was smaller than when aided by the hybrid altimeter, instability in the estimated height profiles can be observed in the case with the barometer. For example, in Figure 7.14(b), during 41s and 75s, the ZUPT-augmented INS aided by the barometer indicated that the subject was under the ground level. The primary error sources of the hybrid altimeter were: 1) barometer bias when operating on a ramp or in an elevator and 2) stair height estimation error from the ultrasonic altimeter. The contributing factors of the stair height estimation error were discussed in Section 7.4.1.B.

Discussion

Section 7.4.3 has demonstrated that the proposed hybrid altimeter can achieve an accurate estimation of vertical displacement when operating under different terrains. However, two major challenges when using the proposed hybrid altimeter were identified.

1. Stair height underestimate. In our opinion, the underestimate is mainly contributed from three factors: 1) multi-path effects of ultrasonic sensors, 2) the fact that the vertical velocity of the foot when going above the edges of stairs is not zero, and 3)

wide detection cone of the ultrasonic sensor SRF08 resulting in detecting the wall of stairs when scanning through the edge of the stairs. The first factor could possibly be resolved by including an additional signal processing module in the hybrid altimeter framework. To address the second factor, additional velocity detectors are required. The third factor can be potentially eliminated by using a range sensor that has a narrow detection cone, such as LiDAR. The underestimate of stair height also implies that the assumption of the state L_k for floor height and the measurement $L_(k, SONAR)$ being zero-mean Gaussian random variables might be inaccurate.

2. False alarms of stairs detection. In the ultrasonic altimeter module, a threshold for determining the smoothness of the ultrasonic measurements was defined, and if the measurement is not smooth, or equivalently has a discontinuity, then a stair is detected. However, due to the low sampling rate of the ultrasonic sensor, in the case of walking fast or running, two consecutive ultrasonic measurements tend to have a difference larger than the smoothness threshold, even when walking on a flat plane. The large difference can result in false alarms of stair detection. This phenomenon could be reduced if an array of sensors with a high sampling rate is used. Another contributing factor to false alarms of stair detection is related to walking patterns. It could be observed that in some gait patterns, there is a short period during the swing phase that the bottom of the foot is facing the surrounding wall of a building, instead of the ground. This phenomenon would result in two significant discontinuities in the ultrasonic measurement collected during this period. One is a positive discontinuity at the beginning of the period, and the other is a negative one at the end of the period. Our solution to this problem was to use a temporal threshold to determine a false alarm of the stair detection if the time difference between occurrences of a positive and negative discontinuities is smaller than the temporal threshold.

This section presented a hybrid altimeter that uses a barometer and a downward-facing

ultrasonic altimeter for aiding foot-mounted INS. The development of the hybrid altimeter aims to minimize the usage of barometric altimeter in height estimation as measurements of barometers are easily affected by ambient temperature and air pressure changes. We first showed that a shoe-mounted downward-facing ultrasonic sensor alone could be used as an altimeter, in the case of flat surfaces and stairs, by simultaneously estimating shoe height and floor elevation. To account for other common indoor terrains, such as ramps and elevators, a multi-model KF to fuse measurements of the barometer and the ultrasonic altimeter was used. In the fusion process, the hybrid altimeter adaptively selects weights for the two altimeters based on the terrains under operation. In the cases of flat planes and stairs, the ultrasonic altimeter has high-resolution and stable measurements, thus the noise variances corresponding to the ultrasonic altimeter are set to lower values than the noise variance of the barometer. In the case of ramps and elevators, the barometer measurements are more reliable than the ultrasonic sensor; therefore, the noise variances of the barometer are decreased, and those of the ultrasonic altimeter are increased. Detection of elevators and ramps were shown in this section to be achieved with a foot-mounted IMU. Three series of experiments were conducted to test the performance of the proposed hybrid altimeter. The first experiment showed that the hybrid altimeter was less sensitive to temperature and air pressure changes in the surrounding environment, as compared to the barometer. The second series of experiments investigated the performance of the hybrid altimeter in the case of walking slowly on a flat plane, and experimental results indicated that the ZUPT-based INS aided by the hybrid altimeter improved the RMSE along the vertical direction by 96%, as compared to a standalone ZUPT-based INS, and by 97%, as compared to the ZUPT-based INS aided by the barometer. In the third series of experiments, we validated the operation of the hybrid altimeter in the cases of common indoor terrains, such as flat surfaces, stairs, ramps, and elevators. The experimental results showed that the RMSE of the ZUPT-based INS aided by the hybrid altimeter outperformed the RMSE of the standalone ZUPT-based INS by 91%. When compared to the RMSE of the ZUPT-based INS aided by the barometer,

the RMSE of the ZUPT-based INS aided by the hybrid altimeter was increased by 41% (0.15 m). In our opinion, the primary sources contributing to the vertical error of the ZUPT-based INS aided by the hybrid altimeter are the underestimate of stair height and the false alarm of stair detection of the ultrasonic altimeter. The results presented in this section was published in [92].

7.5 Real-Time Implementation of ZUPT-Altimeter/aided INS

In Section 7.4 and 4.4, we have shown that integrating the foot-mounted system with a downward-facing ultrasonic sensor could be beneficial for reducing the vertical position error and enhancing performance of the stance phase detection. In Section 7.4, a hybrid altimeter that combined a barometer and a downward-facing ultrasonic sensor was proposed to enhance the ZUPT-aided INS, and the integrated system was demonstrated with higher position resolution and increased robustness against ambient temperature and air pressure changes, as compared to a standalone barometer. In 4.4, the UA-SHOE detector was developed to achieve automatic identification of zero-velocity events. When compared to other existing advanced detectors utilizing adaptive mechanisms [206, 223] and machine learning [203, 225], the UA-SHOE fused measurements obtained from a downward-facing ultrasonic sensor and a foot-mounted IMU and was shown to be capable of functioning while a pedestrian performing different activities, such as walking and running.

Although the hybrid altimeter and the UA-SHOE detector have been demonstrated with advantageous performance, the authors indicated that the setups used in the two approaches were not ideal to do extensive performance evaluation, as the systems were relatively bulky, which could limit the range of motion of a pedestrian, and the on-board sensors were not optimized with high sampling rates, which might not fully reconstruct the complex foot motion. There are other existing platforms for evaluating foot-mounted pedestrian INS

[144, 117, 119]. However, these systems usually do not provide the flexibility to change hardware configuration easily, nor do they come with the freedom to efficiently reprogram on-board processing units. Therefore, it is beneficial to develop a compact and flexible development platform.

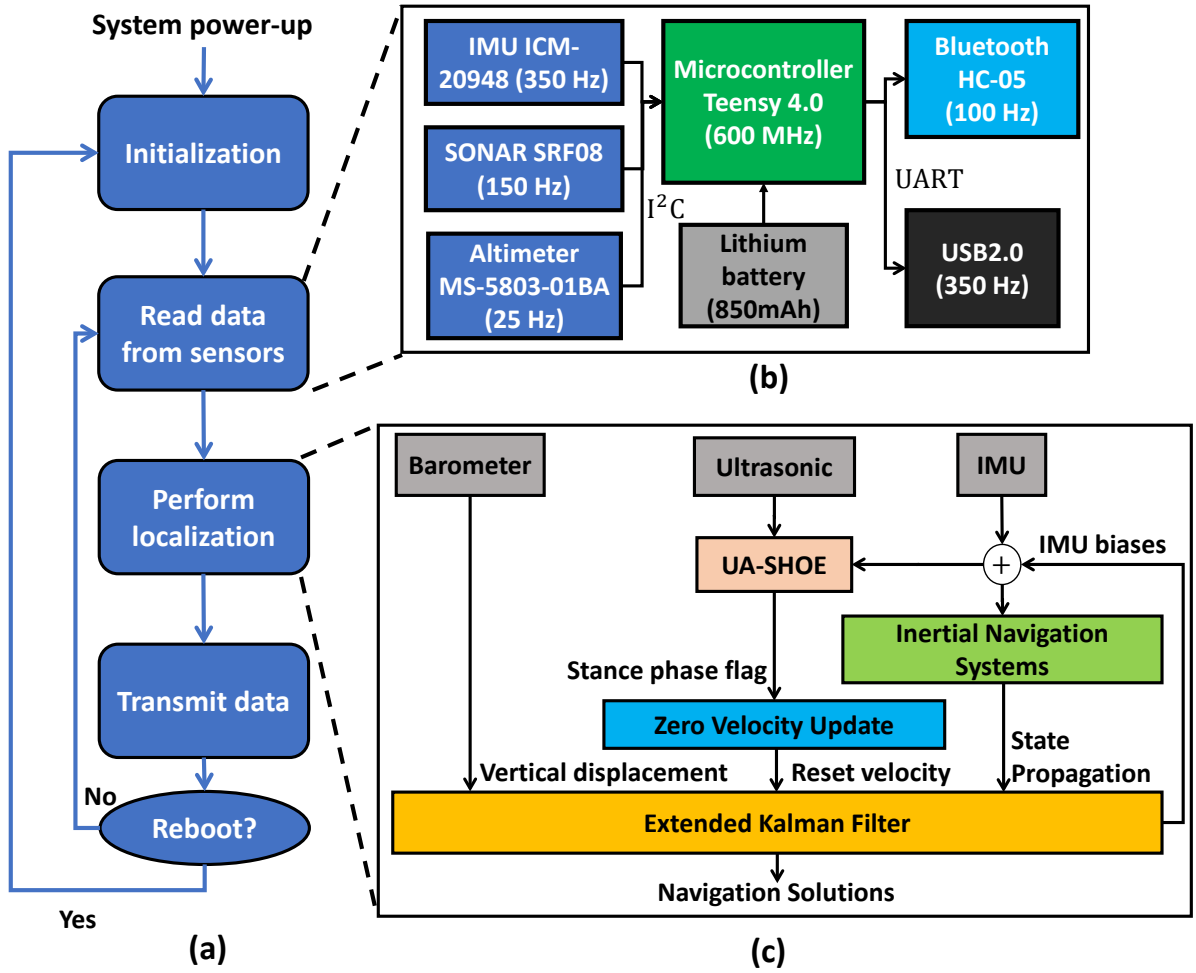


Figure 7.15: (a) The runtime framework of the Sugar-Cube platform. (b) Sensor connection and communication mechanism on the Sugar-Cube platform. (c) Navigation algorithm implemented on the on-board micro-controller.1

This section introduces a flexible and compact multi-sensor hardware testbed, referred to as the Sugar-Cube navigation platform. The Sugar-Cube platform includes a micro-controller, Teensy 4.0, to collect measurements from an IMU, a barometer, and an ultrasonic sensor and produce localization of a user in real-time based on the altimeter-enhanced ZUPT-aided INS

using the UA-SHOE detector. The platform provides a flexible plug-and-play architecture for hardware and software development.

7.5.1 The Sugar-Cube Navigation Platform

Navigation Algorithm

The real-time navigation solution implemented on the current Sugar-Cube platform is the ZUPT-aided INS enhanced by a barometric altimeter with the UA-SHOE detector. Figure 7.15(c) exhibits a block diagram of the algorithm, which is realized in an EKF framework. A detailed discussion of the propagation step and the update step can be found in [91]. The stance phase detector used in this work is the UA-SHOE detector. Comprehensive derivation of the detector is presented in [93]. Table 7.5 lists values of EKF parameters, including ARW σ_g, VRW σ_a, RRW σ_r, AcRW σ_{Ac}, and threshold γ_h used for the UA-SHOE.

Table 7.5: Parameters for the EKF

Hyper-parameter	**Value**
σ_g	2.7221×10^{-5}
σ_a	0.0017
σ_r	8.3174×10^{-7}
σ_{Ac}	6.63×10^{-6}
γ_h	e^{-12}

Hardware System Implementation

A hardware prototype of the Sugar-Cube navigation platform is illustrated in Figure 7.16. The black box in Figure 7.16 is a fixture for placing customized PCBs. A micro-controller Teensy 4.0, a consumer-grade 9-axis IMU ICM−20948, a barometric altimeter MS5803−01BA, and a Bluetooth module HC−05 were located on the PCB. A downward-facing ultrasonic

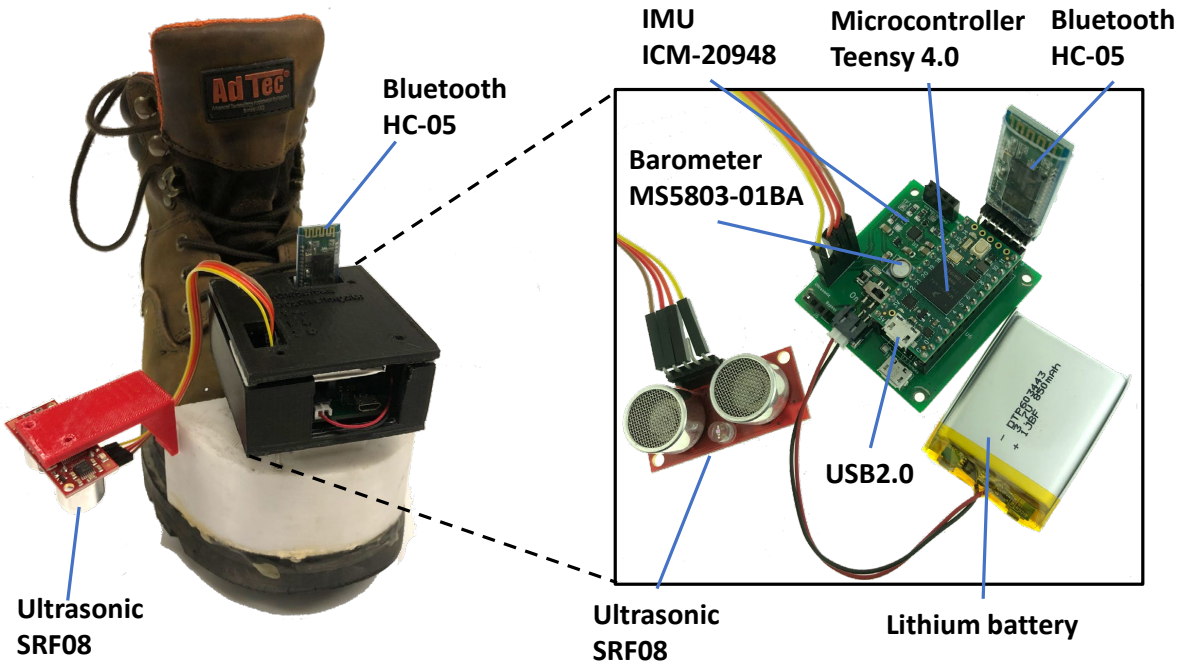


Figure 7.16: Hardware of the Sugar-Cube navigation platform.

sensor SRF08 was firmly attached at the backside of the red extended fixture arm shown in Figure 7.16. The nominal distance between the ultrasonic sensor and the ground was 9 cm. In this configuration, the Sugar-Cube platform was mounted at the toe side, but it can be detached and moved to other locations, such as the heel side. The power source of the Sugar-Cube platform was an 850 mAh lithium battery with a 3.7 v output. The battery was connected to a voltage booster module located on the PCB to bring the voltage up to 5 v.

A block diagram illustrating sensor connection and communication protocols on the Sugar-Cube platform is shown in Figure 7.15(b). The processing unit of the platform is the microcontroller Teensy 4.0, which has a nominal clock rate of 600 MHz and can be boosted up to 1.008 GHz. In this work, the Sugar-Cube platform was programmed with language C/C++ through the Teensyduino library in the Arduino Integrated Development Environment (IDE). Inter-Integrated Circuit (I2C) was used to communicate with the on-board IMU, ultrasonic sensor, and barometer were and had a sampling rate of 350 Hz, 150 Hz, and 25 Hz, respectively. Information, including orientation, velocity, position, zero velocity states, and

sensor readings, can be selectively transmitted to a remote device, such as a smartphone or a computer, via Bluetooth and USB2.0 port with up to 100 Hz and 350 Hz transmission frequency, respectively.

A flow chart describing a runtime framework of the Sugar-Cube platform is presented in Figure 7.15(a). In the initialization process, the Sugar-Cube platform is assumed completely stationary for around 10 s. During this process, all communication protocols are initiated, accelerometer biases are estimated by implementing the ZUPT algorithm, gyroscope biases are calculated by taking the average of the measurements, and initial heading angle of the system is calculated based on the on-board magnetometer. Then, the Sugar-Cube platform enters a sensor data acquisition process. The obtained sensor readings are accessed by the localization module, which produces locations of a user based on the ZUPT-aided INS enhanced by a barometer with the UA-SHOE detector at a rate of 350 Hz. At the end of each iteration, the system waits for user's command to determine whether to continue the next iteration or to re-initiate the entire system.

7.5.2 Real-time Performance Evaluation

To evaluate the navigation performance of the developed Sugar-Cube platform, ten sets of indoor navigation experiments were conducted in the Engineering Gateway Building at the University of California, Irvine. In the experiments, the Sugar-Cube navigation platform was mounted on a pedestrian's right foot, and a Lab-On-Shoe platform was installed on the pedestrian's left shoe. In [18], readers can find a detailed documentation of the Lab-On-Shoe platform, which was a multi-sensor system equipped with near-tactical-grade IMUs. In this work, the Lab-On-Shoe system was used as a benchmark provided by the navigation accuracy. In each experiment, the pedestrian started from a marker placed on the ground, walked a closed-loop trajectory at a speed of 60 step/min on terrains of flat planes, slopes,

and stairs, and returned to the marker at the end of the experiment. The duration of each of the experiments was 1 min, and the length of the trajectories was 50 [m].

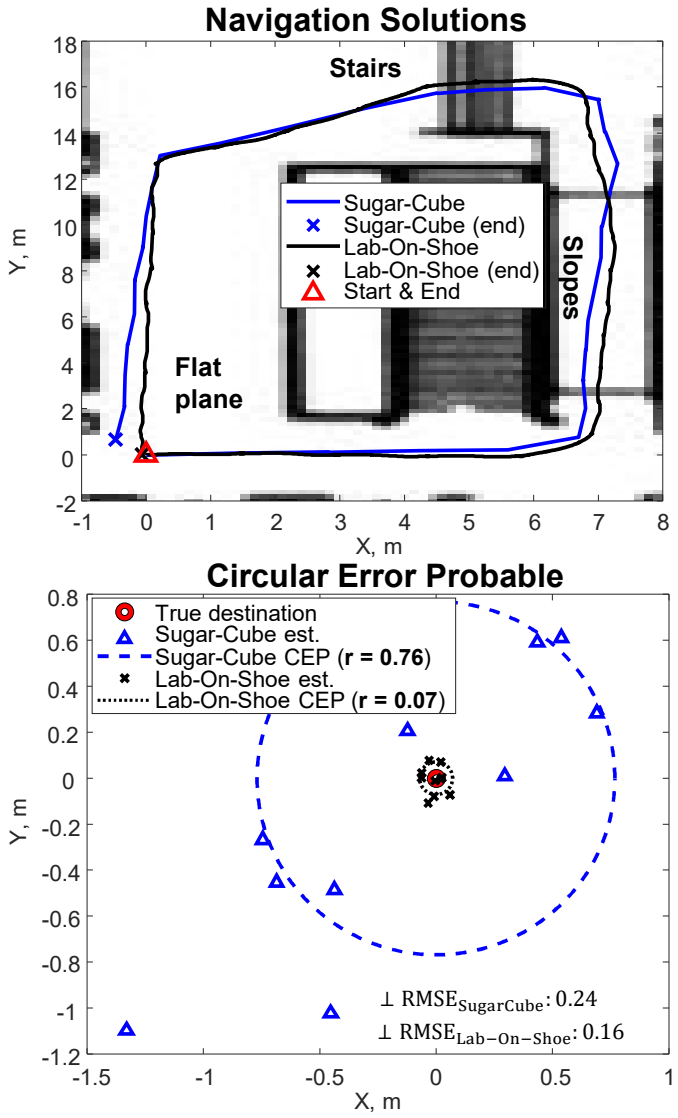


Figure 7.17: The upper plot shows an example of the trajectories estimated by the Sugar-Cube platform and Lab-On-Shoe platform, respectively. The bottom plot shows estimated destination, CEPs, and RMSEs in the ten experiments discussed in Section 7.5.2.

Figure 7.17 presents the experimental results. The Sugar-Cube platform estimated navigation solutions in real-time, and the solutions provided by the Lab-On-Shoe were calculated in a post-processing manner using the same algorithm implemented on the Sugar-Cube platform. The upper plot in Figure 7.17 displays an example of the trajectories estimated from

the two systems overlapped on a floor plan of the building. It could be seen that the real-time solution obtained from the Sugar-Cube platform was able to depict the movement of the pedestrian. The bottom plot presents CEPs, which are circles enclosing 50% of the estimated horizontal destinations in the ten experiments, and RMSEs of the estimated vertical displacements. The CEP's Radius was 0.76 [m] in the case of the Sugar-Cube platform and was 0.07 [m] in the case of the Lab-On-Shoe platform. The vertical RMSEs of the Sugar-Cube and the Lab-On-Shoe platform were 0.24 and 0.16, respectively. In our opinion, the Lab-On-Shoe platform had a higher accuracy mainly because the current configuration of the Sugar-Cube platform used a consumer-grade IMU while the Lab-On-Shoe platform adopted a near-tactical-grade IMU. Based on these experiments, we concluded that the Sugar-Cube platform could collect measurements from an IMU, a barometer, and a downward-facing ultrasonic sensor and perform real-time localization of a pedestrian with respect to an initial location.

This section presented the Sugar-Cube navigation platform, which was developed to be capable of performing real-time localization of a pedestrian based on a ZUPT-aided INS enhanced by an altimeter with the UA-SHOE detector. A series of indoor walking experiments of 60 s were conducted, and the experimental results demonstrated that the real-time navigation results of the Sugar-Cube platform had a horizontal CEP of 0.76 [m] and a vertical RMSE of 0.24 [m]. The result of this section has been published in [84].

7.6 Conclusion

This chapter presents the development of enhancement approaches that improve the positioning accuracy of ZUPT-aided INS along the vertical direction. A closed-form analytical expression was derived to predict the covariance of the ZUPT-aided INS implemented in the EKF along the vertical direction. The analytical expression predicts that the variance of

the vertical displacement is affected by the VRW of an IMU, altimeter sampling rate and resolution, and the ratio of the swing phase and the stance phase during the gait cycle. The developed hybrid altimeter fuses a barometer and a downward-facing ultrasonic sensor in a Multi-Model EKF that simultaneously tracks the heights of both the foot and the ground. It was experimentally demonstrated that the experimental results showed that the RMSE of the ZUPT-aided INS aided by the hybrid altimeter outperformed the RMSE of the standalone ZUPT-aided INS by 91% and the RMSE of the ZUPT-aided INS augmented with a barometer by 41%. A system architecture was developed to demonstrate real-time implementation of the ZUPT-aided INS augmented with a barometric altimeter, and the real-time navigation solutions had a horizontal CEP of 0.76 [m] and a vertical RMSE of 0.24 [m] in a series of experiments involving a pedestrian walking a 50-[m] closed-loop trajectory covering terrains of flat planes, stairs, and ramps.

Chapter 8

On Estimation Filter – Bounding Position Error With Self-Contained Approach

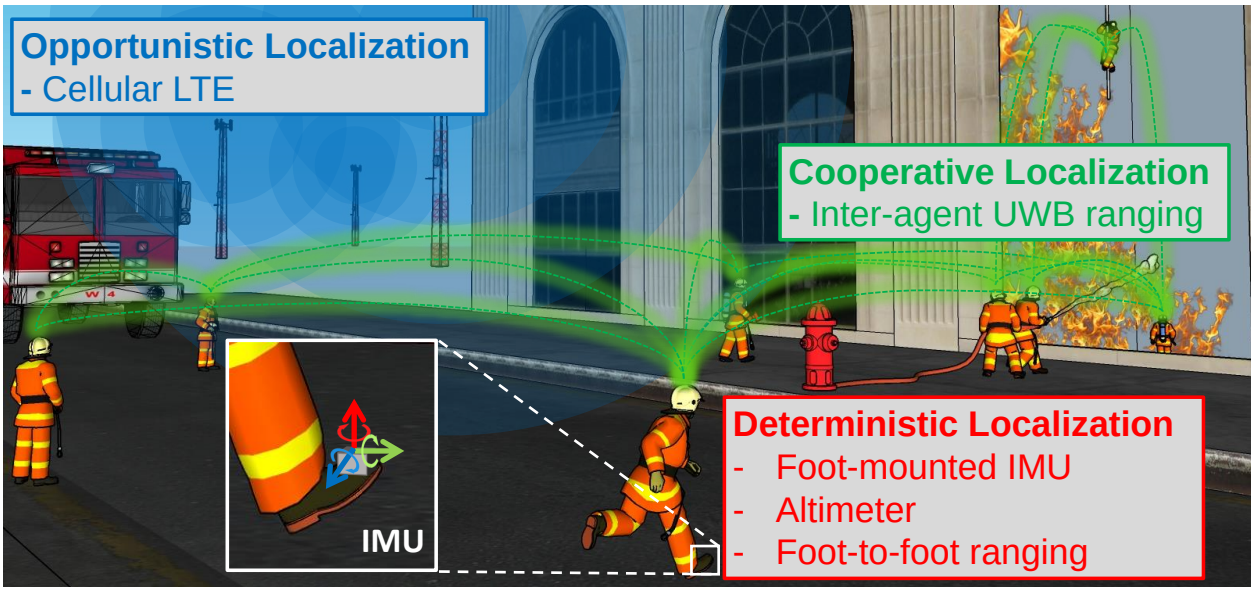


Figure 8.1: Concept of the developed Pedestrian Indoor Navigation System Integrating Deterministic, Opportunistic, and Cooperative Functionalities (PINDOC).

8.1 Introduction

This chapter presents a Pedestrian Indoor Navigation system integrating Deterministic, Opportunistic, and Cooperative functionalities (PINDOC) for navigation of multiple agents. The developed PINDOC can be implemented in different configurations, providing a navigation solution using a different combination of the deterministic, opportunistic, and cooperative functionalities. The deterministic approach utilizes dual foot-mounted IMU and implements ZUPT-aided INS augmented by barometric altimeter and foot-to-foot range measurements. The opportunistic approach uses psuedorange measurements extracted from cellular LTE towers and implements a Deep Neural Network (DNN)-based Synthetic Aperture Navigation (SAN) to spatially mitigate multipath. This approach operates in a base/rover framework in order to tackle one of the main challenges of LTE opportunistic navigation, i.e., the unknown cellular towers' clock states. The cooperative localization approach uses UWBs for inter-agent range measurements and differentiates Line-Of-Sight (LOS) and NLOS components using a power-metric-based detector. The developed PINDOC is implemented with an EKF in a centralized manner.

This chapter is a collaborative effort. The algorithm design, implementation, and experimental hardware associated with the cooperative localization approach, discussed in Section 8.2.2, were carried out by Ph.D. students Changwei Chen and Mingwon Soo under UCI Professor Solmaz Kia's supervision. The DNN-SAN-LTE algorithm and the corresponding experimental setup in the opportunistic approach, discussed in Section 8.2.3, were designed and independently tested by Ali Abdallah with Ohio State University Professor Zak Kassas's guidance. The author of this thesis, Chi-Shih Jao, supervised by UCI professor Andrei Shkel, was responsible for the deterministic localization approach, discussed in Section 8.2.1, as well as the integration of the deterministic, the opportunistic, and the cooperative approaches.

The rest of the chapter is organized as follows. Section 8.2 presents the mathematical model

of the developed approach. Section 8.3 demonstrates a hardware system that realizes the developed algorithm. Section 8.4 discusses validation experiments. Section 8.5 presents concluding remarks.

8.2 Integrating Deterministic, Opportunistic, and Cooperative Functionalities

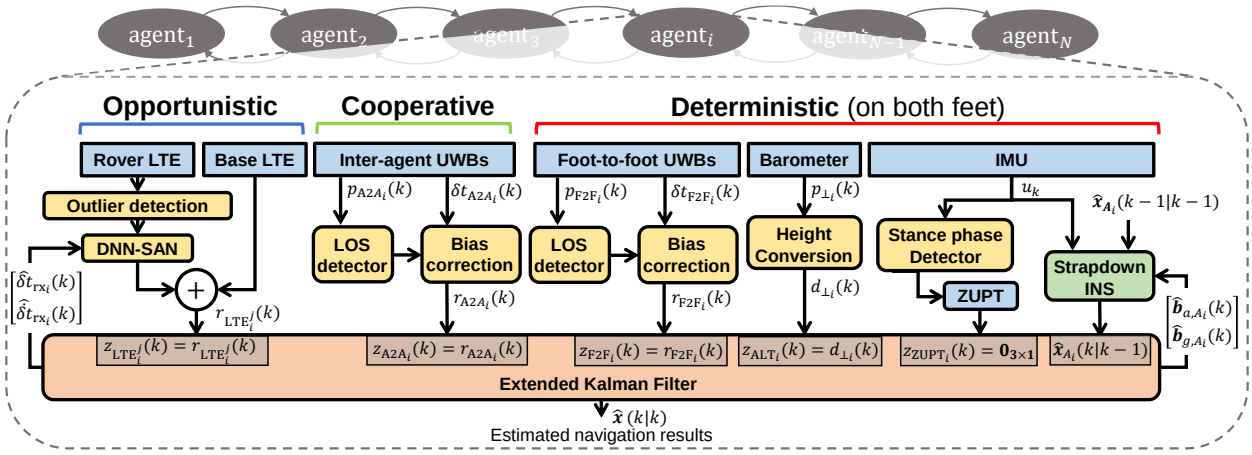


Figure 8.2: Framework for the developed Pedestrian Indoor Navigation system integrating Deterministic, Opportunistic, and Cooperative functionalities (PINDOC). The deterministic module produces navigation solutions for each agent with a Zero-velocity-UPdaTe (ZUPT)-aided Inertial Navigation Systems (INS) augmented with sensing modalities including altimeters and foot-to-foot range measurements. The opportunistic module enhances the deterministic solutions with pseudorange measurements collected based on cellular Long-Term Evolution (LTE) Signal of OPportunity (SOP). Navigation accuracy of each individual agent is further enhanced by cooperative localization using UWB-based inter-agent range measurements.

The developed PINDOC system uses an EKF as the navigation filter. The developed framework is illustrated in Figure 8.2. This section discusses the EKF framework that fuses the deterministic, opportunistic, and cooperative localization approaches used in the PINDOC.

The EKF estimates the state vector $\mathbf{x}(k)$ for a group of N agents, expressed as:

$$\mathbf{x}(k) = [\mathbf{x}_{\mathrm{A}_1}^\top(k), \mathbf{x}_{\mathrm{A}_2}^\top(k), \ldots, \mathbf{x}_{\mathrm{A}_N}^\top(k), \mathbf{x}_{\mathrm{LTE}_1}^\top(k), \mathbf{x}_{\mathrm{LTE}_2}^\top(k), \ldots, \mathbf{x}_{\mathrm{LTE}_N}^\top(k)]^\top,$$

where A_i denotes agent i in the group, and $\mathbf{x}_{\mathrm{A}_i}(k)$ and $\mathbf{x}_{\mathrm{LTE}_i}(k)$ are the states associated with agent i, described as

$$\begin{aligned}
\mathbf{x}_{\mathrm{A}_i}(k) = [&\mathbf{q}_{\mathrm{A}_i^l}^\top(k), \mathbf{v}_{\mathrm{A}_i^l}^\top(k), \mathbf{p}_{\mathrm{A}_i^l}^\top(k), \mathbf{b}_{a,\mathrm{A}_i^l}^\top(k), \mathbf{b}_{g,\mathrm{A}_i^l}^\top(k), \\
&\mathbf{q}_{\mathrm{A}_i^r}^\top(k), \mathbf{v}_{\mathrm{A}_i^r}^\top(k), \mathbf{p}_{\mathrm{A}_i^r}^\top(k), \mathbf{b}_{a,\mathrm{A}_i^r}^\top(k), \mathbf{b}_{g,\mathrm{A}_i^r}^\top(k)]^\top \in \mathbb{R}^{30\times 1} \\
\mathbf{x}_{\mathrm{LTE}_i}(k) = [&c\delta t_{\mathrm{rx}_i}(k), c\dot{\delta} t_{\mathrm{rx}_i}(k)]^\top \in \mathbb{R}^{2\times 1},
\end{aligned}$$

where $\mathbf{q}_{\mathrm{A}_i^l}(k), \mathbf{v}_{\mathrm{A}_i^l}(k), \mathbf{p}_{\mathrm{A}_i^l}(k), \mathbf{b}_{a,\mathrm{A}_i^l}(k)$, and $\mathbf{b}_{g,\mathrm{A}_i^l}(k)$ represent states of the left foot of agent i, including orientations, velocities, positions in the navigation frame, and accelerometer and gyroscope biases in the sensor body frame. $\mathbf{q}_{\mathrm{A}_i^r}(k), \mathbf{v}_{\mathrm{A}_i^r}(k), \mathbf{p}_{\mathrm{A}_i^r}(k), \mathbf{b}_{a,\mathrm{A}_i^r}(k)$, and $\mathbf{b}_{g,\mathrm{A}_i^r}(k)$ are the corresponded states of the right foot agent i. $c\delta t_{\mathrm{rx}_i}(k)$ and $c\dot{\delta} t_{\mathrm{rx}_i}(k)$ indicate the speed of light, c, multiplied by estimated clock bias and drift of an LTE receiver mounted on agent i. This expression of $\mathbf{x}_{\mathrm{LTE}_i}(k)$ has a unit of meter, which is beneficial for numerical calculations, as compared to directly using clock bias and drift.

8.2.1 Deterministic

The deterministic approach is used in the prediction step of the EKF and corrects the states in the update step with sensor measurements acquired from self-contained sensors mounted on the shoes of the same agent. The self-contained sensors include two sets of an IMU, an altimeter, and a UWB. One set of sensors is mounted on each foot of the agent.

Strapdown Inertial Navigation Systems

In the prediction step, the state propagation is implemented by inputting the IMU measurements on each foot to the strapdown inertial navigation systems [198]. The linearized state transition matrix corresponding to strapdown INS, denoted as $\mathbf{F}_{\text{INS}}(k)$, is expressed as follows:

$$\mathbf{F}_{\text{INS}}(k) = e^{\mathbf{A}_{\text{INS}}(t_k)dt},$$

where dt is the sampling rate of the system and

$$\mathbf{A}_{\text{INS}}(t) = \begin{bmatrix} \mathbf{A}_{\text{A}_1}(t) & \mathbf{0}_{30\times30} & \dots & \mathbf{0}_{30\times30} \\ \mathbf{0}_{30\times30} & \mathbf{A}_{\text{A}_2}(t) & & \vdots \\ \vdots & & \ddots & \mathbf{0}_{30\times30} \\ \mathbf{0}_{30\times30} & \dots & \mathbf{0}_{30\times30} & \mathbf{A}_{\text{A}_N}(t) \end{bmatrix},$$

with

$$\mathbf{A}_{\text{A}_i}(t) = \begin{bmatrix} \mathbf{A}_{\text{A}_i^l}(t) & \mathbf{0}_{15\times15} \\ \mathbf{0}_{15\times15} & \mathbf{A}_{\text{A}_i^r}(t) \end{bmatrix},$$

and

$$\mathbf{A}_{\text{A}_i^l}(t) = \begin{bmatrix} \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & -\text{C}(\mathbf{q}_{\text{A}_i^l}(k)) & \mathbf{0}_{3\times3} \\ [\overrightarrow{f_l}^n\times] & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \text{C}(\mathbf{q}_{\text{A}_i^l}(k)) \\ \mathbf{0}_{3\times3} & \mathbf{I}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \end{bmatrix}.$$

Here, $[\overrightarrow{fl}^n\times]$ is the skew-symmetric cross-product-operator of the accelerometer outputs of the left IMU, expressed in the navigation frame. C($\mathbf{q}$) is the DCM corresponding to the quaternion vector $\mathbf{q}$. $\mathbf{0}_{n\times m}$ indicates a zero matrix having n number of rows and m number of columns. $\mathbf{A}_{\mathrm{A}_i^r}(t)$ is constructed in the same manner as $\mathbf{A}_{\mathrm{A}_i^l}(t)$ except that the states corresponding to the right foot are used.

The process noise matrix corresponding to strapdown INS, denoted as $\mathbf{Q}_{\mathrm{INS}}(k)$, is expressed as

$$\mathbf{Q}_{\mathrm{INS}}(k) = \begin{bmatrix} \mathbf{Q}_{\mathrm{A}_1}(k) & \mathbf{0}_{30\times30} & \dots & \mathbf{0}_{30\times30} \\ \mathbf{0}_{30\times30} & \mathbf{Q}_{\mathrm{A}_2}(k) & & \vdots \\ \vdots & & \ddots & \mathbf{0}_{30\times30} \\ \mathbf{0}_{30\times30} & \dots & \mathbf{0}_{30\times30} & \mathbf{Q}_{\mathrm{A}_N}(k) \end{bmatrix},$$

$$\mathbf{Q}_{\mathrm{A}_i}(k) = \begin{bmatrix} \mathbf{Q}_{\mathrm{A}_i^l}(k) & \mathbf{0}_{15\times15} \\ \mathbf{0}_{15\times15} & \mathbf{Q}_{\mathrm{A}_i^r}(k) \end{bmatrix},$$

with $\mathbf{Q}_{\mathrm{A}_i^l}(k) =$

$$\begin{bmatrix} \sigma^2_{\mathrm{ARW}}\mathbf{I}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \sigma^2_{\mathrm{VRW}}\mathbf{I}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{I}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \sigma^2_{\mathrm{AcRW}}\mathbf{I}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \sigma^2_{\mathrm{RRW}}\mathbf{I}_{3\times3} \end{bmatrix}.$$

Here, $\mathbf{I}_{n\times n}$ is the identity matrix having n number of rows and columns. $\sigma^2_{\mathrm{ARW}_i}$, $\sigma^2_{\mathrm{VRW}_i}$, $\sigma^2_{\mathrm{RRW}_i}$, and $\sigma^2_{\mathrm{AcRW}_i}$ are the ARW of the gyroscopes, the VRW of the accelerometers, the RRW of the gyroscopes, and the AcRW of the accelerometers of the IMU mounted on the left shoe of agent i. This section set $\mathbf{Q}_{\mathrm{A}_i^r}(k) = \mathbf{Q}_{\mathrm{A}_i^l}(k)$ based on an assumption that two IMUs, each mounted on the same agent's left and right feet, have similar noise characteristics.

Zero-velocity Update

When a stance phase is detected, the ZUPT algorithm is activated to compensate for the velocity state in the update step of the EKF. The compensation is done by feeding in pseudo-measurements of zero velocity along the three axes, which is denoted as $\mathbf{v}_{\mathrm{ZUPT}}(k) = \mathbf{0}_{3\times 1}$. In this section, the stance phase detection is achieved with the SHOE detector [181]. For the states associated with the left and the right feet of agent i, the ZUPT measurement models, $\mathbf{z}_{\mathrm{ZUPT}_i^l}(k)$ and $\mathbf{z}_{\mathrm{ZUPT}_i^r}(k)$, measurement matrices, $\mathbf{H}_{\mathrm{ZUPT}_i^l}(k)$ and $\mathbf{H}_{\mathrm{ZUPT}_i^r}(k)$, and measurement covariance matrices, $\mathbf{R}_{\mathrm{ZUPT}_i^l}(k)$ and $\mathbf{R}_{\mathrm{ZUPT}_i^r}(k)$, are expressed as follows:

$$
\begin{aligned}
&\mathbf{z}_{\mathrm{ZUPT}_i^l}(k) = \mathbf{z}_{\mathrm{ZUPT}_i^r}(k) = \mathbf{v}_{\mathrm{ZUPT}}(k) \\
&\mathbf{H}_{\mathrm{ZUPT}_i^l}(k) = \begin{bmatrix} \mathbf{0}_{((i-1)\times 30+3)\times 3} \\ \mathbf{I}_{3\times 3} \\ \mathbf{0}_{(24+(N-i)\times 30+2N)\times 3} \end{bmatrix}^\top \\
&\mathbf{H}_{\mathrm{ZUPT}_i^r}(k) = \begin{bmatrix} \mathbf{0}_{((i-1)\times 30+18)\times 3} \\ \mathbf{I}_{3\times 3} \\ \mathbf{0}_{(9+(N-i)\times 30+2N)\times 3} \end{bmatrix}^\top \\
&\mathbf{R}_{\mathrm{ZUPT}_i^l}(k) = \mathbf{R}_{\mathrm{ZUPT}_i^r}(k) = \sigma^2_{\mathrm{ZUPT}_i}\mathbf{I}_{3\times 3},
\end{aligned}
$$

where $\sigma^2_{\mathrm{ZUPT}_i}$ is the noise variance of the zero-velocity measurement $\mathbf{v}_{\mathrm{ZUPT}}$ for agent i.

Altimeter Measurements

In pedestrian navigation, altitude measurements can be obtained from a barometer [91] or, in a hybrid approach, using both barometer and ultrasonic sensors [92]. At time k, altimeters mounted on the left and the right shoes of agent i provide measurements of vertical displacements in the navigation frame, which are denoted as $d_{\perp_i^l}(k)$ and $d_{\perp_i^r}(k)$,

respectively. The altimeter measurements are used in the update step of the EKF to bound error growth of estimated position along the vertical direction[91]. The measurement models corresponding to the altimeter on the left and right feet of agent i, $z_{\text{ALT}_i^l}(k)$ and $z_{\text{ALT}_i^r}(k)$, are described as follows:

$$z_{\text{ALT}_i^l}(k) = d_{\perp_i^l}(k), z_{\text{ALT}_i^r}(k) = d_{\perp_i^r}(k).$$

The associated measurement matrices are described as

$$\mathbf{H}_{\text{ALT}_i^l}(k) = \begin{bmatrix} \mathbf{0}_{1\times((i-1)\times 30+8)} & 1 & \mathbf{0}_{1\times(21+(N-i)\times 30+2N)} \end{bmatrix}$$
$$\mathbf{H}_{\text{ALT}_i^r}(k) = \begin{bmatrix} \mathbf{0}_{1\times((i-1)\times 30+23)} & 1 & \mathbf{0}_{1\times(6+(N-i)\times 30+2N)} \end{bmatrix}.$$

The measurement noise covariance matrices are described as

$$\mathbf{R}_{\text{ALT}_i^l}(k) = \mathbf{R}_{\text{ALT}_i^r}(k) = \sigma^2_{\text{ALT}_i},$$

where $\sigma^2_{\text{ALT}_i}$ is the noise variance of the altimeter measurements for agent i.

Foot-to-foot Ranging Measurements

Distance measurements between the two feet of agent i, denoted as $r_{\text{F2F}_i}(k)$, can be obtained from various different sensing modalities, including ultrasonic sensors [213], foot-to-foot cameras [94], electromagnetic systems [222], and UWB [263, 31]. In this section, UWB-based foot-to-foot ranging measurements are used. The foot-to-foot range measurements are classified into LOS and NLOS by a power metric-based approach [264]. In this section, only LOS UWB measurements are used. The range measurements are processed with bias correction. The processed foot-to-foot measurements are fused in the update step of the EKF to compensate for relative distances between the two feet [117]. The corresponding measurement

model, $z_{\mathrm{F2F}_i}(k)$, and measurement matrix, $\mathbf{H}_{\mathrm{F2F}_i}(k)$, are described as follows:

$$z_{\mathrm{F2F}_i}(k) = r_{\mathrm{F2F}_i}(k)$$

$$\mathbf{H}_{\mathrm{F2F}_i}(k) = \begin{bmatrix} \mathbf{0}_{((i-1)\times 30+6)\times 1} \\ \frac{\partial ||\mathbf{p}_{\mathrm{A}_i^l}(k) - \mathbf{p}_{\mathrm{A}_i^r}(k)||^\top}{\partial \mathbf{p}_{\mathrm{A}_i^l}} \\ \mathbf{0}_{15\times 1} \\ \frac{\partial ||\mathbf{p}_{\mathrm{A}_i^l(k)} - \mathbf{p}_{\mathrm{A}_i^r(k)}||^\top}{\partial \mathbf{p}_{\mathrm{A}_i^r(k)}} \\ \mathbf{0}_{(6+(N-i)\times 30+2N)\times 1} \end{bmatrix}^\top.$$

The measurement noise covariance matrices are described as

$$\mathbf{R}_{\mathrm{F2F}_i}(k) = \sigma^2_{\mathrm{F2F}_i},$$

where $\sigma^2_{\mathrm{F2F}_i}$ is the noise variance of the foot-to-foot range measurements for agent i.

8.2.2 Cooperative

The cooperative localization approach is realized through inter-agent ranging measurements obtained from UWB sensors attached to the right shoe of each agent. Similar to the foot-to-foot range measurements, the measurements between agent i and agent h, denoted as $r_{\mathrm{A2A}_i^h}(k)$, are classified into LOS and NLOS cases, and only LOS cases are used in the update step of the EKF. The LOS measurements are further processed with bias correction. Details of the implementation of the cooperative localization and the classification of LOS and NLOS are documented in [261] and [262]. Assuming that $i < h$, the corresponding inter-agent range measurement model, $z_{\mathrm{A2A}_i^h}(k)$, and measurement matrix, $\mathbf{H}_{\mathrm{F2F}_i^h}(k)$, are

described as follows:

$$z_{\mathrm{A2A}_i^h}(k) = r_{\mathrm{A2A}_i^h}(k)$$

$$\mathbf{H}_{\mathrm{A2A}_i^h}(k) = \begin{bmatrix} \mathbf{0}_{((i-1)\times 30+21)\times 1} \\ \frac{\partial ||\mathbf{p}_{\mathrm{A}_i^r}(k) - \mathbf{p}_{\mathrm{A}_h^r}(k)||^\top}{\partial \mathbf{p}_{\mathrm{A}_i^r}(k)} \\ \mathbf{0}_{(6+(h-i-1)\times 30+21)\times 1} \\ \frac{\partial ||\mathbf{p}_{\mathrm{A}_i^r,k} - \mathbf{p}_{\mathrm{A}_h^r,k}||^\top}{\partial \mathbf{p}_{\mathrm{A}_h^r,k}} \\ \mathbf{0}_{(6+(N-h)\times 30+2N)\times 1} \end{bmatrix}^\top.$$

The measurement noise covariance matrices are described as

$$\mathbf{R}_{\mathrm{A2A}_i^h}(k) = \sigma^2_{\mathrm{A2A}_i^h},$$

where $\sigma^2_{\mathrm{A2A}_i^h}$ is the noise variance of the inter-agent range measurements between agent i and agent h.

8.2.3 Opportunistic

In the developed framework, cellular LTE signals are utilized to provide absolute positioning measurements. This is achieved by exploiting LTE downlink signals opportunistically in a base/rover LTE-DNN-SAN framework. Details of the LTE-DNN-SAN framework were documented in [2]. In this developed framework, a "base" LTE receiver is located outside the building and has access to GNSS signals. The base collects signals from multiple LTE towers (also known as eNodeBs) in the environment. The positions of the eNodeBs are pre-surveyed and assumed to be known (e.g., according to [134]). The base receiver estimates the eNodeBs' clock biases and shares this information with the indoor receivers denoted by "rovers." Each rover has a copy of the same LTE receiver used in the base unit; however,

a DNN-based SAN correction block is applied, in which the pedestrian's motion is utilized to synthesize a geometrically-separated antenna array from time-separated snapshots. This allows for beamforming towards the LOS from the rover to the LTE eNodeB, while suppressing multipath components. This process requires obtaining the LOS steering vector, which is obtained by taking the nearest Direction-of-Arrival (DoA) estimate from the developed DNN-DoA estimator to the LOS DoA estimated using the current estimate of the rover's position and the known LTE eNodeB positions. Figure 8.3 depicts the block diagram of the LTE-DNN-SAN framework.

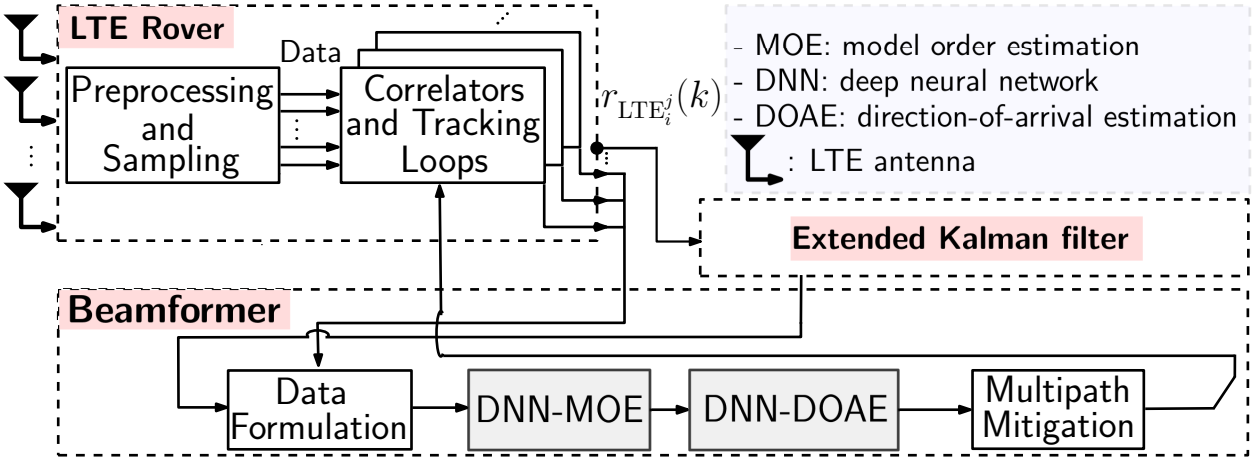


Figure 8.3: A block diagram depicting the LTE-DNN-SAN block diagram used in the developed PINDOC framework shown in Figure 8.2.

The EKF prediction step propagates the state $\mathbf{x}_{\mathrm{LTE}_i}(k)$ with following state transition matrix $\mathbf{F}_{\mathrm{LTE}_i}(k)$ and process noise covariance $\mathbf{Q}_{\mathrm{LTE}_i}(k)$

$$\mathbf{F}_{\mathrm{LTE}_i}(k) = \begin{bmatrix} 1 & dt \\ 0 & 1 \end{bmatrix}$$

$$\mathbf{Q}_{\mathrm{LTE}_i}(k) = c^2 \begin{bmatrix} \sigma_{\delta t_{rx_i}} dt + \sigma_{\dot{\delta} t_{rx_i}} \frac{dt^3}{3} & \sigma_{\dot{\delta} t_{rx_i}} \frac{dt^2}{2} \\ \sigma_{\dot{\delta} t_{rx_i}} \frac{dt^2}{2} & \sigma_{\dot{\delta} t_{rx_i}} dt, \end{bmatrix}$$

where $\sigma_{\delta t_{rx_i}}$ and $\sigma_{\dot{\delta} t_{rx_i}}$ are parameters associated with clock quality. Details regarding modeling of the clock bias and drift are documented in [3].

In the update step of the EKF, LTE psuedorange measurements are fused in a tightly-coupled manner with the deterministic and the cooperative approaches. For LTE signals that are transmitted from eNodeB j and received by LTE receiver on agent i, the associated pseudorange measurement is denoted as $r_{\mathrm{LTE}_i^j}(k)$. The location of eNodeB j is represented by $\mathbf{p}_{\mathrm{eNodeB}_j}$. The corresponding measurement model, $z_{\mathrm{LTE}_i^j}(k)$, measurement matrix, $\mathbf{H}_{\mathrm{LTE}_i^j}(k)$, and measurement covariance matrix, $\mathbf{R}_{\mathrm{LTE}_i^j}(k)$, are described as follows:

$$
\begin{aligned}
&z_{\mathrm{LTE}_i^j}(k) = r_{\mathrm{LTE}_i^j}(k) \\
&\mathbf{H}_{\mathrm{LTE}_i^j}(k) = \begin{bmatrix} \mathbf{0}_{((i-1)\times 30+21)\times 1} \\ \frac{\partial ||\mathbf{p}_{\mathrm{A}_i^r}(k) - \mathbf{p}_{\mathrm{eNodeB}_j}||^\top}{\partial \mathbf{p}_{\mathrm{A}_i^r}(k)} \\ \mathbf{0}_{(6+(N-i)\times 30)\times 1} \\ \mathbf{0}_{((i-1)\times 2)\times 1} \\ 1 \\ \mathbf{0}_{(1+(N-i)\times 2)\times 1} \end{bmatrix}^\top \\
&\mathbf{R}_{\mathrm{LTE}_i^j}(k) = \sigma^2_{\mathrm{LTE}_i^j}.
\end{aligned}
$$

Here, $\sigma^2_{\mathrm{LTE}_i^j}$ is an adaptive value based on $c^2 \frac{\alpha}{(C/N_0)_i^j}$, where $(C/N_0)_i^j$ is the signal-to-noise ratios of the pseudorange measurement $r_{\mathrm{LTE}_i^j}(k)$ and α is a tuning parameter that was chosen to be 2.22×10^{-11} [3]. To detect and remove outliers from LTE observables, a rudimentary innovation-based detector is implemented to filter out inconsistent LTE observables [68].

8.2.4 The EKF for PINDOC

The developed PINDOC combines the deterministic, opportunistic, and cooperative localization approaches in the EKF. When each of the sensing modalities mentioned previously becomes available, the EKF stacks all available measurements and performs the update step. Settings of the noise parameters are determined based on noise characteristics of sensors in-

volved in the implementation.

8.3 System Hardware

This section discusses experimental hardware, shown in Figure 8.4, that is designed to evaluate navigation performance of the developed PINDOC.

8.3.1 Lab-On-Shoe Platform

The Lab-On-Shoe platform, shown in Figure 8.4, is responsible for acquisition of all sensor measurements, except for foot-to-foot ranges, that are associated with the deterministic localization. The Lab-On-Shoe was developed in Microsystems Lab at the University of California, Irvine, as a reconfigurable multi-sensor pedestrian navigation testbed [18]. The agent wears the Lab-On-Shoe platform on both left and right feet. Each shoe of the platform includes an Analog Device ADIS16497−3 tactical-grade IMU, an MS5803−01BA barometric altimeter, an SRF08 ultrasonic sensor, and two SRF02 ultrasonic sensors. The barometer has a nominal resolution of 10 cm in vertical displacement measurement, and the ultrasonic sensor has a range resolution of 1 cm. In this section, the ultrasonic sensors were not used in the experiment discussed in Section 8.4. A microcontroller Teensy 3.2 is used to implement digital communication protocols, including the I2C and SPI, to collect sensor measurements on the Lab-On-Shoe platform. The sampling rate of IMUs and altimeters are 1000 Hz and 20 Hz, respectively. The collected measurements are transmitted to a laptop with the Universal Asynchronous Receiver-Transmitter (UART) through a USB cable for data logging.

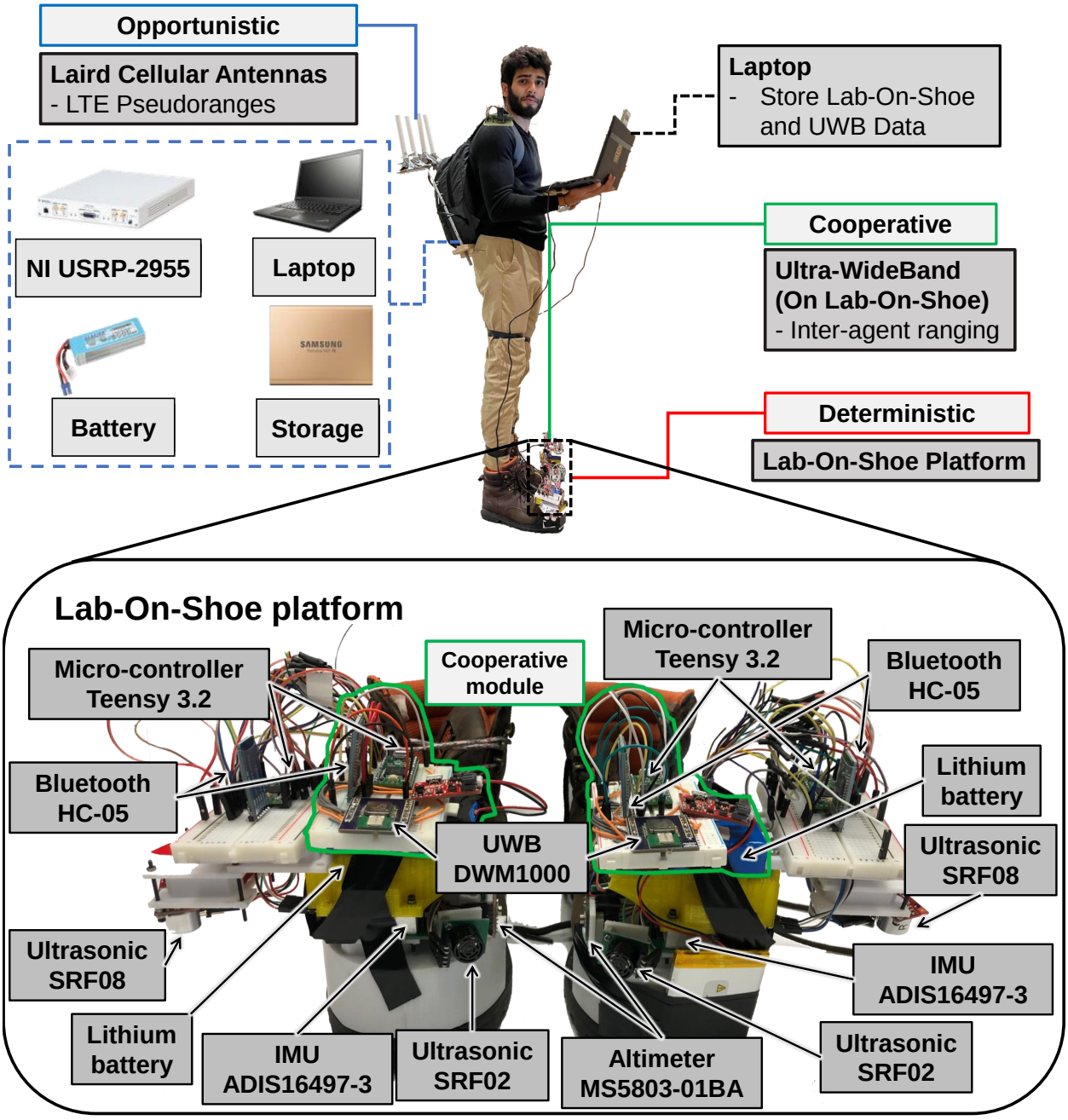


Figure 8.4: Experimental setup used for investigation of the navigation performance of the developed PINDOC. The deterministic localization approach was realized with the Lab-On-Shoe platform, which integrated sensing modalities including IMUs, altimeters, and ultrasonic sensors. In this section, only IMU and altimeters were used. The opportunistic LTE-based pseudoranges were collected by the Laird cellular Antennas and their corresponding signal processing units. The cooperative modules mounted on the Lab-On-Shoe platform included UWBs for foot-to-foot ranging and inter-agent ranging measurements. The laptop was used in the experiment for data logging.

8.3.2 LTE Receivers and Processing Modules

LTE signals are used to realize the opportunistic approach for the developed PINDOC. Each agent in the PINDOC carries a backpack where an LTE receiver is mounted and contains an LTE receiver, a laptop, a battery, and a storage hard drive. The LTE receiver, developed at the Autonomous Systems Perception, Intelligence, & Navigation (ASPIN) Laboratory [3], is equipped with four consumer-grade cellular omni-directional Laird antennas, and a quad-channel National Instruments (NI) Universal Software Radio Peripheral (USRP)−2955 is used to simultaneously down-mix and synchronously sample LTE signals at 10 Megasamples per second (Msps). The sampled LTE signals are transferred from the USRP−2955 via a PCI Express cable and stored on a laptop for post-processing. The LTE measurements have a sampling rate of 100 Hz.

8.3.3 Cooperative UWB Modules

One cooperative module, shown in Figure 8.4, is mounted on each shoe of the Lab-On-Shoe platform. Each of the modules includes a UWB DWM1000, a microcontroller Teensy 3.2, a Bluetooth device HC−05, and a lithium battery [262]. The microcontroller running at a clock rate of 120 Mhz implements an UWB protocol to communicate with the UWB, and the battery provides a power source for the entire module. The module on the left shoe of an agent is paired up with the modules located on the right foot of all other agents and obtains three pieces of information: range, Power Metric (PM), and agent identification (ID). PM is the difference between the total received signal power and the direct-path signal power and is used for LOS/NLOS detection. The range measurements are used for foot-to-foot ranging when obtained from two UWBs mounted on the same agent and for inter-agent ranging when collected from the sensors mounted on two different agents. The collected measurements, including range, PM, and agent ID, are transmitted to nearby Teensy 3.2 on

the Lab-On-Shoe platform in UART communication protocol via the Bluetooth transmitter. The sampling rate of the UWB range measurements is 10 Hz.

The developed pedestrian navigation testbed shown in Figure 8.4 was designed as a research prototype to quickly and flexibly investigate different localization accuracy of algorithms for pedestrian indoor navigation systems. In Figure 8.4, the hardware mounted on each shoe of the Lab-On-Shoe systems, the LTE backpack, and handheld laptop weighted 237 g, 5.44 kg, and 1.53 kg, respectively. It has been demonstrated that it is possible to adapt this pedestrian navigation testbed into a minimized wearable device for practical, real-life applications. For the deterministic module, foot-mounted IMU systems, such as the Sugar-Cube platform [84] and the OpenShoe module[144], with small form factors and real-time positioning functionality have been developed; For the cooperative module, systems like the TRX system [236] are commercially available. For the opportunistic module, it can be implemented on a modern smartphone with the limitation of the number of available RF Front-Ends and the available resources from the cellular modem. For implementing the full capability of opportunistic navigation, hardware (i.e., cellular modem) modifications will be required.

8.4 Experiment Validation

To evaluate navigation performance of the developed PINDOC, we conducted two series of multi-agent pedestrian navigation experiments in an indoor environment at the Engineering Gateway Building at the University of California, Irvine. This section describes the experiment, presents the experimental results, and discusses the performance of the PINDOC.

8.4.1 Performance Metrics

This section considered seven performance metrics, including one computational complexity metric and six different accuracy metrics, to evaluate the navigation performance of the PINDOC. These metrics, listed in Table 8.3, are processing time, position RMSE, two-dimensional (2D) RMSE, vertical ($\perp$) RMSE, position error SD, maximum displacement error, and final position error. In this section, processing time was used to evaluate the computational complexity of a localization solution. The processing time is calculated based on the amount of time for the 2021a MATLAB program, operating on a laptop with an AMD Ryzen 9 5900HS Central Processing Unit (CPU) running at a clock rate of around 4 GHz, to compute a navigation solution based on collected sensor measurements.

The six accuracy metrics were chosen so that the navigation performance of the developed PINDOC can be conveniently compared with the localization accuracy of other indoor navigation systems developed in the literature. Among these six metrics, RMSE is often used to evaluate an estimated localization solution when reference trajectories along the three dimensions are available. 2D RMSE and $\perp$ RMSE are used as benchmarks when estimated positioning solutions emphasize accuracy in the horizontal and the vertical directions, respectively. Position error SD is used to quantify variations of displacement error. Maximum displacement error is used to investigate the worst-case scenarios of an estimated position. Finally, final position error is often used to evaluate dead-reckoning systems, which have localization errors accumulating with time, in navigation experiments where obtaining reference trajectories is challenging. For example, in many pedestrian navigation experiments involving using foot-mounted IMUs [1], the trajectories could cover large indoor areas, on the order of 50$-$100 [m], and include a combination of complex terrains, such as flat planes, stairs, ramps, ladders, and elevators. Deploying a high-precision position reference system like the Opti-Track [250] or the Vicon [15] in such environments can be very expensive, and therefore, final position errors are used in these scenarios.

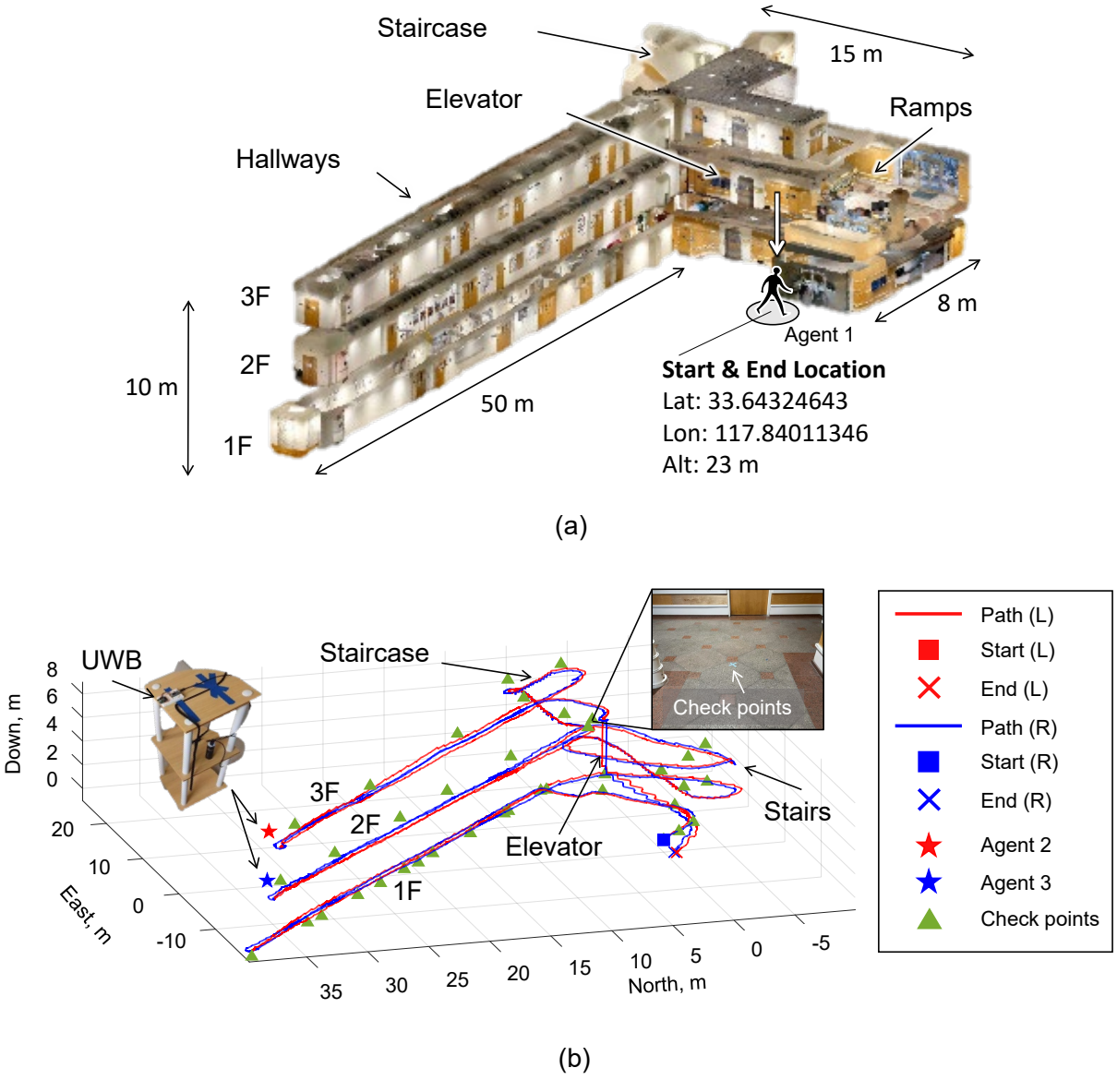


Figure 8.5: (a) Point cloud map of the experimental scenario generated with LiDAR and camera modules installed on iPhone 12 Max Pro. (b) The blue and the red curves represent navigation solutions of the two feet of agent No.1 estimated by the developed PINDOC implementing the ZUPT-aided INS augmented by altimeters, foot-to-foot ranging, and inter-agent ranging measurements in the experiment discussed in Section 8.4. The blue star and the red star marked the locations of stationary agent No.2 and agent No.3 in the navigation frame. The green triangles represent checkpoints that were used to evaluate the in-trajectory localization performance of the navigation solutions.

Table 8.1: LTE ENodeBs' Charactestics

eNodeB	Carrier frequency [MHz]	N_{ID}^{Cell}	Bandwidth [MHz]	Cellular provider
1	2125	223	20	Verizon
2	1955	11	20	AT&T
3	2145	441	20	T-Mobile
4	2112.5	401	20	AT&T

8.4.2 Experiment #1: One Moving Agent, Two Stationary Agents

Experiment Descriptions

The experiment involved three agents. Agent No.1, shown in Figure 8.4, was equipped with the deterministic, opportunistic, and cooperative hardware. Agent No.2 and agent No.3 were both equipped with a set of the cooperative module. Figure 8.5(a) illustrates the experimental scenario, which is presented with point cloud points collected with a LiDAR module and cameras installed in an iPhone 12 Max. At the beginning of the experiment, agent No.1 stood outside of the first floor of the building for one minute to initialize the system. The GNSS and the altimeter mounted on agent No.1 obtained the initial position in latitude, longitude, and altitude during this period. The LTE receiver was initialized and started to track signals transmitted from four eNodeBs. The eNodeBs' characteristics are summarized in Table 8.1. In the initialization process, accelerometer and gyroscope biases of the IMUs were calibrated. Noise parameters used in the EKF for PINDOC are listed in Table 8.2.

After the initialization step, agent No.1 walked inside the building with a trajectory represented by the blue and the red curves in Figure 8.5(b). The trajectory of the experiment included flat planes, stairs, ramps, and elevators. The length of the path was around 600

Table 8.2: Parameters for the EKF

Hyper-parameter	**Value**
σ_{ARW_1}	2.7221×10^{-5}
σ_{VRW_1}	0.0017
σ_{RRW_1}	8.3174×10^{-7}
σ_{AcRW_1}	6.63×10^{-6}
σ_{ZUPT_1}	0.02
σ_{ALT_1}	0.1
σ_{F2F_1}	0.1
σ_{A2A_1}	1
σ_{LTE_1}	$\sqrt{c^2 \frac{2.22\times10^{-11}}{(C/N_0)}}$
$\sigma_{\delta t_{rx_i}}$	1.3×10^{-22}
$\dot{\sigma}_{\delta t_{rx_i}}$	7.8957×10^{-25}

meters, and the duration was approximately 14 minutes. During the experiment, agent No.1 passed by the checkpoints marked with the green triangles in Figure 8.5(b). The locations of the checkpoints were measured with an industrial ruler. The experiment was recorded using a smartphone camera, and the timestamps at which the agent passed through each checkpoint were visually determined from the video. During the entire experiment, agent No.2 and agent No.3 remained stationary at the location marked by the red and blue star markers in Figure 8.5(b). The UWB sensors mounted on the right shoe of agent No.1 had an LOS connection with agent No.2 from 250 s to 310 s and with agent No.3 from 430 s to 485 s. Communication was lost in other periods of time because the UWB modules were too far apart and due to obstacles between the UWB modules. The measurements collected in this experiment indicate that the maximum communication range for the UWBs was around 30 [m] when there was a clear path for signals to be transmitted.

Experimental Results

This section compares the navigation performance for agent No.1 using the developed PINDOC implemented in different configurations. Different configurations of the PINDOC use the ZUPT-aided INS, augmented by different combinations of various sensing modalities, namely altimeter (ALT), foot-to-foot ranging (F2F), inter-agent ranging with agent No.2 (A2) and agent No.3 (A3), and cellular LTE pseudorange measurements. Trajectories estimated by the different configurations are presented in Figure 8.6.

To quantify the localization error at each checkpoint for each navigation solution, the seven performance metrics discussed in Section 8.4.1 were used. Table 8.3 summarizes the performance of the navigation solutions using the PINDOC implemented in different configurations. The accuracy values presented in Table 8.3 were calculated based on the 38 checkpoints marked by the green triangles in Figure 8.5. For the configurations where foot-to-foot ranging measurements were not involved, the accuracy metrics were calculated based on solutions of agent No.1's right foot as inter-agent range measurements were collected with the UWB module mounted on the right foot. The top item Table 8.3, which is PINDOC with configuration G that uses ZUPT-aided INS augmented by ALT, F2F, A2, A3, and LTE had the smallest RMSE of 0.93 [m]. The bottom item in Table 8.3, which is PINDOC with configuration A, had the largest RMSE of 2.53 [m].

Discussion

The following remarks can be concluded from Table 8.3.

- It could be observed that enhancing the ZUPT-aided INS with more aiding methods led to better navigation accuracy, however, with a trade-off of increasing computational complexity. The smallest displacement error was achieved with configuration

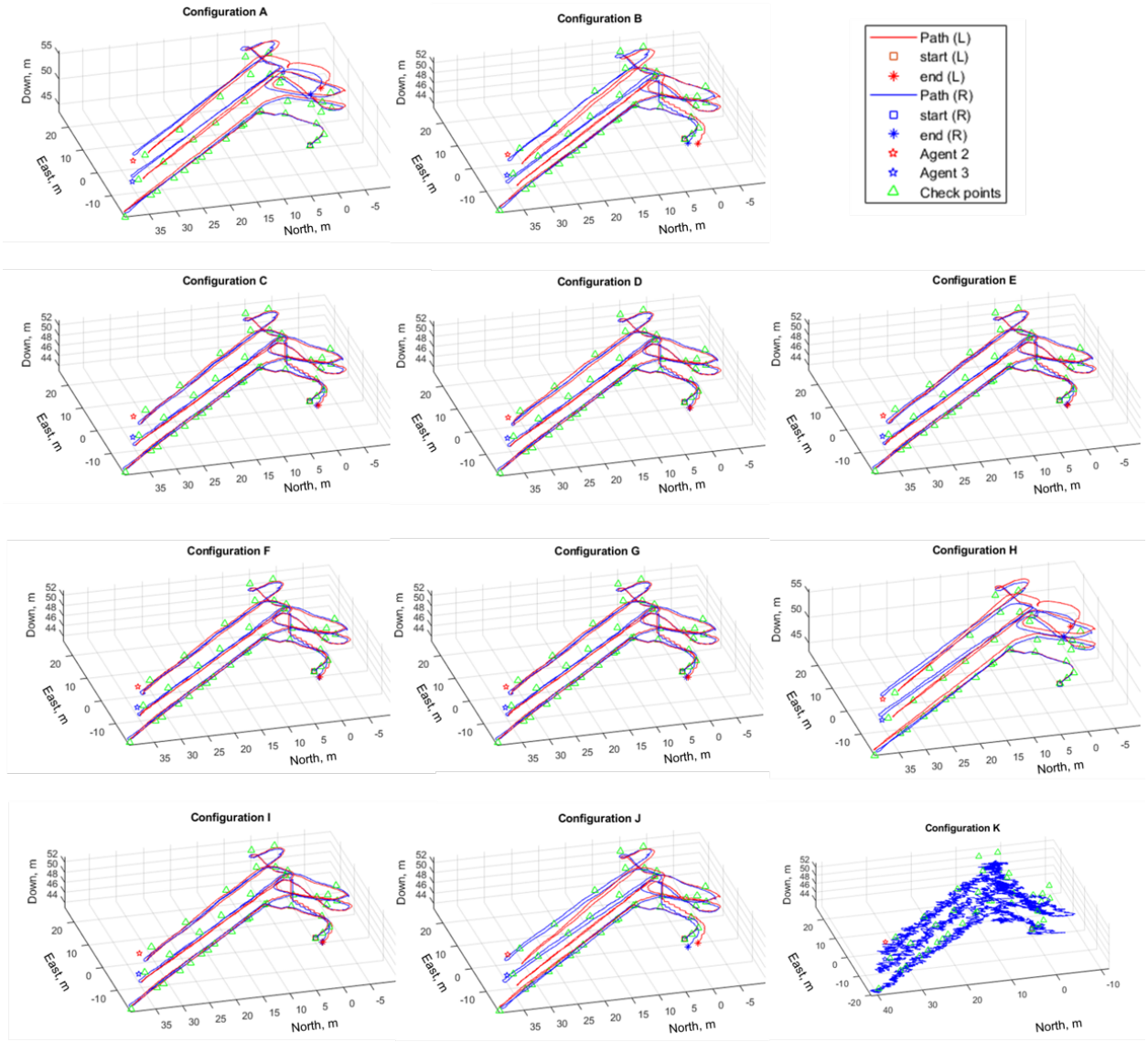


Figure 8.6: Navigation trajectories estimated by different configurations of the developed PINDOC. In the plot of configuration K, only the trajectory associated with Agent 1's right foot was shown because, in the experiment, only the navigation solution of the right foot was augmented with LTE pseudoranges. Error metrics of each of the configurations are documented in TABLE 8.3.

Table 8.3: Navigation Performance of the developed PINDOC implemented in different configurations.

ConFigure	INS Aiding Method						Processing Time (s)	RMSE [m]	2D RMSE [m]	⊥ RMSE [m]	SD [m]	Max Error [m]	Final Error [m]
	ZUPT	ALT	F2F	A2	A3	LTE							
G	✓	✓	✓	✓	✓	✓	231.7	0.93	0.69	0.62	0.44	2.23	1.28
F	✓	✓	✓	✓	✓		211.6	0.93	0.69	0.62	0.44	2.23	1.32
D	✓	✓	✓	✓			211.0	0.94	0.71	0.62	0.43	2.23	1.32
E	✓	✓	✓		✓		210.9	0.95	0.72	0.62	0.46	2.23	1.68
I	✓	✓	✓			✓	227.0	0.96	0.74	0.62	0.45	2.23	1.64
C	✓	✓	✓				210.3	0.97	0.75	0.62	0.46	2.23	1.76
J	✓	✓				✓	234.0	1.07	0.87	0.62	0.48	2.44	1.51
B	✓	✓					207.4	1.64	1.52	0.59	0.75	2.9	2.55
K		✓				✓	227.9	1.97	1.65	0.67	0.88	7.74	3.2
H	✓					✓	222.0	2.48	0.90	2.32	1.87	10.02	10.02
A	✓						191.5	2.53	0.49	2.48	1.9	10.3	10.3

G, which uses the INS aided by the ZUPT algorithm, altimeter, foot-to-foot ranging measurements, the inter-agent range measurements from the other two agents, and LTE pseudorange measurements.

- It could be observed that the maximum position errors did not always occur at the end of the experiment. This could be because, for dead reckoning systems, estimation errors on trajectory length get canceled out at return-to-home or loop-closure positions.

- In the configurations involving the LTE module, an innovation-based outlier detection module, as discussed in Section 8.2.3, was used to produce the opportunistic navigation solutions. The outlier detection module detected if the LTE signal had large biases caused by the multi-path effect. In the case of a positive detection, the LTE psuedorange measurement was not used to augment the navigation solutions. It is worth mentioning that, in the experiment, it could be observed that the outlier detection module indicated several positive detections, and therefore, not all the LTE signals collected during the experiment were used in configuration G, H, I, and J. Nevertheless, when the outlier detection module showed negative detections, the opportunistic solution provided compensation for absolute position errors, increasing navigation accuracy.

- Configuration D and configuration E used deterministic solutions enhanced by inter-agent measurements from only one agent, but the former configuration had a smaller final displacement error. The difference was considered as a result of the experimental setup that agent No.1 first passed by agent No.3 and then agent No.2. This setup led to an advantage of configuration D that the position estimates were corrected by the stationary agent at a later time, and therefore, the final position estimated by configuration D had a smaller error than configuration E.

- In the cases of configurations A and H, where altimeter measurements were not used, the final position errors are much larger than in the other configurations. For configuration A, the final error is larger because when operating in the moving elevator, the stance phase detector used in the ZUPT algorithm would indicate that it is the stance phase and correct the velocity to zero, while in reality, the altitude of the agent was changing. In configuration H, it could be seen that augmenting the ZUPT-aided INS with LTE measurements could reduce the error. However, the horizontal distance between the receiver is significantly larger than the altitude of the LTE towers. As such, the agent's cellular-based navigation solution Vertical Dilution Of Precision (VDOP) will be large. Yet, LTE reduced the vertical errors slightly compared to standalone ZUPT.

- In the developed PINDOC, aiding from LTE pseudorange measurements aims to compensate for absolute position errors. In the presented experiment with a duration of 14 min and a trajectory length of 600 [m], the position errors in systems using only the deterministic and cooperative approaches have not grown to large values. Therefore, the compensation of errors provided by LTE signals was not significant. However, it is expected that in navigation experiments with a longer duration, the LTE module will play a significant role in bounding the position error growth of PINDOC and improving navigation accuracy.

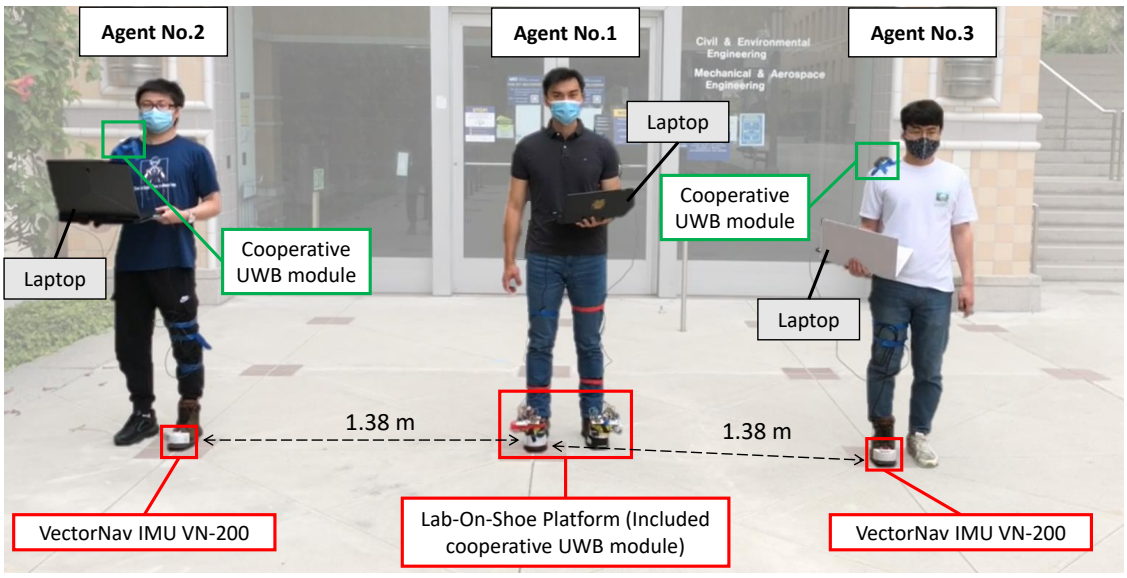


Figure 8.7: Experimental setup of the experiment discussed in Section 8.4.3.

8.4.3 Experiment #2: Three Moving Agents

Experiment Description

The second series of experiments involved three agents, all moving in the indoor environment shown in Figure 8.5. The experimental setup is shown in Figure 8.7. Agent No.1 was equipped with the Lab-On-Shoe platform integrated with the cooperative UWB modules discussed in Section 8.3. Agent No.2 in Figure 8.7 mounted a VectorNav VN−200 IMU on the left foot and attached a cooperative UWB module at the right shoulder. The position of the UWB module was assumed to be 1.4 [m] above the foot-mounted IMU. Agent No.3 in Figure 8.7 mounted another VectorNav VN−200 IMU on the right foot and attached another cooperative UWB module at the right shoulder. The position of the UWB module on agent No.3 was assumed to be 1.3 [m] above the foot-mounted IMU. Both VN−200 IMUs were configured to collect IMU measurements at a sampling rate of 800 Hz and altimeter measurements at a sampling rate of 10 Hz. The UWB modules attached to agent No.2 and agent No.3 were not connected with each other and were both programmed to be paired with the cooperative UWB module integrated on the right shoe of the Lab-On-Shoe platform used

by agent No.1. In Figure 8.7, the location of agent No.1's right foot was considered as the origin of the local coordinate frame, which had the same global coordinates as the starting position shown in Figure 8.5. The starting positions of the other two agents were 1.38 [m] apart from the origin, as shown in Figure 8.7.

At the beginning of the experiment, the three agents stood stationary at their starting positions for 15 seconds. During this period, all IMUs were calibrated. Then, agent No.3 first walked inside the building, followed by agent No.2 and then agent No.3. Figure 8.8 presents the trajectories of the three agents. The duration of the experiment was around 12.5 minutes and the lengths of the trajectories corresponding to agent No.1, agent No.2, and agent No.3 were around 600 [m], 540 [m], and 550 [m], respectively. The three agents traveled on different terrains, including flat planes, stairs, slopes, and elevators. From timestamps of 420 s to 443 s and from 568 s to 596 s, agent No. 2 entered office spaces and did not have LOS UWB range measurements with agent No.1. From timestamps of 346 s to 422 s, agent No.3 entered a laboratory space and did not have LOS UWB range measurements with agent No.1. At the end of the experiment, the three agents returned to their starting positions.

Experimental Results

In this experiment, we used the loop-closure error, described in Section 8.4.1, as the accuracy metric to compare the PINDOC system implemented in different configurations. Four different PINDOC configurations involving ZUPT, ALT, F2F, and CL, were used to produce navigation solutions for agent No.1, and three different PINDOC configurations involving ZUPT, ALT, and CL, were used to estimate navigation solutions for agent No.2 and agent No.3. In the case of CL, inter-agent range measurements between agent No.1 and agent No.2 as well as the range measurements between agent No.1 and agent No.3 were used. Table 8.4 presents the navigation accuracy of the different configurations for different agents.

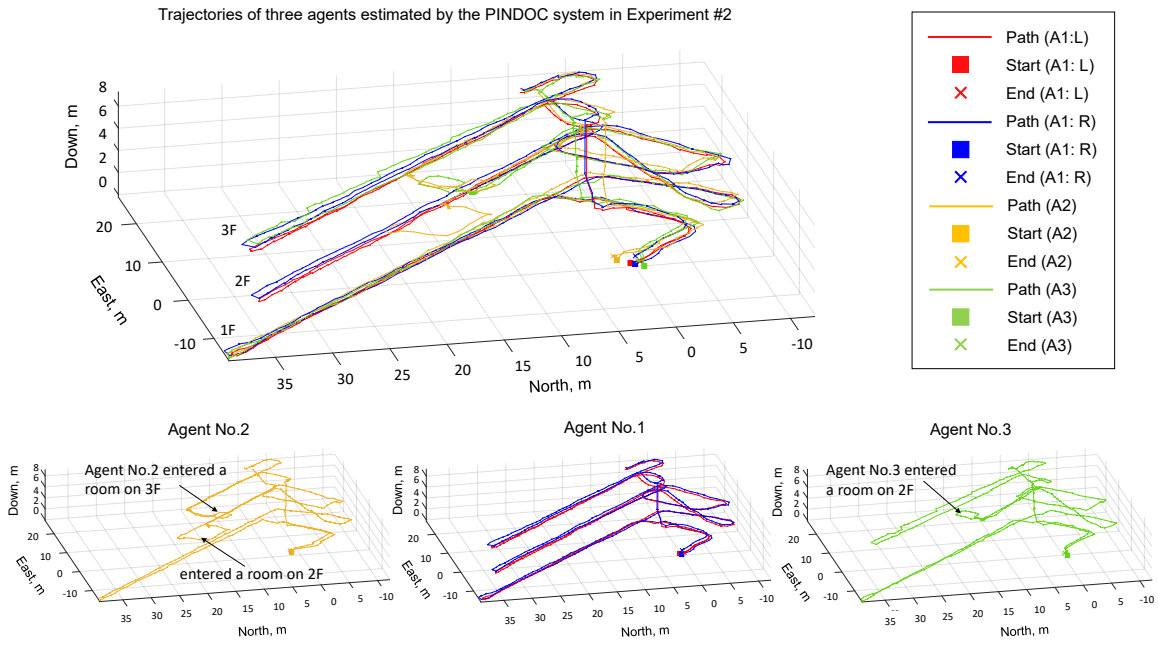


Figure 8.8: The top plot shows the navigation solutions of the three agents produced by the PINDOC system in the experiment discussed in Section 8.4.3. The bottom three plots separately present the same navigation solution of each agent. Agent No.1's trajectories were generated with the ZUPT-aided INS augmented with altimeter measurements, foot-to-foot ranging, and inter-agent ranging measurements. Agent No.2 and agent No.3's trajectories were generated with the ZUPT-aided INS augmented with altimeter measurements, and inter-agent ranging measurements.

Discussion

Three remarks could be made based on the experimental results shown in Table 8.4.

- The navigation solutions of the three agents based on standalone ZUPT-aided INS had the largest errors because the stance phase detector used in the ZUPT algorithm indicated stationary phases when the agents were inside a moving elevator, leading to falsely updating the velocity estimate to zero. The errors introduced by the elevator were also discussed previously in Section 8.4.2. When altimeters were used to enhance the navigation solutions, the errors associated with the three agents were reduced.

- when inter-agent range measurements were used, the errors of agent No. 2 and agent

Table 8.4: Navigation errors of the PINDOC implemented in different configurations in an experiment discussed Section 8.4.3

Agent	**INS Aiding Method**				**Final**
	ZUPT	ALT	F2F	CL	**Error [m]**
No.1	✓	✓	✓	✓	0.35
	✓	✓	✓		0.44
	✓	✓			0.84
	✓				10.27
No.2	✓	✓		✓	0.82
	✓	✓			4.25
	✓				13.41
No.3	✓	✓		✓	1.15
	✓	✓			4.38
	✓				15.83

No.3 were greatly reduced while the error of agent No.1 had only a marginal improvement. This phenomenon was because, as compared to the IMUs on the Lab-On-Shoe platform, the VectorNav IMUs mounted on agent No.2 and agent No.3 had higher noise levels and biases, leading to position uncertainties growing faster than the uncertainties of agent No.1. As a result, inter-agent range measurements had larger impacts on positions of agent No.2 and agent No.3 in this experiment than those of agent No.1.

- Among all the PINDOC implementations used in the experiment, the implementation using ZUPT-aided INS augmented with altimeter, foot-to-foot ranging, and inter-agent ranging had the smallest loop-closure errors of 0.35 [m] for agent No.1, 0.82 [m] for agent No.2, and 1.15 [m] for agent No.3. The trajectories of the three agents estimated by the later PINDOC implementation are presented in Figure 8.8.

8.5 Conclusion

This chapter discussed the PINDOC, which is a multi-agent pedestrian navigation system integrating the deterministic, the opportunistic, and the cooperative functionalities. The deterministic module uses sensing modalities, including IMUs, altimeters, and foot-to-foot range measurements. The opportunistic module utilizes cellular LTE pseudorange measurements, corrects clock biases with a base/rover framework, and mitigates multipath effect by LTE-SAN. The cooperative module enhances the navigation accuracy of each agent with UWB-based inter-agent range measurements. A dedicated multi-sensor hardware platform was developed and used to conduct an experiment with the platform to evaluate the navigation performance of the developed PINDOC. The experiment was a 14-min and 600-m indoor pedestrian navigation task involving three agents, two of which were stationary and one of which was walking on terrains of flat surfaces, stairs, ramps, and elevators. This chapter compared navigation accuracy using different configurations of the PINDOC for the moving agent. The experimental results showed that the configuration using ZUPT-aided INS enhanced by altimeters, foot-to-foot ranging, inter-agent ranging from the other two agents, and LTE pseudoranges had an position RMSE of 0.93 [m] and a position error SD of 0.44 [m], which are the best navigation accuracy, as compared to other configurations of the developed PINDOC. We conclude that the developed PINDOC had a navigation error of less than 1 meter in terms of position RMSE in the experiment. The results presented in this chapter have been published in [83].

Chapter 9

On Estimation Filter – SLAMing With UWB and Foot-mounted IMU

9.1 Introduction

This chapter discusses a SLAM framework based on foot-mounted IMUs and UWB, referred to as the UWB-Foot-SLAM, that allows for bounding position error growth of a ZUPT-aided INS along the 3-axis. The rest of this chapter is organized as follows. Section 9.2 presents theory, implementation, and experimental validation of the UWB-Foot-SLAM, Section 9.3 presents the UWB-Foot-SLAM2 algorithm that uses two additional self-contained fusion solutions to enhance the original UWB-Foot-SLAM, and Section 9.4 concludes this chapter with a highlight of the experimental results.

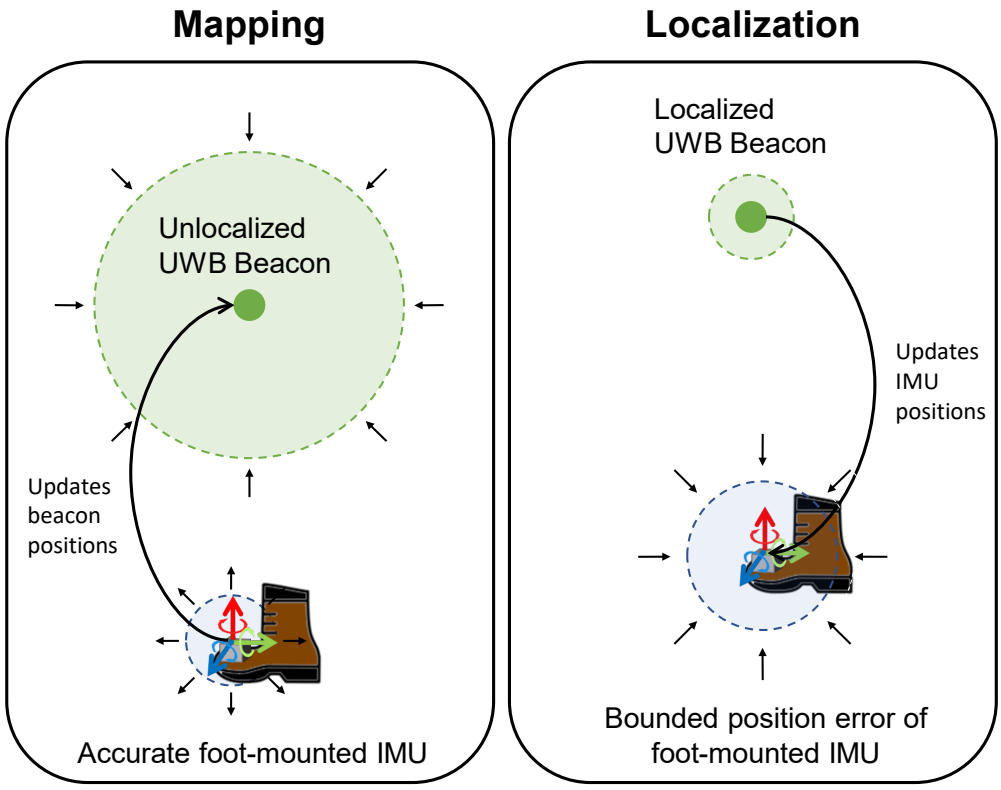


Figure 9.1: Concept of the developed UWB-Foot-SLAM.

9.2 The Original UWB-Foot-SLAM

In this section, we develop an algorithm of Simultaneous Localization And Mapping using both UWBs and foot-mounted IMUs referred to as UWB-Foot-SLAM. The developed UWB-Foot-SLAM, illustrated in Figure 9.1, is designed to use UWB range measurements to bound position errors of ZUPT-aided INS using foot-mounted IMUs without the need to pre-deploy and pre-survey the UWB beacons. The developed UWB-Foot-SLAM algorithm considers that a pedestrian performing navigation is equipped with hardware consisting of a foot-mounted platform that integrates an IMU and a UWB module and multiple standalone UWB beacons. During a navigation task, the pedestrians deploy the unknown UWB beacons in the environments. The developed UWB-Foot-SLAM algorithm sets locations of these UWB beacons as states to be estimated in an EKF framework and leverages a property of the ZUPT-aided INS that the system has very high position accuracy at the beginning of an

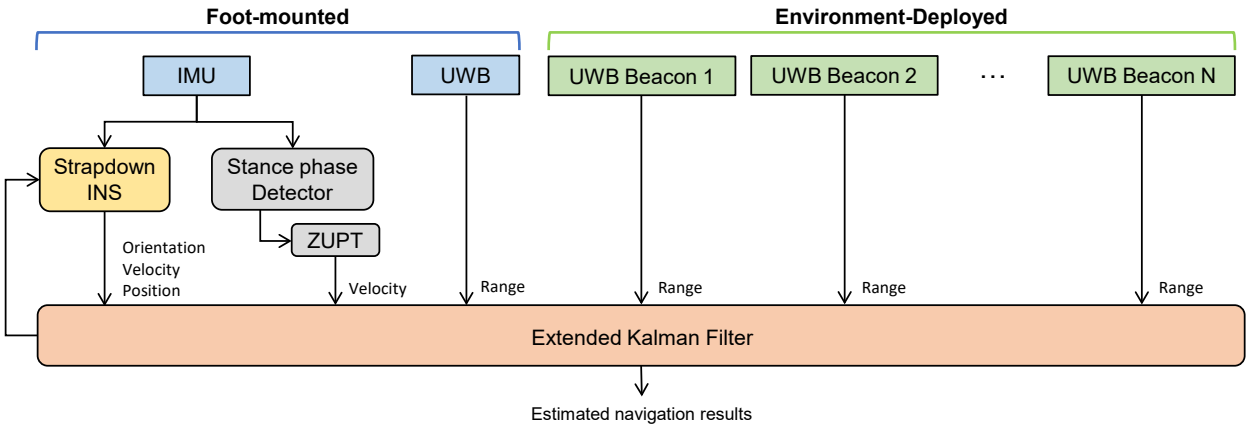


Figure 9.2: Block diagram illustrating the developed UWB-Foot-SLAM algorithm. The algorithm involved a foot-mounted IMU, a foot-mounted UWB, and several UWB beacons to be deployed in an operating environment during a navigation task.

operation. During this period, the localization solution of the ZUPT-aided INS is used to estimate unknown UWB beacons' locations based on range measurements between the foot-mounted UWB and the UWB beacons. When the estimation uncertainty of the UWB beacon's location reaches a sufficiently low value, the UWB range measurements are used to provide position compensation for the ZUPT-aided INS.

This section is organized as follows. Section 9.2.1 describes the algorithm of the developed UWB-Foot-SLAM, Section 9.2.2 discusses a hardware prototype developed to validate the developed algorithm, Section 9.2.3 presents the results of two different experiments and concludes the section with a highlight of the results.

9.2.1 Algorithm Design

Overview

The developed UWB-Foot-SLAM simultaneously localizes a pedestrian's positions and maps positions of unknown beacons with measurements collected from a foot-mounted IMU, a foot-mounted UWB sensor, and several UWB sensors to be deployed in a surrounding environment

during navigation. Figure 9.2 shows a block diagram illustrating the developed UWB-Foot-SLAM algorithm. To differentiate UWB sensors for different usages, we will refer to a UWB sensor mounted on the foot as a foot-mounted UWB and a UWB deployed in a navigation environment as a UWB beacon in the rest of the text. We consider a UWB beacon to have two statuses: *unlocalized* and *localized*. An unlocalized UWB beacon has an estimated position uncertainty significantly larger than the position uncertainty of a foot-mounted IMU, while the position uncertainty of a localized UWB beacon is similar to or smaller than that of the foot-mounted IMU. All of the UWB beacons when first deployed are considered to be in the unlocalized status.

In our developed UWB-Foot-SLAM, localization of a pedestrian utilizes a combination of the ZUPT-aided INS and foot-to-beacon range measurements collected from a pair of foot-mounted UWB and a localized UWB beacon. If a UWB beacon is in the unlocalized status, its range measurements would have minimal numerical impacts on the estimated pedestrian's positions. Mapping of the developed UWB-Foot-SLAM estimates unlocalized beacons' positions using the knowledge of a pedestrian's current position and foot-to-beacon range measurements collected from a pair of foot-mounted UWB and the unlocalized UWB beacons. The localization and mapping steps are both achieved with an EKF. After initialization of the EKF, the filter enters the mapping step. In this step, the uncertainties of the UWB position states are reduced, and the uncertainties of the foot-mounted IMU position states increase. After operating for a while, the filter performs the localization step, and the position error growth of the foot-mounted IMU is bounded.

Extended Kalman Filter

The developed UWB-Foot-SLAM algorithm is realized in an EKF framework, shown in Figure 9.2. N number of beacons are assumed to be involved in a navigation task.

Filter States The EKF uses states that includes orientations, velocities, and positions of a foot-mounted IMU as well as the positions of UWB beacons. The states are expressed as follows:

$$\mathbf{x}_k = \begin{bmatrix} \mathbf{q}_k^\top & \mathbf{v}_k^\top & \mathbf{p}_k^\top & \mathbf{b}_{\mathrm{g},k}^\top & \mathbf{b}_{\mathrm{a},k}^\top & \mathbf{p}_{\mathrm{B}_1,k}^\top & \cdots & \mathbf{p}_{\mathrm{B}_N,k}^\top \end{bmatrix}^\top,$$

where $\mathbf{q}_k$, $\mathbf{v}_k$, and $\mathbf{p}_k \in \mathbb{R}^{3\times 1}$ are the orientation, velocity, and position states of an agent expressed in the navigation frame. $\mathbf{b}_{\mathrm{g},k}$ and $\mathbf{b}_{\mathrm{a},k} \in \mathbb{R}^{3\times 1}$ are the gyroscope and accelerometer biases along the three axes of the IMU body frame. $\mathbf{p}_{\mathrm{B}_1,k}, \ldots, \mathbf{p}_{\mathrm{B}_N,k} \in \mathbb{R}^{3\times 1}$ represent the position of the N UWB beacons being deployed.

Filter Initialization In the beginning of a navigation task, we assumed a pedestrian would remain stationary for a short period. The initial roll angle, θ_0, and pitch angle, ϕ_0, are expressed as follows:

$$\theta_0 = \tan^{-1}(\frac{-\bar{\mathbf{a}}_y}{-\bar{\mathbf{a}}_z}), \phi_0 = \tan^{-1}(\frac{\bar{\mathbf{a}}_x}{\sqrt{\bar{\mathbf{a}}_y^2 + \bar{\mathbf{a}}_z^2}}),$$

where $\bar{\mathbf{a}}_i$ indicate the averaged ith-axis accelerometer readings collected during the initialization period. The initial yaw angle, ψ_0, can be determined with additional sensors, such as a magnetometer. The initial position, $\mathbf{p}_0$, and velocity, $\mathbf{v}_0$, can be determined with external localization systems. Accelerometer states, $\mathbf{b}_{\mathrm{a},0}$, is set to zeros. Initial gyroscope biases, $\mathbf{b}_{\mathrm{g},0}$, are expressed as

$$\mathbf{b}_{\mathrm{g},0} = \begin{bmatrix} \bar{\boldsymbol{\omega}}_x & \bar{\boldsymbol{\omega}}_y & \bar{\boldsymbol{\omega}}_z \end{bmatrix}^\top,$$

where $\bar{\boldsymbol{\omega}}_i$ indicates the averaged ith-axis gyroscope readings collected during the initialization period.

When a UWB beacon with an IDentification (ID) number j is first connected to the foot-mounted UWB at time k, the developed UWB-Foot-SLAM sets the initial beacon positions, denoted as $\mathbf{p}_{\mathrm{B}_j,0}$, with the current position of the foot-mounted IMU, $\mathbf{p}_k$. The corresponding initial position uncertainties, $\sigma_{\mathbf{p}_{\mathrm{B}_j},0}$, are initialized with the size of a navigation environment. $\mathbf{p}_{\mathrm{B}_j,0}$ and $\sigma_{\mathbf{p}_{\mathrm{B}_j},0}$ are expressed as follows.

$$\mathbf{p}_{\mathrm{B}_j,0} = \mathbf{p}_k, \sigma_{\mathbf{p}_{\mathrm{B}_j},0} = \mathrm{A},$$

where A is the dimension of a navigation environment.

Prediction Step In the prediction step of the EKF, the states corresponding to the foot-mounted IMU, including $\mathbf{q}_k, \mathbf{v}_k, \mathbf{p}_k, \mathbf{b}_{\mathrm{g},k}$, and $\mathbf{b}_{\mathrm{a},k}$, are propagated with the strapdown INS algorithm [198]. The position states of the beacons are assumed constant. The EKF propagation matrix, denoted by $\mathbf{A}_k$, is expressed as follows

$$\mathbf{A}_k = \begin{bmatrix} \mathbf{A}_{\mathrm{INS},k} & \mathbf{0}_{15\times 3N} \\ \mathbf{0}_{3N\times 15} & \mathbf{0}_{3N\times 3N} \end{bmatrix}$$

$$\mathbf{A}_{\mathrm{INS},k} = \begin{bmatrix} \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & -\mathrm{C}(\mathbf{q}_k) & \mathbf{0}_{3\times3} \\ [\overrightarrow{f}^n\times] & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathrm{C}(\mathbf{q}_k) \\ \mathbf{0}_{3\times3} & \mathbf{I}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \end{bmatrix}.$$

Here, $[\overrightarrow{f}^n\times]$ is the skew-symmetric cross-product-operator of the accelerometer outputs of the IMU, expressed in the navigation frame. $\mathrm{C}(\mathbf{q})$ is the Directional Cosine Matrix (DCM) corresponding to the quaternion vector $\mathbf{q}$. $\mathbf{0}_{n\times m}$ indicates a zero matrix having n number of rows and m number of columns. The corresponding process noise matrix, denoted as $\mathbf{Q}_k$, is

expressed as

$$\mathbf{Q}_k = \begin{bmatrix} \mathbf{Q}_{\text{INS},k} & \mathbf{0}_{15\times 3N} \\ \mathbf{0}_{3N\times 15} & \mathbf{0}_{3N\times 3N} \end{bmatrix},$$

with $\mathbf{Q}_{\text{INS},k} =$

$$\begin{bmatrix} \sigma^2_{\text{ARW}}\mathbf{I}_{3\times 3} & \mathbf{0}_{3\times 3} & \mathbf{0}_{3\times 3} & \mathbf{0}_{3\times 3} & \mathbf{0}_{3\times 3} \\ \mathbf{0}_{3\times 3} & \sigma^2_{\text{VRW}}\mathbf{I}_{3\times 3} & \mathbf{0}_{3\times 3} & \mathbf{0}_{3\times 3} & \mathbf{0}_{3\times 3} \\ \mathbf{0}_{3\times 3} & \mathbf{I}_{3\times 3} & \mathbf{0}_{3\times 3} & \mathbf{0}_{3\times 3} & \mathbf{0}_{3\times 3} \\ \mathbf{0}_{3\times 3} & \mathbf{0}_{3\times 3} & \mathbf{0}_{3\times 3} & \sigma^2_{\text{AcRW}}\mathbf{I}_{3\times 3} & \mathbf{0}_{3\times 3} \\ \mathbf{0}_{3\times 3} & \mathbf{0}_{3\times 3} & \mathbf{0}_{3\times 3} & \mathbf{0}_{3\times 3} & \sigma^2_{\text{RRW}}\mathbf{I}_{3\times 3} \end{bmatrix}.$$

Here, $\mathbf{I}_{n\times n}$ is the identity matrix having n number of rows and columns. σ^2_{ARW}, σ^2_{VRW}, σ^2_{RRW}, and σ^2_{AcRW} are the ARW of the gyroscopes, the VRW of the accelerometers, the RRW of the gyroscopes, and the AcRW of the accelerometers of the foot-mounted IMU.

Update Step When a stance phase is detected, the ZUPT algorithm is activated to compensate for the velocity state in the update step of the EKF. The compensation is done by feeding in pseudo-measurements of zero velocity along the three axes, which is denoted as $\mathbf{v}_{\text{ZUPT},k} = \mathbf{0}_{3\times 1}$. In this section, the stance phase detection is achieved with the SHOE detector [181], which determines a stance phase if a test statistics, $T(\mathbf{u}_n) =$

$$\frac{1}{M}\sum_{k\in\Omega_n}\left(\frac{1}{\sigma^2_{\text{VRW}}} \parallel \mathbf{y}^{\alpha}_k - g\frac{\bar{\mathbf{y}}^{\alpha}_k}{\parallel \bar{\mathbf{y}}^{\alpha}_k \parallel} \parallel^2 + \frac{1}{\sigma^2_{\text{ARW}}} \parallel \mathbf{y}^{\omega}_k \parallel^2\right) < \gamma,$$

where $\mathbf{u}_n = \{\mathbf{y}_k\}_{k=n}^{k=N-1}$ with $\mathbf{y}_k = [\mathbf{y}^{\alpha}_k, \mathbf{y}^{\omega}_k]^{\top}$, $\mathbf{y}^{\alpha}_k$ is 3-axis accelerometer measurements at time k, $\mathbf{y}^{\omega}_k$ is 3-axis gyroscope measurements at time k, g is the gravitational constant, $\Omega_n = \{l \in \mathbb{N}, n \leq l < M-1\}$ is a collection of the sensor measurement indexes at time n with a window of length M, and γ are user-defined thresholds.

The ZUPT measurement models, $\mathbf{z}_{\mathrm{ZUPT},k}$, measurement matrices, $\mathbf{H}_{\mathrm{ZUPT},k}$, and measurement covariance matrices, $\mathbf{R}_{\mathrm{ZUPT},k}$ are expressed as follows:

$$
\begin{aligned}
&\mathbf{z}_{\mathrm{ZUPT},k} = \mathbf{v}_{\mathrm{ZUPT},k} \\
&\mathbf{H}_{\mathrm{ZUPT},k} = \begin{bmatrix} \mathbf{0}_{3\times 3} & \mathbf{I}_{3\times 3} & \mathbf{0}_{(9+3N)\times 3} \end{bmatrix} \\
&\mathbf{R}_{\mathrm{ZUPT},k} = \sigma^2_{\mathrm{ZUPT}} \mathbf{I}_{3\times 3},
\end{aligned}
$$

where σ^2_{ZUPT} is the noise variance of the zero-velocity measurement $\mathbf{v}_{\mathrm{ZUPT}}$.

When a ith UWB measurement, denoted as $r_{\mathrm{UWB}_i,k}$, becomes available, the measurements are first classified into Line-of-Sight (LoS) and Non-Line-of-Sight (NLoS) cases, and only LoS cases are used in the update step of the EKF. This section uses a probabilistic power metric approach to differentiate LoS and NLoS UWB measurements [32]. The LoS measurements are further processed with bias correction through a curve-fitting approach. The corresponding foot-to-beacon range measurement model, $z_{\mathrm{UWB}_i,k}$, measurement matrix, $\mathbf{H}_{\mathrm{UWB}_i,k}$, and measurement noise matrix, $\mathbf{R}_{\mathrm{UWB}_i,k}$, are described as follows:

$$
\begin{aligned}
&z_{\mathrm{UWB}_i,k} = r_{\mathrm{UWB}_i,k} \\
&\mathbf{H}_{\mathrm{UWB}_i,k} = \begin{bmatrix} \mathbf{0}_{6\times 1} \\ \frac{\partial ||\mathbf{p}_k - \mathbf{p}_{\mathrm{B}_i,k}||^\top}{\partial \mathbf{p}_k} \\ \mathbf{0}_{(6+3(i-1))\times 1} \\ \frac{\partial ||\mathbf{p}_k - \mathbf{p}_{\mathrm{B}_i,k}||^\top}{\partial \mathbf{p}_{\mathrm{B}_i,k}} \\ \mathbf{0}_{3(N-i-1)\times 1} \end{bmatrix}^\top \\
&\mathbf{R}_{\mathrm{UWB}_i,k} = \sigma^2_{\mathrm{UWB}_i},
\end{aligned}
$$

where $\sigma^2_{\mathrm{UWB}_i}$ is the noise variance of the foot-to-beacon range measurements between the foot-mounted UWB and the ith UWB beacon.

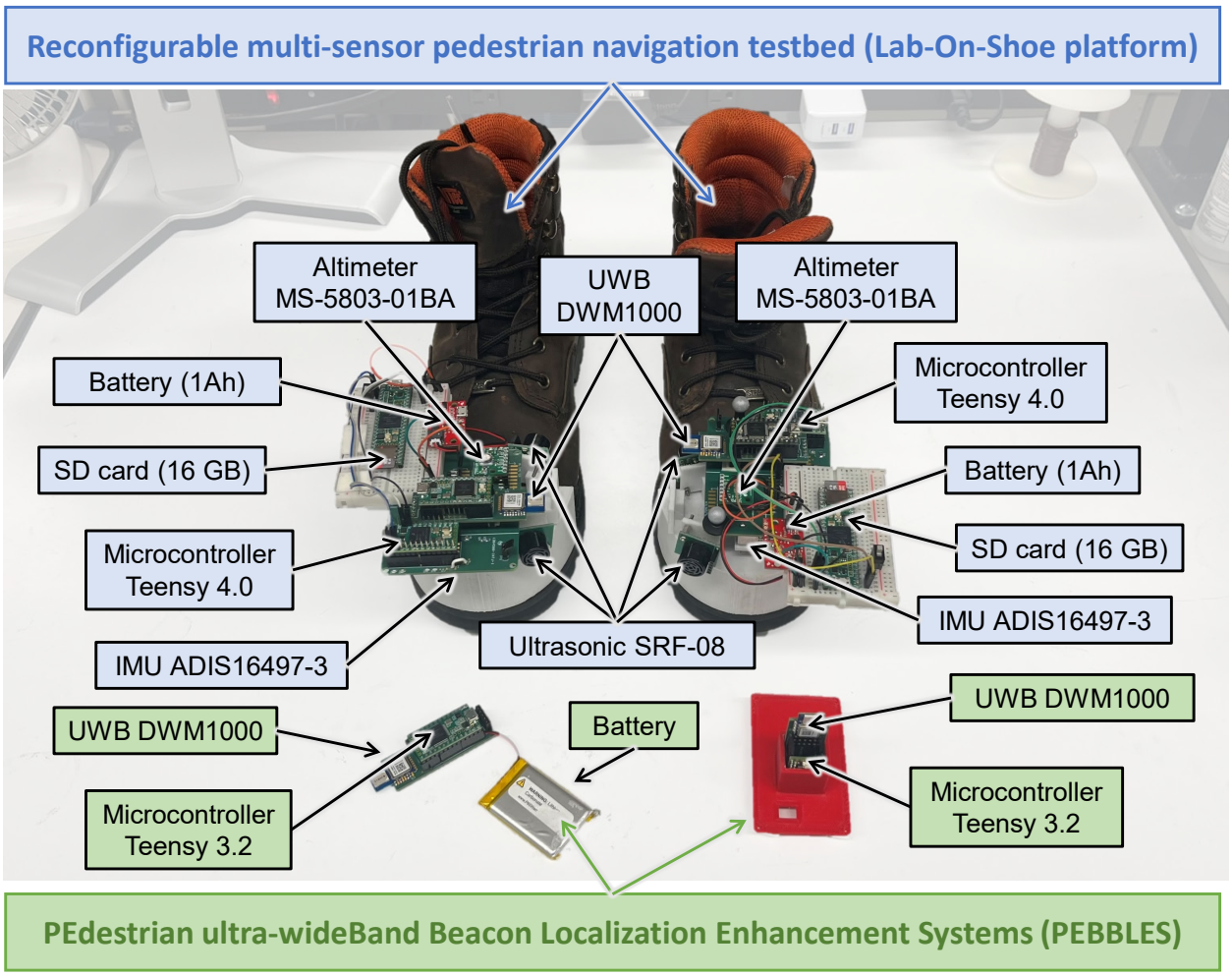


Figure 9.3: Experimental setup. The setup included the Lab-On-Shoe platform and the PEBBLE system. The Lab-On-Shoe platform included multiple sensing modalities. This section only used the IMU and UWB mounted on the left foot.

9.2.2 System Design

To realize the developed UWB-Foot-SLAM algorithm, we developed a reconfigurable multi-sensor pedestrian navigation testbed, referred to as the Lab-On-Shoe platform, and multiple integrated UWB beacon units, referred to as PEdestrian ultra-wideBand Beacon Localization Enhancement (PEBBLE) systems. This section discusses both the hardware and firmware implementation of the developed prototype.

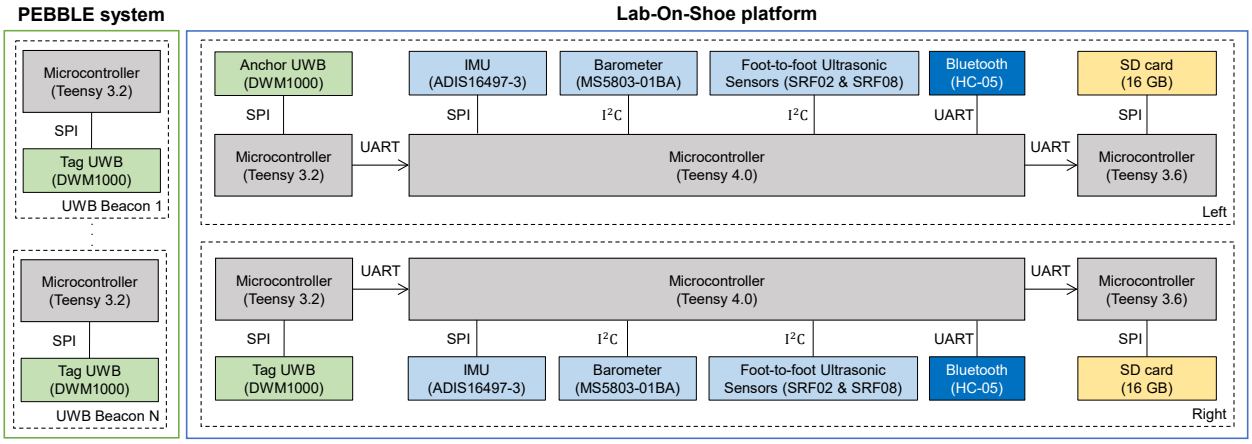


Figure 9.4: Block diagram illustrating firmware of the Lab-On-Shoe platform and the PEBBLE system.

Hardware Implementation

Figure 9.3 presents the developed experimental prototypes. The Lab-On-Shoe platform was previously developed as a flexible hardware testbed at the Microsystem Laboratory at the University of California, Irvine, with the purpose of investigating sensor fusion solutions for integrated pedestrian inertial navigation system [83, 18, 82]. In this section, we upgraded the platform. On each shoe, we integrated multiple COTS components, including three Teensy micro-controllers, an Analog Device IMUs ADIS16497−3, a barometric altimeter MS−5803−01BA, two ultrasonic range sensor SRF02, a UWB module DWM1000, and a SD card. The three Teensy microcontrollers are Teensy 4.0, Teensy 3.6, and Teensy 3.2, having CPU clock rates of 600 [MHz], 180 [MHz], and 96 [MHz], respectively. The SD card module was a built-in module on the Teensy 3.6 microcontroller. All the components were firmly mounted on a customized 3D-printed PolyLactic Acid (PLA) fixture. Each UWB beacon in the PEBBLE system included a microcontroller Teensy 3.2 and a UWB module DWM1000. The Lab-On-Shoe platform and the PEBBLE system were both powered up with 5.0-V lithium-ion batteries.

Firmware Implementation

Figure 9.4 presents a block diagram describing firmware schematics implemented on the Lab-On-Shoe platform and the PEBBLE system. On the Lab-On-Shoe platform, the Teensy 3.2 microcontroller collected information on the connected node ID number and measurements of range, transmitter power, receiver first pulse power, and power metrics at a rate of 10 [Hz] from the DWM1000 UWB module via the SPI communication protocol. The DWM1000 module mounted on the left shoe was programmed to operate in the anchor mode, and the module mounted on the right shoe was in the tag mode. The collected measurements were transmitted to the Teensy 4.0 microcontroller via the UART communication protocol. On the Teensy 4.0 microcontroller, we implemented the SPI protocol to collect IMU measurements at a rate of 1000 [Hz] as well as the I2C communication protocol to collect pressure and thermal measurements from the MS5803—01BA barometer at a rate of 25 [Hz] and inter-foot ranging measurements from the two SRF02 ultrasonic sensors at a rate of 25 [Hz]. After all the sensor measurements were collected at each implementation loop, the Teensy 4.0 transmitted the measurements to the Teensy 3.6 microcontroller via UART communication protocol. The Teensy 3.6 microcontroller implemented the SPI protocol to write all the received measurements to an SD card.

On the PEBBLE system, the Teeny 3.2 microcontroller implemented the SPI communication protocol to collect information on the connected node ID number and measurements of range, transmitter power, receiver first pulse power, and power metrics at a rate of 10 [Hz]. All the UWB modules in the developed PEBBLE system were programmed to operate in the tag mode. A DWM1000 UWB operating in the tag mode can only be paired with a UWB operated in the anchor mode, and the range measurements between the two UWBs were obtained through a two-way ranging method. Therefore, all the UWB modules involved in the PEBBLE system, when within a detectable range, were connected only to the UWB mounted on the left shoe of the Lab-On-Shoe platform.

9.2.3 Experimental Validation

We conducted two experiments to validate the developed UWB-Foot-SLAM using the developed Lab-On-Shoe platform and the PEBBLE system.

Scenarios #1: A Small Area With Reference Motion Capture Cameras

In the first experiment, a subject was equipped with the Lab-On-Shoe platform and carried two UWB beacons. Figure 9.5 shows the experimental scenario. The subject walked a close-loop trajectory along a square shape in a 6 [m] by 6 [m] area for around 3 minutes, resulting in a trajectory length of around 140 [m]. In the beginning of the experiment, the subject stood still at the origin for 10 seconds. Two beacons, denoted as beacon #1 and beacon #2, were deployed at the beginning of the experiment. The first LoS range measurements of beacon #1 were collected at the 15 [s] timestamp, and the LoS measurements of beacon #2 were collected at the 20 [s] timestamp. A set of OptiTrack motion capture cameras was used to obtain the ground truth positions of the two beacons and the subject's feet. The sampling rate of the camera system was 120 [Hz]. In this experiment, among all the measurements produced by the Lab-On-Shoe platform, we only used the ones collected by the IMU and UWB mounted on the left shoe.

We compared the navigation solutions computed by our developed UWB-Foot-SLAM algorithm with a standalone ZUPT-aided INS. The initial yaw angle and positions used in the estimated solutions were aligned with the coordinate system of the motion capture cameras. The EKF parameters used in this section are summarized in TABLE 9.1. Figure 9.6 presents the two navigation solutions. The ZUPT-aided INS solution only estimated the positions of the agent while the developed UWB-Foot-SLAM estimated both the agent's and beacons' positions. We used ground truth positions provided by the motion capture cameras to evaluate the accuracy of the estimated navigation solutions. Agent's positions estimated by the

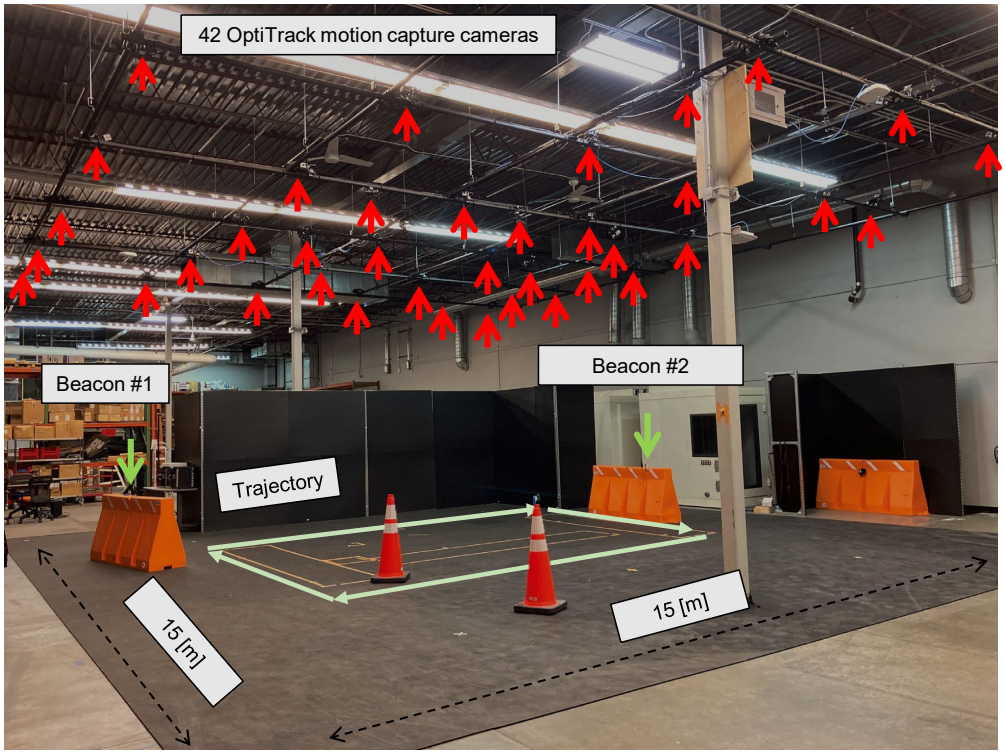


Figure 9.5: Experimental scenario for the experiment discussed in Section 9.2.3. 42 OptiTrack motion capture cameras were mounted on the ceiling of a warehouse and obtain the ground truth position and orientation. Two beacons were placed on top of the orange barricades during the experiment. A pedestrian performed the experiment by walking along the light green trajectory.

ZUPT-aided INS had a 2D RMSE of 0.15 [m], a 2D final displacement error of 0.22 [m], and a 2D maximum displacement error of 0.37 [m]. The positions estimated by the developed UWB-Foot-SLAM had a 2D RMSE, 2D final displacement error, and a 2D maximum displacement error of 0.15 [m], 0.27 [m], and 0.31 [m], respectively. Beacons' positions estimated by the developed UWB-Foot-SLAM had 2D final displacement errors of 0.28 [m] for beacon #1 and 0.22 [m] for beacon #2.

Figure 9.7 presents the covariances associated with the position states of the agent and the beacons when using the developed UWB-Foot-SLAM algorithm. We could see that the uncertainties of the agent's positions grew over time while the beacons' position uncertainties decreased. In this experiment, the developed UWB-Foot-SLAM was considered to operate in

the mapping mode, as the agent's position uncertainties had not grown beyond the beacons' position uncertainties at the end of the experiment. Therefore, the UWB range measurements did not have significant numerical impacts on the agent's estimated positions.

Scenarios #2: A large Area Including Multiple Floors

In the second experiment, the subject walked a closed-loop trajectory in a building on two different floors covering terrains of flat planes, stairs, and ramps. The experimental scenario had a physical dimension of approximately 70 [m] by 25 [m] by 6 [m]. The duration was around 25 minutes, and the trajectory length was approximately 1.5 [km]. The subject started the experiment on the second floor of the building and distributed four UWB beacons at different locations on the first and second floors of the building. Beacon #1, #2, #3, and #4 were deployed at timestamps of 21 [s], 72 [s], 197 [s], and 266 [s]. Beacon #1 and #2 were deployed on the first floor, and beacon #3 and #4 two were deployed on the second floor.

Figure 9.8 shows the navigation solutions estimated by the standalone ZUPT-aided INS and the developed UWB-Foot-SLAM. The 2D LCE of the agent's positions was 11 [m] when estimated by the ZUPT-aided INS and 1.49 [m] when estimated by our developed UWB-Foot-SLAM algorithm. The vertical displacement error was 4.5 [m] in the case of the ZUPT-aided

Table 9.1: Parameters for the EKF

Hyper-parameter	**Value**
σ_{ARW}	2.7221×10^{-5}
σ_{VRW}	0.0017
σ_{RRW}	8.3174×10^{-7}
σ_{AcRW}	6.63×10^{-6}
σ_{ZUPT}	0.02
σ_{UWB_i}	0.5

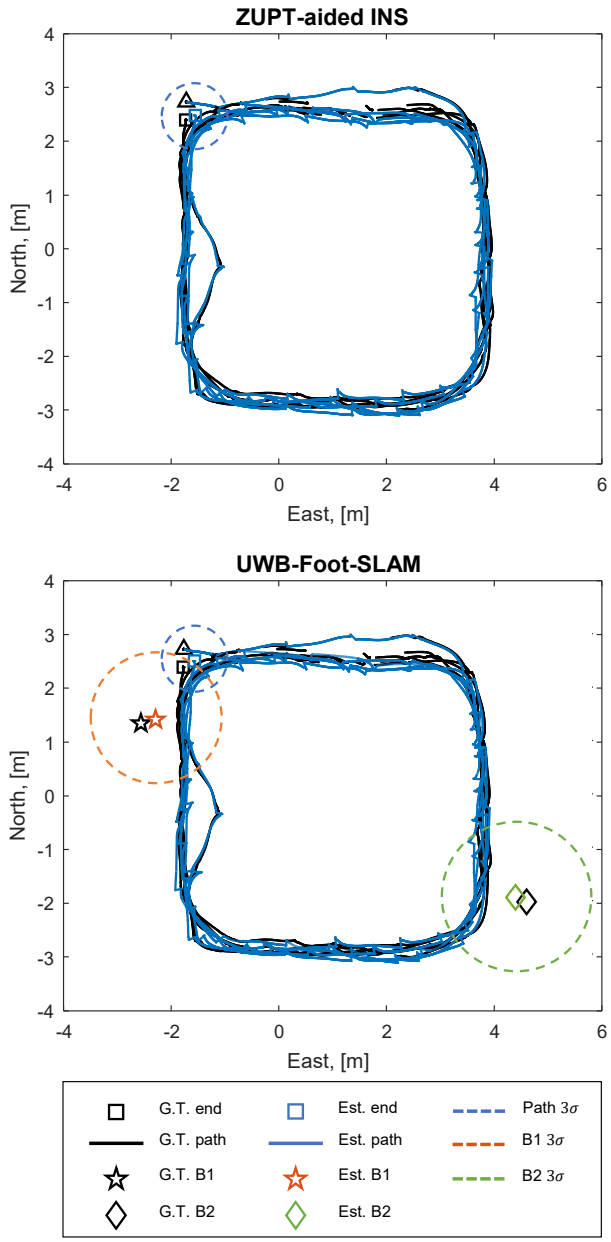


Figure 9.6: Estimated (Est.) Navigation solutions computed with a standalone ZUPT-aided INS and the developed UWB-Foot-SLAM in the experiment discussed in Section 9.2.3. Items colored in black correspond to the Ground Truth (G.T.) collected by motion capture cameras. The radius of each dashed circle represents three times the position standard deviation. Positions of Beacon #1 (B1) and beacon #2 (B2) are marked with star and diamond symbols, respectively.

INS and 1.09 [m] in the case of the UWB-Foot-SLAM. The positions of the deployed beacons estimated by the developed UWB-Foot-SLAM at the end of the experiment were colored in orange, green, yellow, and purple in Figure 9.8.

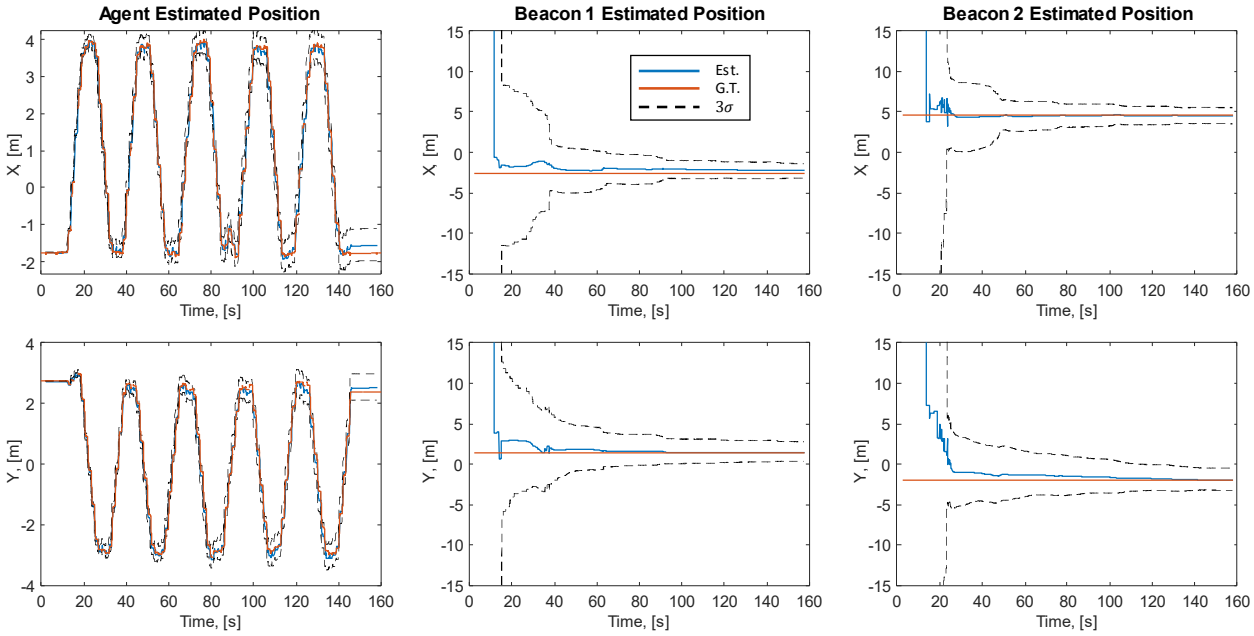


Figure 9.7: Position estimates and its associated covariances of the developed UWB-Foot-SLAM algorithm in the experiment discussed in Section 9.2.3. It could be observed that the covariances of the agent's positions increased over time while the covariances of the beacons' positions were reduced. At the end of this experiment, the covariances of the agent's positions were still less than that of the beacons' locations.

Figure 9.9 shows the position uncertainties of different states predicted by the EKF in the experiment. The horizontal position uncertainties in Figure 9.9 were computed by summing the three times Standard Deviation (3σ) along the x- and the y-axis. The vertical uncertainties were the 3σ along the z-axis. Three observations could be made in Figure 9.9. First, when the UWB beacons of the PEBBLE system were connected to the UWB module on the Lab-On-Shoe platform, the position uncertainties associated with the beacons were reduced. Second, At timestamps of 145 [s], the position uncertainty of the agent becomes larger than the position uncertainty of beacon #1 and #2. After this timestamp, the two beacons were considered localized and would start compensating for the position errors of the agents. Third, it could be seen that both the horizontal and vertical position uncertainties of the agent in the ZUPT-aided INS follow increasing trends while the uncertainties in the case of our developed UWB-Foot-SLAM were reduced and bounded by the localized beacons.

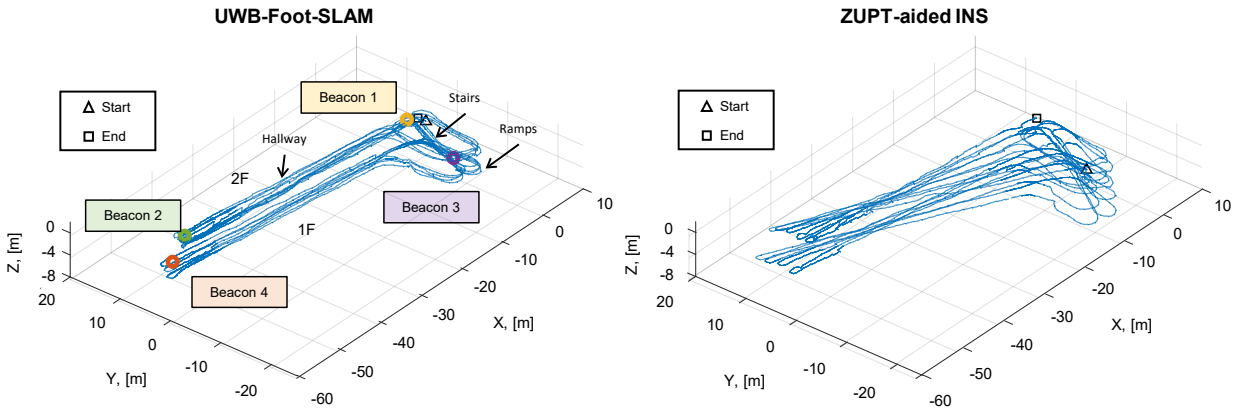


Figure 9.8: Navigation solutions estimated by the developed UWB-Foot-SLAM and a standalone ZUPT-aided INS in the experiment presented in Section 9.2.3.

The experimental results presented in Section 9.2.3 and Section 9.2.3 demonstrate that the developed UWB-Foot-SLAM could simultaneously localize unknown beacons' positions with sufficiently high accuracy. Range measurements collected by the localized beacons could provide position compensation in the developed approach, significantly improving long-term pedestrian navigation accuracy, as compared to a standalone ZUPT-aided INS.

Discussion

Several remarks could be made during navigation testing of the developed UWB-Foot-SLAM:

- Initialization of UWB position states can affect accuracy of the beacon's estimated position. In the developed approach, a beacon's position is initialized with an agent's current position. During the experiments presented in Section 9.2.3 and Section 9.2.3, the agent deployed a beacon within reach, matching the design of our approach. However, the deployment could be done in a more flexible manner, such as by throwing beacons to distant locations. In such cases, the initialization mechanism discussed in this section could lead to the estimated beacons' location being stuck in a statistical local minimum, degrading the navigation performance. One potential approach to ad-

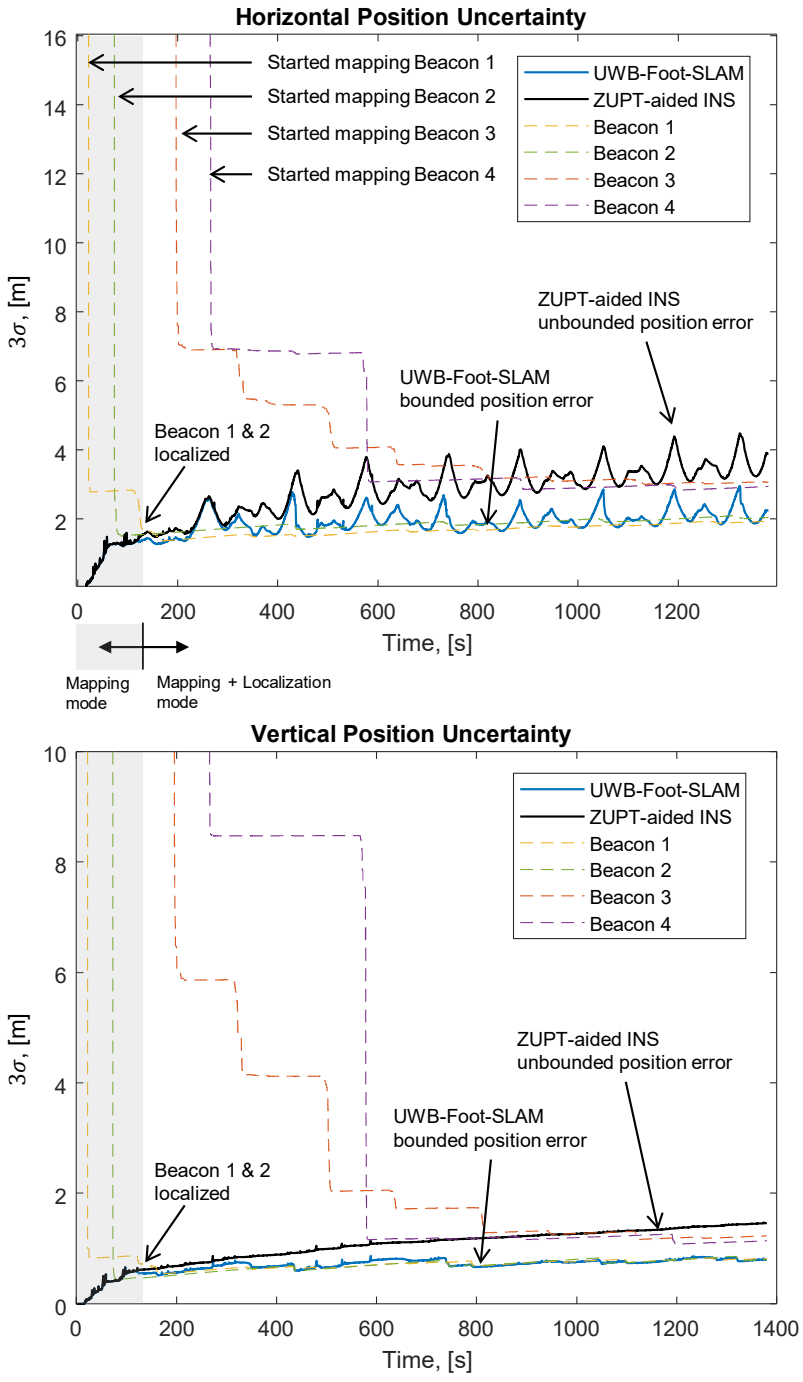


Figure 9.9: Propagation profile of the covariances associated with agent's and beacon's positions. It could be seen that the agent's position uncertainties were bounded in the case of the UWB-Foot-SLAM while the uncertainties in the case of the ZUPT-aided INS followed an increasing trend.

dress this issue is to use multiple initial guesses of a beacon's location, compute the likelihood of each guess, and select the one with the highest likelihood.

- The pattern of a pedestrian's navigation trajectory could affect the estimation accuracy of beacons' locations. This phenomenon is similar to the Position Dilution Of Precision (PDOP) in other RF-signal-based positioning systems [116]. During our testings of the developed UWB-Foot-SLAM algorithm, we observed that, when a pedestrian traveled only horizontally along a straight line, the estimated beacons' positions had significantly larger errors along the axes perpendicular to the direction of travel than those in parallel to the direction. The large errors could exceed the associated covariances, indicating existence of unmodeled errors in the estimation filter. It would be beneficial for future research to develop a multi-model approach to mitigate this issue.

- The ability to identify and compensate for NLoS UWB range measurements directly affects both the mapping and localization performance of our developed UWB-Foot-SLAM. The experimental prototype discussed in Section 9.2.2 included foot-mounted UWB modules. This configuration was designed to avoid the need to estimate relative positions between a UWB and an IMU attached to a pedestrian. However, as compared to other mounting positions, such as head or shoulder, foot-mounted UWBs had more difficulties in receiving LoS measurements, as the modules were close to the ground and the direct signal path could be blocked by a pedestrian body part [32]. To improve the UWB range measurement accuracy, advanced LoS/NLoS detection and bias compensation approaches could be advantageous [262].

- The UWB-Foot-SLAM algorithm developed in this section was realized in a centralized framework, where all the states were updated in every iteration of the EKF, even if some of the beacons were not connected. The developed UWB-Foot-SLAM could be extended to a de-centralized framework [264], which is computationally less expensive and would be more friendly to be implemented in real-time on a microcontroller. It would be beneficial to investigate the trade-offs between navigation performance and computational complexity of the centralized and de-centralized realizations.

- The performance of localization of pedestrians and mapping unknown beacons in our developed UWB-Foot-SLAM depends on the performance of the built-in ZUPT-aided INS. Our developed approach could be improved with an enhanced ZUPT-aided INS. The enhancement could be achieved on multiple different aspects of the system, including robust stance phase detection [178, 93, 223], self-contained sensor fusion solutions [94, 117, 82, 1], and IMU compensation [90, 87].

This section developed a UWB-Foot-SLAM algorithm that simultaneously localizes positions of a pedestrian and maps locations of unknown beacons. We developed an experimental prototype, including the Lab-On-Shoe platform and the PEBBLE system, and compared the performance of the developed UWB-Foot-SLAM algorithm with a standalone ZUPT-aided INS in two different experiments. The first experiment involved evaluating the navigation solutions with a high-accuracy motion capture camera system. The ZUPT-aided INS had a position RMSE of 0.15 [m] and an LoS of 0.22 [m]. The developed UWB-Foot-SLAM had a position RMSE of 0.15 [m] and an LoS of 0.27 [m]. Positions of the two UWB beacons estimated by the developed UWB-Foot-SLAM had displacement errors of 0.28 [m] and 0.22 [m], respectively. In the second experiment, the ZUPT-aided INS had an LoS of 11 [m] along the horizontal direction and 4.5 [m] along the vertical direction. The developed UWB-Foot-SLAM algorithm achieved an LoS of 1.49 [m] along the horizontal direction and 1 [m] along the vertical direction. The experimental result also showed that the EKF covariances associated with pedestrian's positions in the case of the developed UWB-Foot-SLAM were bounded. The experimental results show that the developed UWB-Foot-SLAM algorithm could significantly improve the long-term positioning accuracy of a pedestrian inertial navigation system using foot-mounted IMUs.

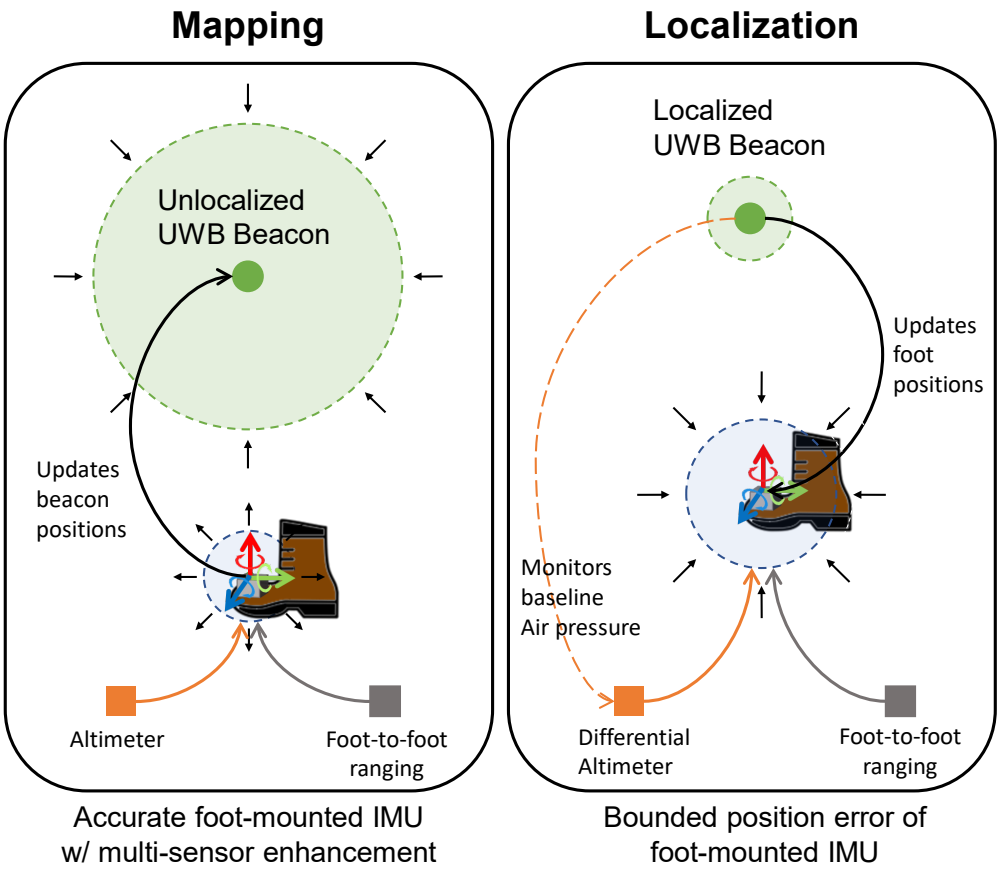


Figure 9.10: Concept of the developed UWB-Foot-SLAM2 algorithm.

9.3 UWB-Foot-SLAM2

This section develops an augmented UWB-Foot-SLAM framework, referred to as UWB-Foot-SLAM2. The developed UWB-Foot-SLAM2, illustrated in Figure 9.10, considers pedestrian navigation hardware consisting of a foot-mounted platform and a set of multiple beacons. The two subsystems are integrated with IMUs, barometers, and UWB. The UWB-Foot-SLAM2 algorithm inherits the original UWB-Foot-SLAM algorithm's characteristics of using UWB beacons to bound the position error propagation of foot-mounted INS without the need to pre-surveying the beacons and enhance the original algorithm with three additional mechanisms: foot-to-foot ranging measurements, differential altimeters, beacon motion detection. The foot-to-foot ranging approach uses UWB measurements of distances between the two

feet to enhance a dual foot-mounted INS configuration [32, 117]. The differential barometer mechanism combines the foot-mounted barometer with barometers integrated into beacon nodes to monitor barometric altitude errors due to ambient air pressure changes, aiming to improve the reliability of altimeter measurements[238]. Beacon motion detection is included in UWB-Foot-SLAM2 to avoid violating the assumption made by the original UWB-Foot-SLAM that a beacon is always stationary. When a beacon experiences motion, the estimated positions of the beacon are re-initialized and re-mapped.

This section is organized as follows. Section 9.3.1 describes the algorithm of the developed UWB-Foot-SLAM2, Section 9.3.2 discusses a hardware prototype developed to validate the developed algorithm, Section 9.3.3 presents the results of two different experiments and concludes the section with a highlight of the results and suggests potential future research directions.

9.3.1 Algorithm Design

This section first defines terminologies commonly used throughout the rest of this section, discusses an overview of the developed UWB-Foot-SLAM2, and presents the detailed implementation of an EKF used to implement the developed approach.

Terminology Definition

We define the following terminologies and will use them in the rest of the text. A foot-mounted localization system is referred to as two subsystems located on two separate feet of a person. Each subsystem consists of an IMU, a barometer, and a UWB. A beacon is referred to as a module to be deployed during a navigation environment, and one such module includes an IMU, a barometer, and a UWB transceiver. An IMU mounted on the foot is called a

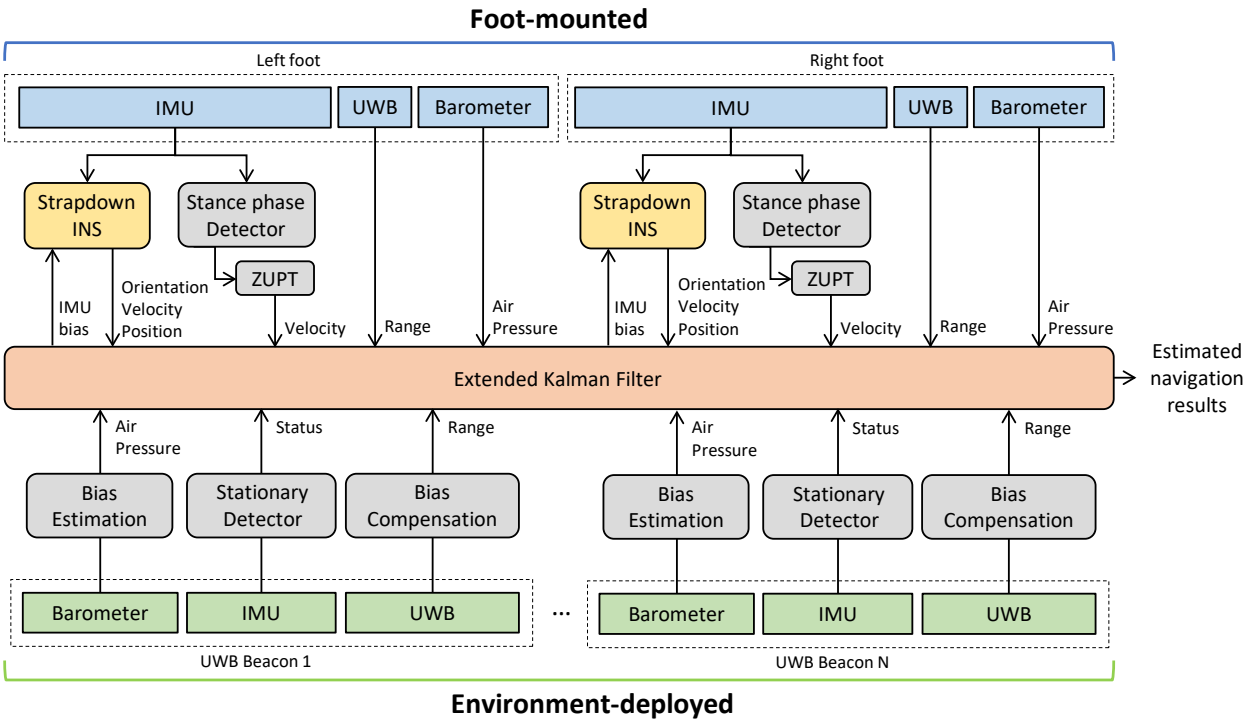


Figure 9.11: Block diagram illustrating the developed UWB-Foot-SLAM2 algorithm. The algorithm involved two IMUs, two barometers, two UWBs mounted on a foot-mounted localization system, as well as seeded UWBs, reference barometers, and event IMUs integrated into beacons to be deployed in an operating environment during a navigation task.

foot-mounted IMU, and an IMU integrated into a beacon is defined as an event IMU. A barometer co-located with a foot-mounted IMU is referred to as a foot-mounted barometer, and a barometer integrated into a beacon is referred to as a reference barometer. A UWB sensor mounted on foot is referred to as a foot-mounted UWB, and a UWB integrated into a beacon is called a seeded UWB. Distance measurements collected from a pair of foot-mounted UWBs located on two different feet are defined as foot-to-foot range measurements, and distance measurements collected from a pair of a foot-mounted UWB and a seeded UWB are referred to as foot-to-beacon range measurements. We consider a beacon to have two statuses: *unlocalized* and *localized*. An unlocalized beacon has an estimated position uncertainty significantly larger than the position uncertainty of a foot-mounted localization system, while the position uncertainty of a localized UWB beacon is similar to or smaller than that of the foot-mounted localization system.

Algorithm Overview

The developed UWB-Foot-SLAM2 simultaneously localizes a pedestrian's positions and maps positions of unknown beacons with measurements collected from all sensors integrated into a foot-mounted localization system and multiple beacons. Figure 9.11 shows a block diagram illustrating the developed UWB-Foot-SLAM2 algorithm. The algorithm is designed to be implemented with an EKF framework. At the beginning of a navigation task, all the beacons are considered to be in the unlocalized status, and the EKF initializes the position of the beacons with very large uncertainties. Right after the initialization, the filter enters the mapping mode, where the uncertainties of the unlocalized beacon position states are reduced, and the uncertainties of position states associated with the foot-mounted localization system increase. In this mode, the corresponding foot-to-beacon range measurements would have minimal numerical impacts on the estimated pedestrian's positions. When the position uncertainties of the beacons become smaller than that of the foot-mounted system, the beacons are localized, and the filter performs both mapping and localization modes. In this mode, the position error growth of the foot-mounted localization system is bounded with foot-to-beacon range measurements collected from a pair of foot-mounted UWB and the localized seeded UWB.

In our developed UWB-Foot-SLAM2, localization of a pedestrian uses a combination of the ZUPT-aided INS and measurements of foot-mounted barometers, reference barometers, foot-to-foot ranges, and foot-to-beacon ranges. Reference barometers monitor baseline air pressure changes and local pressure fluctuations and are used in combination with foot-mounted barometers to achieve a differential altimeter mechanism. When a motion is detected by an event IMU embedded on a beacon, the estimated beacon's position can no longer be trusted, and our developed algorithm re-initializes the associated positions and uncertainties.

In the following subsections, we present the EKF used to realize the developed UWB-Foot-

SLAM2 algorithm.

EKF States and Initialization

The developed UWB-Foot-SLAM2 algorithm is realized in an EKF framework, shown in Figure 9.11. N denotes the number of deployed UWB beacons. The location of these beacons are unknown. The pedestrian's navigation system however, knows the unique Media Access Control (MAC) addresses of the beacons. Therefore, UWB-Foot-SLAM2 does not need a data associate step when measurements are taken from the beacons.

Filter States The EKF uses states that includes orientations, velocities, positions, biases of two foot-mounted IMUs, biases of the two barometers, and the positions of beacons. The states are expressed as follows:

$$\begin{aligned}\mathbf{x}_k = [&\mathbf{q}_{\mathrm{L},k}^\top, \mathbf{v}_{\mathrm{L},k}^\top, \mathbf{p}_{\mathrm{L},k}^\top, \mathbf{b}_{\mathrm{g,L},k}^\top, \mathbf{b}_{\mathrm{a,L},k}^\top, \mathrm{b}_{\mathrm{b,L},k}, \mathbf{q}_{\mathrm{R},k}^\top, \mathbf{v}_{\mathrm{R},k}^\top, \mathbf{p}_{\mathrm{R},k}^\top, \\ &\mathbf{b}_{\mathrm{g,R},k}^\top, \mathbf{b}_{\mathrm{a,R},k}^\top, \mathrm{b}_{\mathrm{b,R},k}, \mathbf{p}_{\mathrm{B}_1,k}^\top, \ldots, \mathbf{p}_{\mathrm{B}_N,k}^\top]^\top \in \mathbb{R}^{(32+3N)\times 1},\end{aligned}$$

where $\mathbf{q}_{\mathrm{L},k}$, $\mathbf{v}_{\mathrm{L},k}$, and $\mathbf{p}_{\mathrm{L},k} \in \mathbb{R}^{3\times 1}$ are the orientation, velocity, and position states of the left foot of an agent expressed in the navigation frame. $\mathbf{b}_{\mathrm{g,L},k}$ and $\mathbf{b}_{\mathrm{a,L},k} \in \mathbb{R}^{3\times 1}$ are the gyroscope and accelerometer biases along the three axes of the body frame of the IMU mounted on the left foot. $\mathrm{b}_{\mathrm{b,L},k}$ represents barometric altitude biases experienced by the left foot-mounted barometer. $\mathbf{q}_{\mathrm{R},k}$, $\mathbf{v}_{\mathrm{R},k}$, $\mathbf{p}_{\mathrm{R},k}$, $\mathbf{b}_{\mathrm{g,R},k}$, $\mathbf{b}_{\mathrm{a,R},k}$, and $\mathrm{b}_{\mathrm{b,R},k}$ represent the orientation, velocity, position, gyroscope bias, accelerometer bias, and barometric altitude bias states associated with the right foot. $\mathbf{p}_{\mathrm{B}_1,k}, \ldots, \mathbf{p}_{\mathrm{B}_N,k} \in \mathbb{R}^{3\times 1}$ represent the position of the N beacons being deployed.

This section denotes the states computed in the EKF prediction step, discussed in Section 9.3.1, as $\mathbf{x}_{k|k-1}$, and the states calculated in the EKF update step, discussed in Section 9.3.1,

as $\mathbf{x}_{k|k}$. We denoted $\mathbf{P}_{k|k-1} \in \mathbb{R}^{(32+3N)\times(32+3N)}$ as the *a priori* estimate covariance matrix that includes estimated accuracy of the states $\mathbf{x}_{k|k-1}$ and $\mathbf{P}_{k|k} \in \mathbb{R}^{(32+3N)\times(32+3N)}$ as the *a posteriori* estimate covariance matrix that includes estimated accuracy of the states $\mathbf{x}_{k|k}$.

Filter Initialization At the beginning of a navigation task, we assume a pedestrian remains stationary for a short period of time. The initial roll angle, $\theta_{\mathrm{L},0}$, and pitch angle, $\phi_{\mathrm{L},0}$, of the left foot are expressed as follows:

$$\theta_{\mathrm{L},0} = \tan^{-1}(\frac{-\bar{\mathrm{a}}_{\mathrm{L},y}}{-\bar{\mathrm{a}}_{\mathrm{L},z}}), \phi_{\mathrm{L},0} = \tan^{-1}(\frac{\bar{\mathrm{a}}_{\mathrm{L},x}}{\sqrt{\bar{\mathrm{a}}^2_{\mathrm{L},y} + \bar{\mathrm{a}}^2_{\mathrm{L},z}}}),$$

where $\bar{\mathrm{a}}_{\mathrm{L},i}$ indicate the averaged readings collected by the ith-axis accelerometer of the left IMU during the initialization period. The initial roll angle, $\theta_{\mathrm{R},0}$, and pitch angle, $\phi_{\mathrm{R},0}$, of the right foot are computed similarly to the case of the left foot as follows:

$$\theta_{\mathrm{R},0} = \tan^{-1}(\frac{-\bar{\mathrm{a}}_{\mathrm{R},y}}{-\bar{\mathrm{a}}_{\mathrm{R},z}}), \phi_{\mathrm{R},0} = \tan^{-1}(\frac{\bar{\mathrm{a}}_{\mathrm{R},x}}{\sqrt{\bar{\mathrm{a}}^2_{\mathrm{R},y} + \bar{\mathrm{a}}^2_{\mathrm{R},z}}}),$$

where $\bar{\mathrm{a}}_{\mathrm{R},i}$ indicate the averaged readings collected by the ith-axis accelerometer of the right IMU during the initialization period. The initial yaw angles, $\psi_{\mathrm{L},0}$, $\psi_{\mathrm{R},0}$, can be determined with additional sensors, such as a magnetometer. The initial positions, $\mathbf{p}_{\mathrm{L},0}$ and $\mathbf{p}_{\mathrm{R},0}$, and velocities, $\mathbf{v}_{\mathrm{L},0}$ and $\mathbf{v}_{\mathrm{L},0}$, can be determined with external localization systems, such as GNSS modules or vision systems. In a case where relative positions are of interest, we assume that the initial yaw angles of the two feet are aligned with the north and initial positions and velocities are zeros. Accelerometer states, $\mathbf{b}_{\mathrm{a,L},0}$ and $\mathbf{b}_{\mathrm{a,R},0}$, are set to zeros. Initial gyroscope

biases, $\mathbf{b}_{\mathrm{g,L},0}$ and $\mathbf{b}_{\mathrm{g,R},0}$, are expressed as

$$\mathbf{b}_{\mathrm{g,L},0} = \begin{bmatrix} \bar{\omega}_{\mathrm{L},x} & \bar{\omega}_{\mathrm{L},y} & \bar{\omega}_{\mathrm{L},z} \end{bmatrix}^{\top},$$
$$\mathbf{b}_{\mathrm{g,R},0} = \begin{bmatrix} \bar{\omega}_{\mathrm{R},x} & \bar{\omega}_{\mathrm{R},y} & \bar{\omega}_{\mathrm{R},z} \end{bmatrix}^{\top},$$

where $\bar{\omega}_{\mathrm{L},i}$ and $\bar{\omega}_{\mathrm{R},i}$ indicates the averaged readings collected by the ith-axis gyroscope of the left and the right IMUs during the initialization period, respectively. The initial barometric altitude bias, $\mathrm{b}_{\mathrm{b,L},k}$ and $\mathrm{b}_{\mathrm{b,R},k}$, are both set to zeros.

When a seeded UWB with an identification (ID) number j is first connected to the foot-mounted UWB at time k, the developed UWB-Foot-SLAM2 sets the initial beacon positions, denoted as $\mathbf{p}_{\mathrm{B}_j,0}$, with the current estimated positions of the left foot-mounted IMU, $\mathbf{p}_{\mathrm{L},k}$. The choice of the left foot is based on a pedestrian navigation testbed, discussed later in Section 9.3.2, that seeded beacons are connected with the left foot-mounted UWB. The corresponding initial position uncertainties, $\sigma_{\mathbf{p}_{\mathrm{B}_j},0}$, are initialized with the size of a navigation environment. $\mathbf{p}_{\mathrm{B}_j,0}$ and $\sigma_{\mathbf{p}_{\mathrm{B}_j},0}$ are expressed as follows.

$$\mathbf{p}_{\mathrm{B}_j,0} = \mathbf{p}_{\mathrm{L},k}, \sigma_{\mathbf{p}_{\mathrm{B}_j},0} = \mathrm{D},$$

where D is the dimension of a navigation environment.

When a beacon's onboard IMU detects that the beacon is experiencing a motion, the developed UWB-Foot-SLAM2 resets the beacon's estimated positions and uncertainties to the initial values. The motion detection mechanism follows the AMV detector [181], where a motion is identified of the following inequality is violated:

$$T(\bar{\mathbf{y}}_k^{\alpha}) = \frac{1}{M} \sum_{k \in \Omega_n} \left(\frac{1}{\sigma_{\mathrm{VRW}}^2} \| \mathbf{y}_k^{\alpha} - \bar{\mathbf{y}}_k^{\alpha} \|^2 \right) < \gamma_{\mathrm{beacon}},$$

where $\mathbf{y}_k^{\alpha}$ is 3-axis accelerometer measurements of an event IMU at time k, $\Omega_n = \{l \in \mathbb{N}, n \leq$

$l < M-1\}$ is a collection of the sensor measurement indexes at time n with a window of length M, and γ_{beacon} is a pre-defined thresholds.

EKF Prediction Step

In the prediction step of the EKF, the states corresponding to the foot-mounted IMUs, including $\mathbf{q}_{\text{L},k|k-1}$, $\mathbf{v}_{\text{L},k|k-1}$, $\mathbf{p}_{\text{L},k|k-1}$, $\mathbf{b}_{\text{g,L},k|k-1}$, $\mathbf{b}_{\text{a,L},k|k-1}$, $\mathbf{q}_{\text{R},k|k-1}$, $\mathbf{v}_{\text{R},k|k-1}$, $\mathbf{p}_{\text{R},k|k-1}$, $\mathbf{b}_{\text{g,R},k|k-1}$, and $\mathbf{b}_{\text{a,R},k|k-1}$ are propagated with the strapdown INS algorithm [198]. The barometric altitude biases are assumed unchanged, and the position states of the beacons are assumed constant. The linearized continuous-time EKF propagation matrix, denoted by $\mathbf{A}_k$, is expressed as follows

$$\mathbf{A}_k = \begin{bmatrix} \mathbf{A}_{\text{Foot,L},k} & \mathbf{0}_{16\times 16} & \mathbf{0}_{16\times 3N} \\ \mathbf{0}_{16\times 16} & \mathbf{A}_{\text{Foot,R},k} & \mathbf{0}_{16\times 3N} \\ \mathbf{0}_{3N\times 16} & \mathbf{0}_{3N\times 16} & \mathbf{0}_{3N\times 3N} \end{bmatrix},$$

$$\mathbf{A}_{\text{Foot,L},k} = \begin{bmatrix} \mathbf{A}_{\text{INS,L},k} & \mathbf{0}_{15\times 1} \\ \mathbf{0}_{1\times 15} & 0 \end{bmatrix}, \mathbf{A}_{\text{Foot,R},k} = \begin{bmatrix} \mathbf{A}_{\text{INS,R},k} & \mathbf{0}_{15\times 1} \\ \mathbf{0}_{1\times 15} & 0 \end{bmatrix},$$

where

$$\mathbf{A}_{\text{INS,L},k} = \begin{bmatrix} \mathbf{0}_{3\times 3} & \mathbf{0}_{3\times 3} & \mathbf{0}_{3\times 3} & -\text{C}(\mathbf{q}_{\text{L},k|k-1}) & \mathbf{0}_{3\times 3} \\ [\overrightarrow{f}^{\,n}_{\text{L},k}\times] & \mathbf{0}_{3\times 3} & \mathbf{0}_{3\times 3} & \mathbf{0}_{3\times 3} & \text{C}(\mathbf{q}_{\text{L},k|k-1}) \\ \mathbf{0}_{3\times 3} & \mathbf{I}_{3\times 3} & \mathbf{0}_{3\times 3} & \mathbf{0}_{3\times 3} & \mathbf{0}_{3\times 3} \\ \mathbf{0}_{3\times 3} & \mathbf{0}_{3\times 3} & \mathbf{0}_{3\times 3} & \mathbf{0}_{3\times 3} & \mathbf{0}_{3\times 3} \\ \mathbf{0}_{3\times 3} & \mathbf{0}_{3\times 3} & \mathbf{0}_{3\times 3} & \mathbf{0}_{3\times 3} & \mathbf{0}_{3\times 3} \end{bmatrix},$$

and

$$\mathbf{A}_{\mathrm{INS,R},k} = \begin{bmatrix} \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & -\mathrm{C}(\mathbf{q}_{\mathrm{R},k|k-1}) & \mathbf{0}_{3\times3} \\ [\overrightarrow{f}^{\,n}_{\mathrm{R},k}\times] & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathrm{C}(\mathbf{q}_{\mathrm{R},k|k-1}) \\ \mathbf{0}_{3\times3} & \mathbf{I}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \end{bmatrix}.$$

Here, $[\overrightarrow{f}^{\,n}\times]$ is the skew-symmetric cross-product-operator of the accelerometer outputs of the IMU, expressed in the navigation frame. $\mathrm{C}(\mathbf{q})$ is the DCM corresponding to the quaternion vector $\mathbf{q}$. $\mathbf{0}_{n\times m}$ indicates a zero matrix having n number of rows and m number of columns. The continuous-time propagation matrix, $\mathbf{A}_k$, is converted to a discrete-time propagation matrix, denoted as, $\mathbf{F}_k$, as follows

$$\mathbf{F}_k = \exp\left(\mathbf{A}_k \mathrm{dt}\right),$$

where dt is the sampling period of an IMU. The process noise matrix, denoted as $\mathbf{Q}_k$, is expressed as

$$\mathbf{Q}_k = \begin{bmatrix} \mathbf{Q}_{\mathrm{Foot,L},k} & \mathbf{0}_{16\times16} & \mathbf{0}_{16\times3N} \\ \mathbf{0}_{16\times16} & \mathbf{Q}_{\mathrm{Foot,R},k} & \mathbf{0}_{16\times3N} \\ \mathbf{0}_{3N\times16} & \mathbf{0}_{3N\times16} & \mathbf{0}_{3N\times3N} \end{bmatrix},$$

$$\mathbf{Q}_{\mathrm{Foot,L},k} = \begin{bmatrix} \mathbf{Q}_{\mathrm{INS,L},k} & \mathbf{0}_{15\times1} \\ \mathbf{0}_{1\times15} & \sigma_{\mathrm{b}}^2 \end{bmatrix},$$

$$\mathbf{Q}_{\mathrm{Foot,R},k} = \begin{bmatrix} \mathbf{Q}_{\mathrm{INS,R},k} & \mathbf{0}_{15\times1} \\ \mathbf{0}_{1\times15} & \sigma_{\mathrm{b}}^2 \end{bmatrix},$$

where σ_{b}^2 is the process noise variance associated with barometer bias states of the two feet and

$$\mathbf{Q}_{\mathrm{INS,L},k} = \mathbf{Q}_{\mathrm{INS,R},k} = \begin{bmatrix} \sigma_{\mathrm{ARW}}^2\mathbf{I}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \sigma_{\mathrm{VRW}}^2\mathbf{I}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{I}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \sigma_{\mathrm{AcRW}}^2\mathbf{I}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \sigma_{\mathrm{RRW}}^2\mathbf{I}_{3\times3} \end{bmatrix}.$$

Here, $\mathbf{I}_{n\times n}$ is the identity matrix having n number of rows and columns. σ_{ARW}^2, σ_{VRW}^2, σ_{RRW}^2, and σ_{AcRW}^2 are the ARW of the gyroscopes, the VRW of the accelerometers, the RRW of the gyroscopes, and the AcRW of the accelerometers of the foot-mounted IMUs. This section assumes that the two foot-mounted IMUs have identical noise characteristics.

At each iteration of the EKF prediction step, the *a priori* estimate covariance matrix of time k, $\mathbf{P}_{k|k-1}$, is computed as follows

$$\mathbf{P}_{k|k-1} = \mathbf{F}_k\mathbf{P}_{k-1|k-1}\mathbf{F}_k^\top + \mathbf{Q}_k,$$

where $\mathbf{P}_{k-1|k-1}$ is the *a posteriori* estimate covariance matrix at time $k-1$, computed in the previous EKF update step. The update step is discussed in Section 9.3.1.

EKF Update Step

The update step of the EKF uses five different types of measurements, including pseudo measurements of zero-velocity, barometric altimeters, barometric altitude biases, foot-to-foot ranging, and foot-to-beacon ranging. The measurements of zero-velocity, barometric altimeters, barometric altitude biases, and foot-to-foot ranging are implemented for both the left and right feet of the foot-mounted localization system, and the foot-to-beacon ranging

measurements are used for all involved beacons.

Zero-velocity Updates When a stance phase is detected, the ZUPT algorithm is activated to compensate for the velocity states corresponding to the stationary feet in the update step of the EKF. Pseudo-measurements of zero velocity along the three axes, denoted as $\mathbf{v}_{\text{ZUPT}} = \mathbf{0}_{3\times1}$ are feedback to the EKF. The stance phase detection can be formulated as a hypothesis testing mechanism with input from different sensors [93, 88, 181, 223] or machine-learning-based approaches with different mathematical models [178, 202]. This section adapts the SHOE detector [181], which determines a stance phase if a test statistics, $T(\mathbf{u}_n) =$

$$\frac{1}{M}\sum_{k\in\Omega_n}\left(\frac{1}{\sigma^2_{\text{VRW}}}\parallel \mathbf{y}^{\alpha}_k - g\frac{\bar{\mathbf{y}}^{\alpha}_k}{\parallel \bar{\mathbf{y}}^{\alpha}_k \parallel}\parallel^2 + \frac{1}{\sigma^2_{\text{ARW}}}\parallel \mathbf{y}^{\omega}_k \parallel^2\right) < \gamma,$$

where $\mathbf{u}_n = \{\mathbf{y}_k\}_{k=n}^{k=N-1}$ with $\mathbf{y}_k = [\mathbf{y}^{\alpha}_k, \mathbf{y}^{\omega}_k]^{\top}$, $\mathbf{y}^{\alpha}_k$ is 3-axis accelerometer measurements at time k, $\mathbf{y}^{\omega}_k$ is 3-axis gyroscope measurements at time k, g is the gravitational constant, $\Omega_n = \{l \in \mathbb{N}, n \leq l < M-1\}$ is a collection of the sensor measurement indexes at time n with a window of length M, and γ are user-defined thresholds.

The ZUPT measurement models, $\mathbf{z}_{\text{ZUPT,L},k}$ and $\mathbf{z}_{\text{ZUPT,R},k}$, measurement matrices, $\mathbf{H}_{\text{ZUPT,L},k}$ and $\mathbf{H}_{\text{ZUPT,R},k}$, and measurement covariance matrices, $\mathbf{R}_{\text{ZUPT,L},k}$ and $\mathbf{R}_{\text{ZUPT,R},k}$, of the two feet are expressed as follows:

$$\begin{aligned}
&\mathbf{z}_{\text{ZUPT,L},k} = \mathbf{v}_{\text{ZUPT},k}, \mathbf{z}_{\text{ZUPT,R},k} = \mathbf{v}_{\text{ZUPT},k}\\
&\mathbf{H}_{\text{ZUPT,L},k} = \begin{bmatrix}\mathbf{0}_{6\times3} & \mathbf{I}_{3\times3} & \mathbf{0}_{(23+3N)\times3}\end{bmatrix}\\
&\mathbf{H}_{\text{ZUPT,R},k} = \begin{bmatrix}\mathbf{0}_{19\times3} & \mathbf{I}_{3\times3} & \mathbf{0}_{(10+3N)\times3}\end{bmatrix}\\
&\mathbf{R}_{\text{ZUPT,L},k} = \mathbf{R}_{\text{ZUPT,R},k} = \sigma^2_{\text{ZUPT}}\mathbf{I}_{3\times3},
\end{aligned}$$

where σ^2_{ZUPT} is the noise variance of the zero-velocity measurement $\mathbf{v}_{\text{ZUPT}}$. It should be

noted that the variance could be different when mounting an IMU at different positions and performing different activities [178, 153, 214, 218].

Barometric Altimeter Barometric altimeter measurements are used to provide compensation along the vertical displacements [91, 92]. A COTS electronic barometer usually has a sampling rate much lower than a COTS IMU. This section feedbacks barometer measurements to the EKF only when the measurements become available. A barometer measures air pressure, denoted as P_k. This measurement can be converted to accurate altitude estimates above sea level, h, when ambient air pressure is stable [154], as follows:

$$h = -\frac{RT_0}{gM} \ln \frac{P_k}{P_0}, \tag{9.1}$$

where g is the gravitational constant, R is the universal gas constant, T_0 is temperature, M is molar mass of Earth's air, and $P_0 = 1013.25$ [mbar] is the standard pressure at sea level. This section takes into account that the estimated height directly derived from a single barometer could be affected by variations in baseline air pressure and local pressure fluctuation due to opening doors/windows or air conditioning control in a building. These effects are considered barometric altitude biases.

At timestamp k, altimeters mounted on the left and the right shoes provide measurements of vertical displacements in the navigation frame, which are denoted as $d_{\perp,\mathrm{L},k}$ and $d_{\perp,\mathrm{R},k}$, respectively. The measurement models of the EKF corresponding to the altimeter on the left and right feet, $z_{\mathrm{ALT,L},k}$ and $z_{\mathrm{ALT,R},k}$, are described as follows:

$$z_{\mathrm{ALT,L},k} = d_{\perp,\mathrm{L},k} = p_{\mathrm{L,D},k} + b_{\mathrm{b,L},k},$$
$$z_{\mathrm{ALT,R},k} = d_{\perp,\mathrm{R},k} = p_{\mathrm{R,D},k} + b_{\mathrm{b,R},k},$$

where $p_{\mathrm{L,D},k}$ and $p_{\mathrm{L,D},k}$ are EKF states corresponding to the positions of the left and right

feet along the down direction in the navigation frame. The associated measurement matrices are described as

$$\mathbf{H}_{\mathrm{ALT,L},k} = \begin{bmatrix} \mathbf{0}_{1\times 8} & 1 & \mathbf{0}_{1\times 6} & 1 & \mathbf{0}_{1\times(16+3N)} \end{bmatrix}$$
$$\mathbf{H}_{\mathrm{ALT,R},k} = \begin{bmatrix} \mathbf{0}_{1\times 24} & 1 & \mathbf{0}_{1\times 6} & 1 & \mathbf{0}_{1\times 3N} \end{bmatrix}.$$

The measurement noise covariance matrices are described as

$$\mathbf{R}_{\mathrm{ALT,L},k} = \mathbf{R}_{\mathrm{ALT,R},k} = \sigma^2_{\mathrm{ALT}},$$

where σ^2_{ALT} is the noise variance of the foot-mounted altimeter measurements.

Differential Barometer Reference barometers integrated into beacons are used to measure barometric altitude biases for both the left and right feet. These measurements collected from the ith beacon, denoted as $P_{\mathrm{ref}_i,k}$, become available when the foot-mounted UWBs are within the LoS connection with the seeded UWB. Altitude estimates computed using (9.1) based on $P_{\mathrm{ref}_i,k}$, are denoted as $h_{\mathrm{ref}_i,k}$. When beacons are stationary, variations in $h_{\mathrm{ref}_i,k}$ are considered barometric altitude biases. The EKF measurement model corresponding to the barometric altitude biases is expressed as follows:

$$z_{\mathrm{ALT}_i,k} = h_{\mathrm{ref}_i,k} - h_{\mathrm{ref}_i,0} = b_{\mathrm{b,L},k} = b_{\mathrm{b,R},k},$$

where $h_{\mathrm{ref}_i,0}$ is the first attitude measurement collected by the reference barometer on the ith beacon. Corresponding measurement matrices $\mathbf{H}_{\mathrm{b,L},k}$ and $\mathbf{H}_{\mathrm{b,R},k}$ are expressed as follows:

$$\mathbf{H}_{\mathrm{b,L},k} = \begin{bmatrix} \mathbf{0}_{1\times 15} & 1 & \mathbf{0}_{1\times(16+3N)}, \end{bmatrix}$$
$$\mathbf{H}_{\mathrm{b,R},k} = \begin{bmatrix} \mathbf{0}_{1\times 31} & 1 & \mathbf{0}_{1\times 3N} \end{bmatrix},$$

and the measurement noise matrices are expressed as follows:

$$\mathbf{R}_{\mathrm{b,L},k} = \mathbf{R}_{\mathrm{b,R},k} = \sigma^2_{\mathrm{ALT}_i},$$

where $\sigma^2_{\mathrm{ALT}_i}$ is the measurement noise variance of a reference barometer.

Foot-to-foot Range Augmentation Distance measurements between the two feet, denoted as $r_{\mathrm{F2F},k}$, can be obtained from various different sensing modalities, including ultrasonic sensors [213], foot-to-foot cameras [94], electromagnetic systems [222], and UWB [263, 32]. In this section, UWB-based foot-to-foot ranging measurements are used. The foot-to-foot range measurements are classified into LoS and NLoS by a power metric-based approach [264]. In this section, only LoS UWB measurements are used. The range measurements are processed with bias correction. The processed foot-to-foot measurements are fused in the update step of the EKF to compensate for relative distances between the two feet [117]. The corresponding measurement model, $z_{\mathrm{F2F},k}$, measurement matrix, $\mathbf{H}_{\mathrm{F2F},k}$, and measurement noise covariance matrices, $\mathbf{R}_{\mathrm{F2F},k}$, are described as follows:

$$z_{\mathrm{F2F},k} = r_{\mathrm{F2F},k} = ||\mathbf{p}_{\mathrm{L},k} - \mathbf{p}_{\mathrm{R},k}||$$

$$\mathbf{H}_{\mathrm{F2F},k} = \begin{bmatrix} \mathbf{0}_{6\times 1} \\ \frac{\partial ||\mathbf{p}_{\mathrm{L},k} - \mathbf{p}_{\mathrm{R},k}||^\top}{\partial \mathbf{p}_{\mathrm{L},k}} \\ \mathbf{0}_{16\times 1} \\ \frac{\partial ||\mathbf{p}_{\mathrm{L},k} - \mathbf{p}_{\mathrm{R},k}||^\top}{\partial \mathbf{p}_{\mathrm{R},k}} \\ \mathbf{0}_{(7+3N)\times 1} \end{bmatrix}^\top, \mathbf{R}_{\mathrm{F2F},k} = \sigma^2_{\mathrm{F2F}},$$

where σ^2_{F2F} is the noise variance of the UWB foot-to-foot range measurements.

UWB Foot-to-beacon Range Updates We denoted the foot-to-beacon range measurements, derived from wireless communication between a foot-mounted UWB and a seeded UWB on the ith beacon, as $r_{\mathrm{UWB}_i,k}$. When the range measurement becomes available, the measurements are first classified into LoS and NLoS cases, and only LoS cases are used in the update step of the EKF. This section uses a probabilistic power metric approach to differentiate LoS and NLoS UWB measurements [32]. The LoS measurements are further processed with bias correction through a curve-fitting approach. The corresponding foot-to-beacon range measurement model, $z_{\mathrm{UWB}_i,k}$, measurement matrix, $\mathbf{H}_{\mathrm{UWB}_i,k}$, and measurement noise matrix, $\mathbf{R}_{\mathrm{UWB}_i,k}$, are described as follows:

$$z_{\mathrm{UWB}_i,k} = r_{\mathrm{UWB}_i,k} = ||\mathbf{p}_{\mathrm{L},k} - \mathbf{p}_{\mathrm{B}_i,k}||$$

$$\mathbf{H}_{\mathrm{UWB}_i,k} = \begin{bmatrix} \mathbf{0}_{6\times 1} \\ \frac{\partial ||\mathbf{p}_{\mathrm{L},k} - \mathbf{p}_{\mathrm{B}_i,k}||^\top}{\partial \mathbf{p}_{\mathrm{L},k}} \\ \mathbf{0}_{(23+3(i-1))\times 1} \\ \frac{\partial ||\mathbf{p}_{\mathrm{L},k} - \mathbf{p}_{\mathrm{B}_i,k}||^\top}{\partial \mathbf{p}_{\mathrm{B}_i,k}} \\ \mathbf{0}_{3(N-i-1)\times 1} \end{bmatrix}^\top, \mathbf{R}_{\mathrm{UWB}_i,k} = \sigma^2_{\mathrm{UWB}_i},$$

where $\sigma^2_{\mathrm{UWB}_i}$ is the noise variance of the foot-to-beacon range measurements between the foot-mounted UWB and the ith UWB beacon. Note that the left foot-mounted UWB was chosen to be paired with seeded UWBs. This is a convenient choice based on a pedestrian navigation testbed that will be discussed later in Section 9.3.2.

Update Step Summary In the EKF update step, when each of the sensing modalities discussed previously becomes available, the EKF stacks all available measurements and performs the update step. For example, in a case where two feet are stationary, altimeter measurements are available, and the 1st and 2nd seeded UWBs are connected to the left foot-mounted UWB, the EKF measurement model, z_k, measurement matrix, $\mathbf{H}_k$, and measurement noise

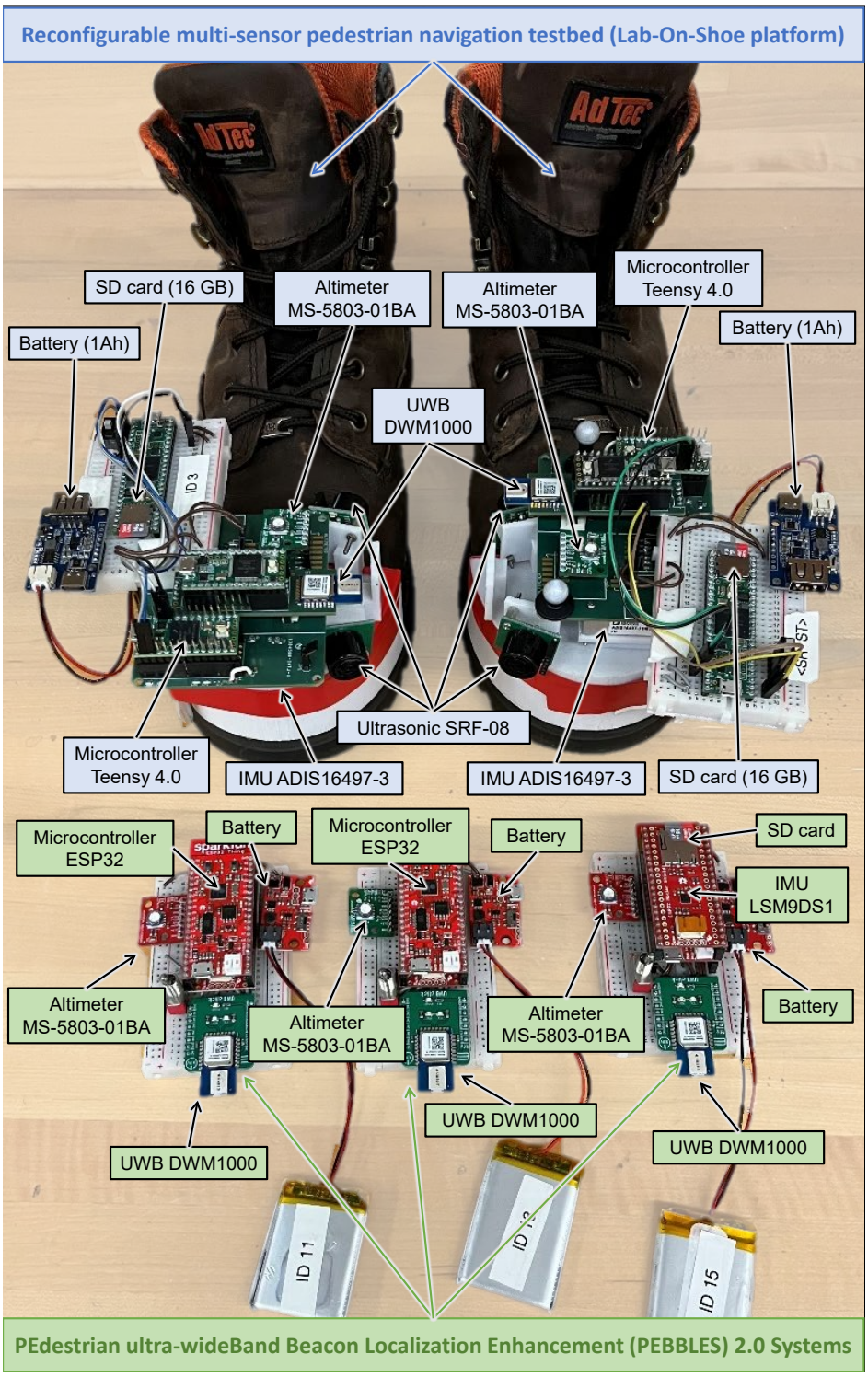


Figure 9.12: A picture showing prototypes of a Lab-On-Shoe platform and three beacons of a PEBBLE 2.0 system. Firmware implementation of the two systems are discussed in Figure 9.13 and Figure 9.14.

covariance matrix, $\mathbf{R}_k$, are expressed as follows:

$$z_k = \begin{bmatrix} \mathbf{z}_{\mathrm{ZUPT,L},k} \\ \mathbf{z}_{\mathrm{ZUPT,R},k} \\ z_{\mathrm{ALT,L},k} \\ z_{\mathrm{ALT,R},k} \\ z_{\mathrm{ALT}_1,k} \\ z_{\mathrm{ALT}_2,k} \\ z_{\mathrm{F2F},k} \\ z_{\mathrm{UWB}_1,k} \\ z_{\mathrm{UWB}_2,k} \end{bmatrix}, \mathbf{H}_k = \begin{bmatrix} \mathbf{H}_{\mathrm{ZUPT,L},k} \\ \mathbf{H}_{\mathrm{ZUPT,R},k} \\ \mathbf{H}_{\mathrm{ALT,L},k} \\ \mathbf{H}_{\mathrm{ALT,R},k} \\ \mathbf{H}_{\mathrm{ALT}_1,k} \\ \mathbf{H}_{\mathrm{ALT}_2,k} \\ \mathbf{H}_{\mathrm{F2F},k} \\ \mathbf{H}_{\mathrm{UWB}_1,k} \\ \mathbf{H}_{\mathrm{UWB}_2,k} \end{bmatrix}, \mathbf{R}_k = \mathrm{blkdiag}(\begin{bmatrix} \mathbf{R}_{\mathrm{ZUPT,L},k} \\ \mathbf{R}_{\mathrm{ZUPT,R},k} \\ \mathbf{R}_{\mathrm{ALT,L},k} \\ \mathbf{R}_{\mathrm{ALT,R},k} \\ \mathbf{R}_{\mathrm{ALT}_1,k} \\ \mathbf{R}_{\mathrm{ALT}_2,k} \\ \mathbf{R}_{\mathrm{F2F},k} \\ \mathbf{R}_{\mathrm{UWB}_1,k} \\ \mathbf{R}_{\mathrm{UWB}_2,k} \end{bmatrix})$$

Settings of the noise parameters are determined based on noise characteristics of sensors involved in the hardware implementation. TABLE 9.2 shows the EKF parameters used in this section.

Table 9.2: Parameters for the EKF

Hyper-parameter	**Value**
σ_{ARW}	2.7221×10^{-5}
σ_{VRW}	0.0017
σ_{RRW}	8.3174×10^{-7}
σ_{AcRW}	6.63×10^{-6}
σ_{b}	2.2×10^{-5}
σ_{ZUPT}	0.02
σ_{ALT}	0.3
σ_{F2F}	0.1
σ_{ALT_i}	0.3
σ_{UWB_i}	0.5

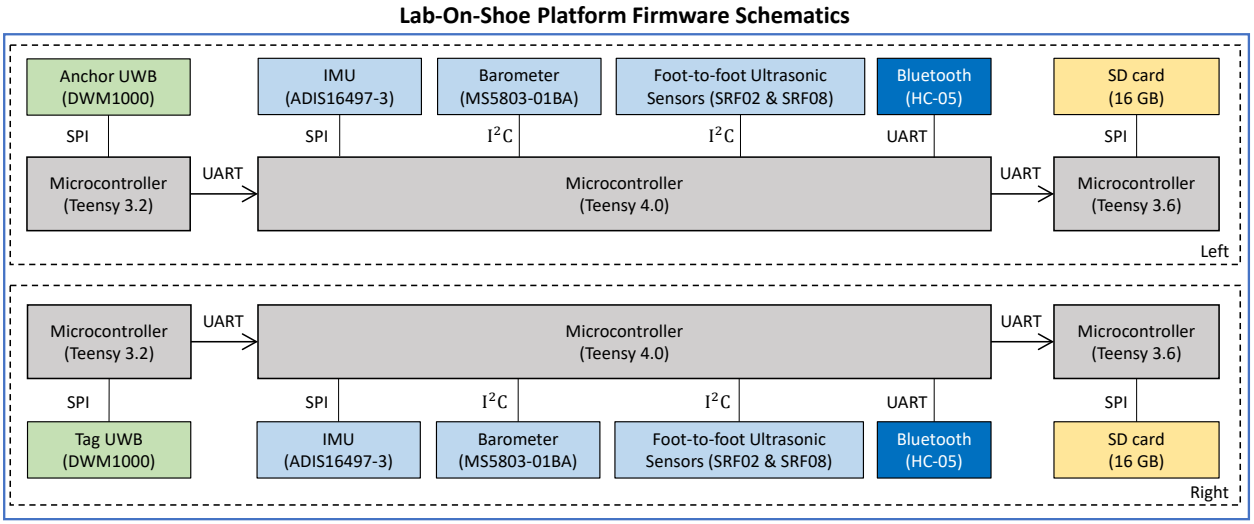


Figure 9.13: A block diagram illustrating firmware implemented on the Lab-On-Shoe platform shown in Figure 9.12.

At each iteration of the EKF update step, the state, $\mathbf{x}_{k|k}$ and the *a posteriori* estimate covariance matrix at time k, $\mathbf{P}_{k|k}$, are updated as follows

$$\mathbf{x}_{k|k} = \mathbf{x}_{k|k-1} + \mathbf{K}_k(\mathbf{z}_k - \mathbf{H}_k\mathbf{x}_{k|k-1})$$

$$\mathbf{P}_{k|k} = (\mathbf{I} - \mathbf{K}_k\mathbf{H}_k)\mathbf{P}_{k|k-1}$$

where $\mathbf{x}_{k|k-1}$ and $\mathbf{P}_{k|k-1}$ are the state and the *a priori* estimate covariance matrix calculated in the EKF prediction step discussed in Section 9.3.1, $\mathbf{I}$ is the identity matrix having the same dimension as $\mathbf{P}_{k|k}$, and $\mathbf{K}_k$ is the Kalman gain computed as

$$\mathbf{K}_k = \mathbf{P}_{k|k-1}\mathbf{H}_k^\top(\mathbf{H}_k\mathbf{P}_{k|k-1}\mathbf{H}_k^\top + \mathbf{R}_k)^{-1}.$$

9.3.2 System Design

To realize the developed UWB-Foot-SLAM2 algorithm, we developed a reconfigurable multi-sensor pedestrian navigation testbed, referred to as the Lab-On-Shoe platform, and multiple integrated UWB beacon units, referred to as PEdestrian ultra-wideBand Beacon Localization Enhancement (PEBBLE) 2.0 systems. This section discusses both the hardware and firmware implementation of the Lab-On-Shoe platform and PEBBLE 2.0 prototypes.

Hardware Implementation

Figure 9.12 presents the developed experimental prototypes. The Lab-On-Shoe platform was previously developed as a flexible hardware testbed at the Microsystem Laboratory at the University of California, Irvine, with the purpose of investigating sensor fusion solutions for integrated pedestrian inertial navigation system [83, 18, 82, 89]. On each shoe, we integrated

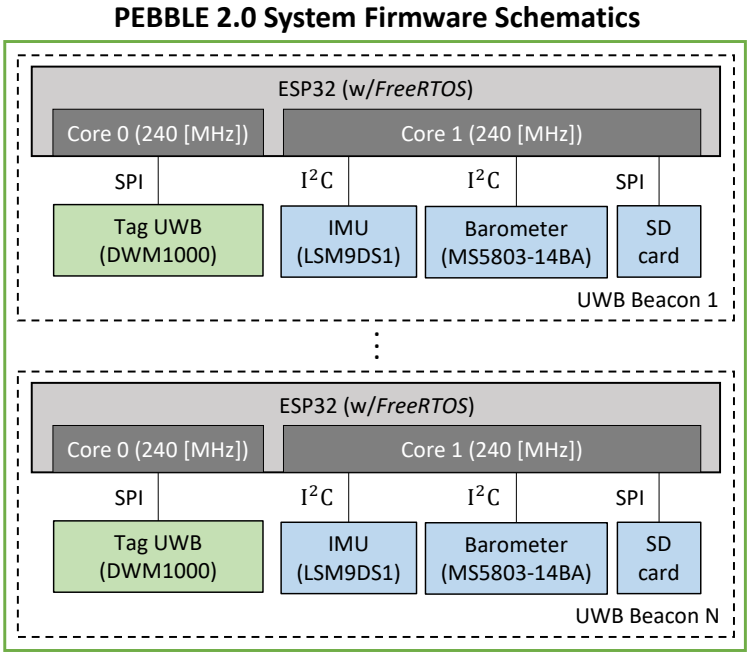


Figure 9.14: A block diagram illustrating firmware implemented on each beacon in the PEBBLE 2.0 system shown in Figure 9.12.

Table 9.3: Summary of navigation performance of a standalone ZUPT-aided INS and the original UWB-Foot-SLAM algorithm in the experiments discussed in Section 9.3.3.

Trial	**Trajectory profile**			**ZUPT-aided INS**			**UWB-Foot-SLAM**				
	Shape	Duration [s]	Length [m]	Localization error [m]			Localization error [m]			Mapping error [m]	
				RMSE	Max	LCE	RMSE	Max	LCE	Beacon #1	Beacon #2
#1	Square	160	110	0.16	0.38	0.32	0.15	0.32	0.30	0.28	0.25
#2	Straight line	210	180	0.19	0.50	0.30	0.18	0.45	0.27	0.36	0.20
#3	Figure 8	500	430	0.17	0.47	0.47	0.15	0.37	0.37	0.36	0.21

multiple COTS components, including three Teensy micro-controllers, an Analog Device IMUs ADIS16497−3, a barometric altimeter MS−5803−01BA, two ultrasonic range sensor SRF02, a UWB module DWM1000, and a SD card. The three Teensy microcontrollers are Teensy 4.0, Teensy 3.6, and Teensy 3.2, having CPU clock rates of 600 [MHz], 180 [MHz], and 96 [MHz], respectively. The SD card module was a built-in module on the Teensy 3.6 microcontroller. All the components were firmly mounted on a customized 3D-printed PLA fixture.

The PEBBLE 2.0 system is an upgraded version of the PEBBLE system described in [89] that was used to realize the original UWB-Foot-SLAM. Each beacon in the PEBBLE 2.0 system included a dual-core microcontroller ESP32, an LSM9DS1 IMU, an MS5303−01BA barometer, and a UWB module DWM1000. The Lab-On-Shoe platform and the PEBBLE 2.0 system were both powered up with 5.0-V lithium-ion batteries.

Firmware Implementation

Figure 9.13 and Figure 9.14 present block diagrams describing firmware schematics implemented on the Lab-On-Shoe platform and the PEBBLE 2.0 system, respectively. On the Lab-On-Shoe platform, the Teensy 3.2 microcontroller collected information on the connected node ID number and measurements of range, transmitter power, receiver first pulse power, and power metrics at a rate of 10 [Hz] from the DWM1000 UWB module via SPI

communication protocol. The DWM1000 module mounted on the left shoe was programmed to operate in the anchor mode, and the module mounted on the right shoe was in the tag mode. The collected measurements were transmitted to the Teensy 4.0 microcontroller via the UART communication protocol. On the Teensy 4.0 microcontroller, we implemented the SPI protocol to collect IMU measurements at a rate of 1000 [Hz] as well as the I2C communication protocol to collect pressure and thermal measurements from the MS5803−01BA barometer at a rate of 25 [Hz] and inter-foot ranging measurements from the two SRF02 ultrasonic sensors at a rate of 25 [Hz]. After all the sensor measurements were collected at each implementation loop, the Teensy 4.0 transmitted the measurements to the Teensy 3.6 micro-

Figure 9.15: Experimental scenario for the experiment discussed in Section 9.3.3. 42 OptiTrack motion capture cameras were mounted on the ceiling of a warehouse and obtain the ground truth position and orientation. Two beacons were placed on top of the orange barricades during the experiment. The operation range of the motion capture camera system was around 15 [m] $\times$ 15 [m].

controller via UART communication protocol. The Teensy 3.6 microcontroller implemented the SPI protocol to write all the received measurements to an SD card.

On the PEBBLE 2.0 system, the ESP32 microcontroller was programmed in the *FreeRTOS* framework with two cores, core0 and core1, each with a clock rate of 240 [MHz]. On core0, we implemented the SPI communication protocol to collect information on the connected node ID number and measurements of range, transmitter power, receiver first pulse power, and power metrics at a rate of 10 [Hz]. All the UWB modules in the developed PEBBLE 2.0 system were programmed to operate in the tag mode. A DWM1000 UWB operating in the tag mode can only be paired with a UWB operated in the anchor mode, and the range measurements between the two UWBs were obtained through a two-way ranging method. Therefore, all the UWB modules involved in the PEBBLE 2.0 system, when within a detectable range, were connected only to the UWB mounted on the left shoe of the Lab-On-Shoe platform. On core1, we implemented the I^2C communication protocol to collect data from the LSD9DS1 IMU and the ME5803−01BA barometer at a rate of 100 [Hz] and 25 [Hz]. All the collected data, including IMU, barometer, and UWB, are logged into an SD card at a rate of 100 [Hz] through the SPI protocol.

9.3.3 Experimental Validation

We conducted two experiments to validate the developed UWB-Foot-SLAM2 using the developed Lab-On-Shoe platform and the PEBBLE 2.0 system. The first experiment was conducted in a small indoor area on a single floor with ground truth measurements collected by high-accuracy motion capture cameras. The second experiment involved testing in a large multi-floor building for a long duration of time.

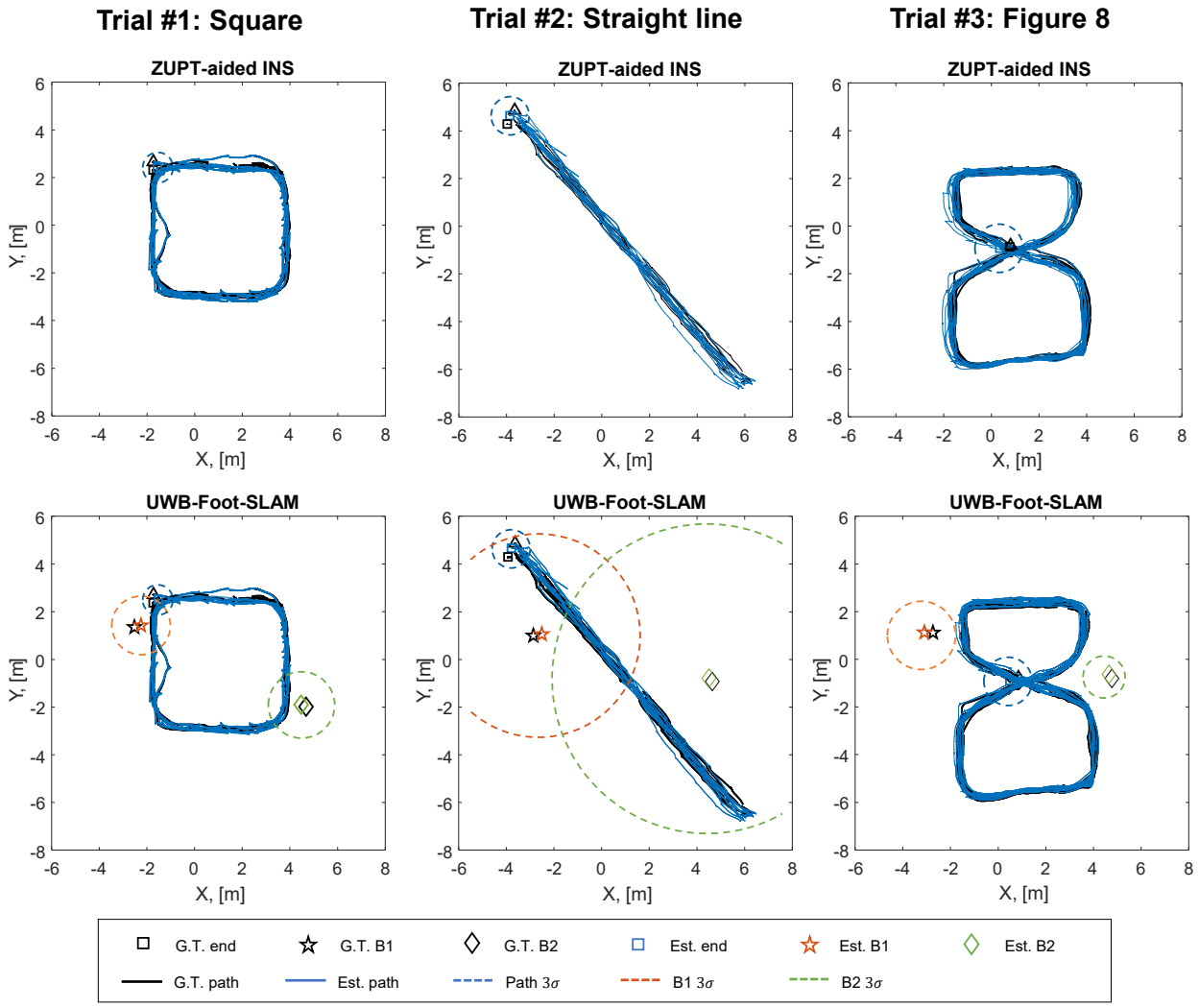


Figure 9.16: Estimated (Est.) Navigation solutions computed with a standalone ZUPT-aided INS and the original UWB-Foot-SLAM in the experiment discussed in Section 9.3.3. Items colored in black correspond to the Ground Truth (G.T.) collected by motion capture cameras. It could be seen that the estimated and ground truth trajectories have small discrepancies. Quantitative evaluation of the estimated solutions is summarized in TABLE 9.3. The radius of each dashed circle represents three times the position standard deviation predicted by the EKF at the end of the experiments. Positions of Beacon #1 (B1) and beacon #2 (B2) are marked with star and diamond symbols, respectively.

Scenarios #1: A Small Area With Reference Motion Capture Cameras

Experimental Scenario The first experiment was conducted to extensively evaluate the original UWB-Foot-SLAM discussed in [89] and investigate the repeatability of the navigation performance. A subject was equipped with the Lab-On-Shoe platform and carried two

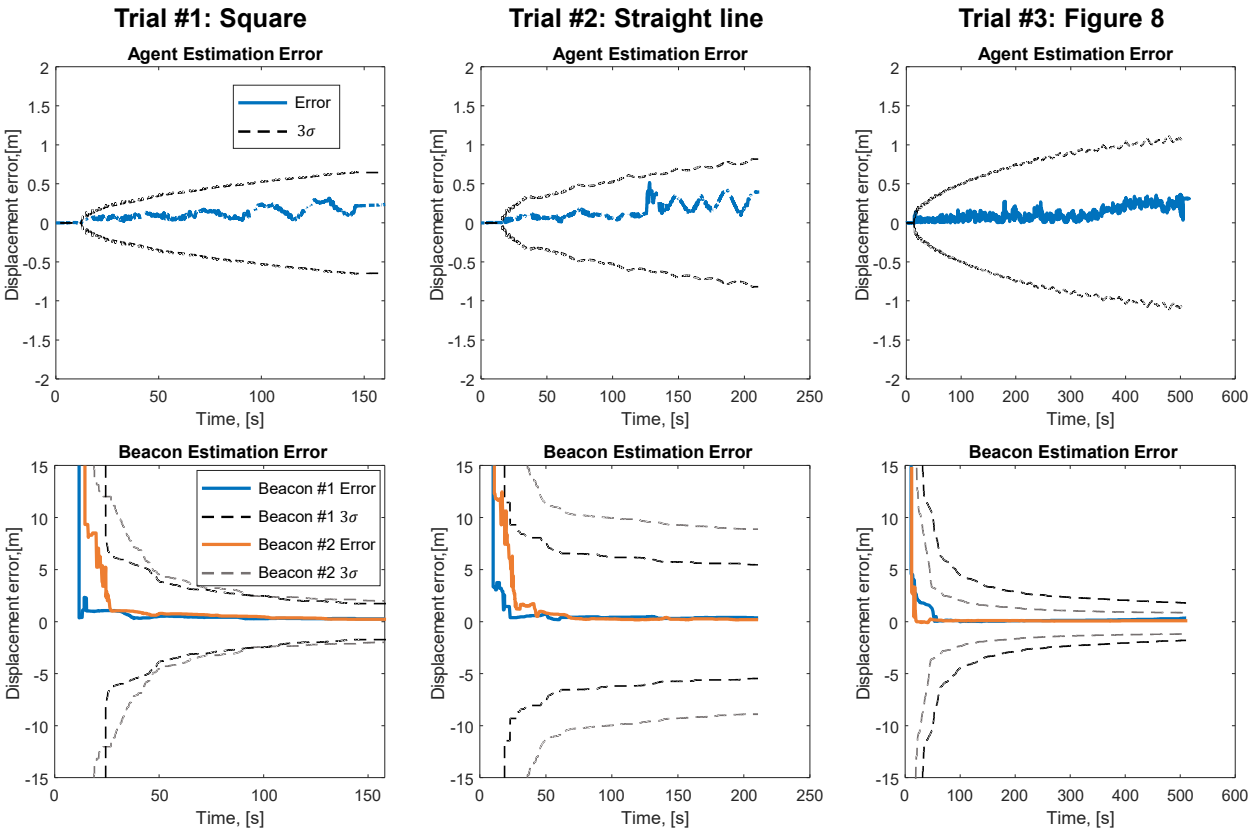


Figure 9.17: Position estimates and its associated covariances of the original UWB-Foot-SLAM algorithm in the experiment discussed in Section 9.3.3. It could be observed that the covariances of the agent's positions increased over time while the covariances of the beacons' positions were reduced. At the end of this experiment, the covariances of the agent's positions were still less than that of the beacons' locations. It could also be observed that the patterns of a subject's trajectories have impacts on the mapping performance.

beacons. Figure 9.15 shows the experimental scenario. The subject performed three different trials consisting of close-loop trajectories with distinct patterns. TABLE 9.3 summarizes details of the experimental processes. At the beginning of each trial, the subject stood still at the origin for 10 seconds. Two beacons, denoted as beacon #1 and beacon #2, were deployed at the beginning of the experiment. The first LoS range measurements of beacon #1 were collected at the 15 [s] timestamp, and the first LoS measurements of beacon #2 were collected at the 20 [s] timestamp. When the two beacons were both within connection range, their range measurements were processed sequentially. As a result, each iteration of the EKF only processed one beacon at a time.

OptiTrack motion capture cameras shown in Figure 9.15 were used to obtain the ground truth positions of the two beacons and the subject's feet. The sampling rate of the camera system was 120 [Hz]. In this experiment, among all the measurements produced by the Lab-On-Shoe platform and the PEBBLE 2.0 system, we only used the ones collected by left foot-mounted IMU, left foot-mounted UWB, and seeded UWBs.

Experimental Results We compared the navigation solutions computed by the original UWB-Foot-SLAM algorithm with a standalone ZUPT-aided INS. The initial yaw angle and positions used in the estimated solutions were aligned with the coordinate system of the motion capture cameras. Ground truth positions provided by the motion capture cameras are used to evaluate the accuracy of the estimated navigation solutions. We down-sampled our solutions from 1000 [Hz] to 120 [Hz] to align with the sampling rate of the motion capture cameras. 2D RMSEs, 2D final displacement errors, and 2D maximum displacement errors were used as performance metrics in this experiment.

Figure 9.16 presents the two navigation solutions with position uncertainties predicted by the EKF. Position uncertainties are expressed in terms of three times the Standard Deviation (3σ). The ZUPT-aided INS solution only estimated the positions of the agent, while the UWB-Foot-SLAM estimated both the agent's and beacons' positions. TABLE 9.3 summarizes the quantitative navigation performance of the ZUPT-aided INS and the original UWB-Foot-SLAM. The experimental results show, in all three trials, that the original UWB-Foot-SLAM algorithm had a smaller navigation error, as compared to the standalone ZUPT-aided INS, and could achieve a mapping error of less than 0.5 [m]. Figure 9.17 presents propagation profiles of the covariances associated with the position states of the subject and the beacons when using the original UWB-Foot-SLAM algorithm. The experimental results are discussed in the next subsection.

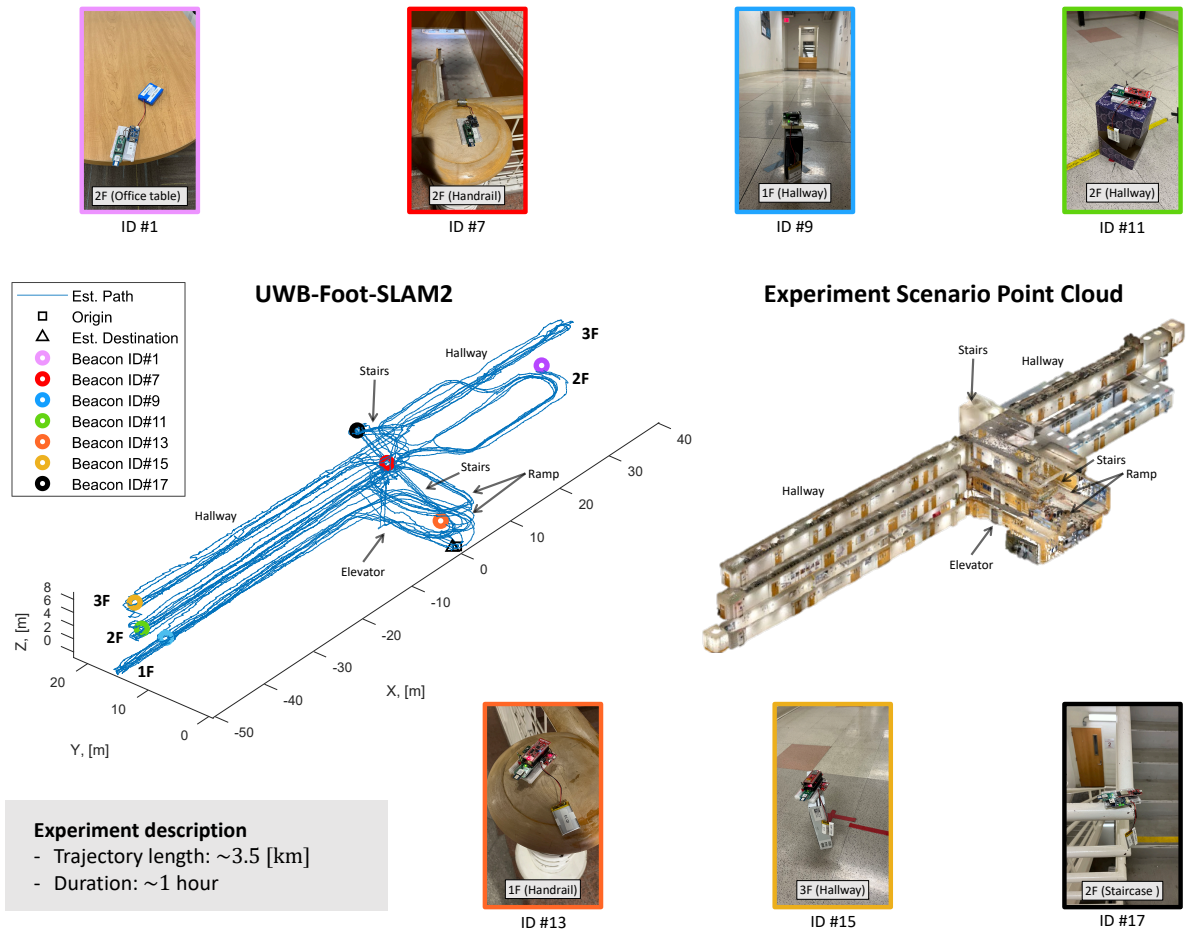


Figure 9.18: An illustration of the experimental scenario and process. Seven beacons were deployed during the experiment discussed in Section 9.3.3 at different indoor locations. The locations of the beacons shown in this figure were not pre-surveyed but estimated by our developed UWB-Foot-SLAM2 algorithm. The point cloud representation of the experimental scenario is used as a visual reference.

Discussion A few remarks can be made in this experiment:

- The foot trajectories generated by the OptiTrack motion capture system were discontinuous. When using the OptiTrack motion capture system, three retro-reflective markers were mounted on each shoe of the Lab-On-Shoe platform and each of the two beacons. We noticed multiple instances of discontinuities in the foot trajectories generated by the OptiTrack motion capture system. The discontinuities were caused by the fact that the markers mounted on the shoes were blocked by the subject's body

when performing normal walking activities.

- It could be observed in Figure 9.16 that the 3σ uncertainties of the two beacons at the end of each trial had different values. The values are lower than 5 [m] in the case of the square and the figure−8 shapes and larger than 5 [m] in the case of walking a straight line. In our opinion, this phenomenon is similar to the PDOP in other RF-signal-based positioning systems [116], and the differences are results of different observability of the EKF with respect to foot-to-beacon range measurements. In the case of a straight-line motion, using one-dimensional range measurements led to difficulties in uniquely determining the positions of the beacon along the directions perpendicular to the direction of the straight-line motion. To maximize the mapping performance of the original UWB-Foot-SLAM algorithm, it is suggested to have a variety of motions.

- We could see in Figure 9.17 that the uncertainties of the agent's positions follow an increasing trend over time while the beacons' position uncertainties decrease. In this experiment, the original UWB-Foot-SLAM was considered to operate in the mapping mode, as the agent's position uncertainties had not grown beyond the beacons' position uncertainties at the end of the experiment. Therefore, the UWB range measurements did not have significant numerical impacts on the agent's estimated positions.

- Following the previous bullet point, the propagation of the 3σ uncertainties corresponding to the subject's positions before the uncertainties are bounded behaves similarly to a standalone ZUPT-aided INS. An analytical expression of the position uncertainties was derived in [217].

- It could be seen in TABLE 9.3 that although both the duration and trajectory length of trial #2 were longer than trial #3, the RMSEs of the solution estimated either by the ZUPT-aided INS or the original UWB-Foot-SLAM were larger in trial #2 than in trial #3. The different sizes of the navigation environments cause this seeming contradiction. In trial #2, the subject walked in an area of around 10 [m] $\times$ 12 [m]

while the area in trial #3 was 6 [m] × 8 [m]. When navigating in a large area, the error sources of underestimated trajectory length and heading angle errors in a ZUPT-aided INS become the dominant sources.

Scenarios #2: A Large Area Including Multiple Floors

Experimental Scenario In the second experiment, the subject walked inside the engineering gateway building at the University of California, Irvine, on three different floors covering terrains of flat planes, stairs, ramps, and an elevator. The experimental scenario had a physical dimension of approximately 90 [m] by 25 [m] by 10 [m]. A point cloud representation of the experimental scenario is shown in Figure 9.18.

The subject started the experiment on the first floor of the building by standing still for around 10 seconds and then walked continuously along a similar closed-loop path for four rounds. Each round lasted around 15 minutes, and the traveling distance was around 900 [m]. The duration of the entire experiment was around 1 hour, and the total trajectory length was approximately 3.5 [km]. At the end of each round, the subject returned to the starting location to evaluate an LCE.

Seven beacons of the PEBBLE 2.0 system were deployed during the first round at different locations on the three floors of the building. Pictures of beacon deployment locations can be found in Figure 9.18. Beacons with ID #13, #9, #7, #11, #1, #17 and #15 were deployed at timestamps of 63 [s], 151 [s], 290 [s], 399 [s], 517 [s], 585 [s], and 786 [s], respectively. Beacons with ID #9 and #13 were deployed on the first floor, beacons with ID #1, #7, and #11 were deployed on the second floor, and beacons with ID #15 and #17 were deployed on the third floor.

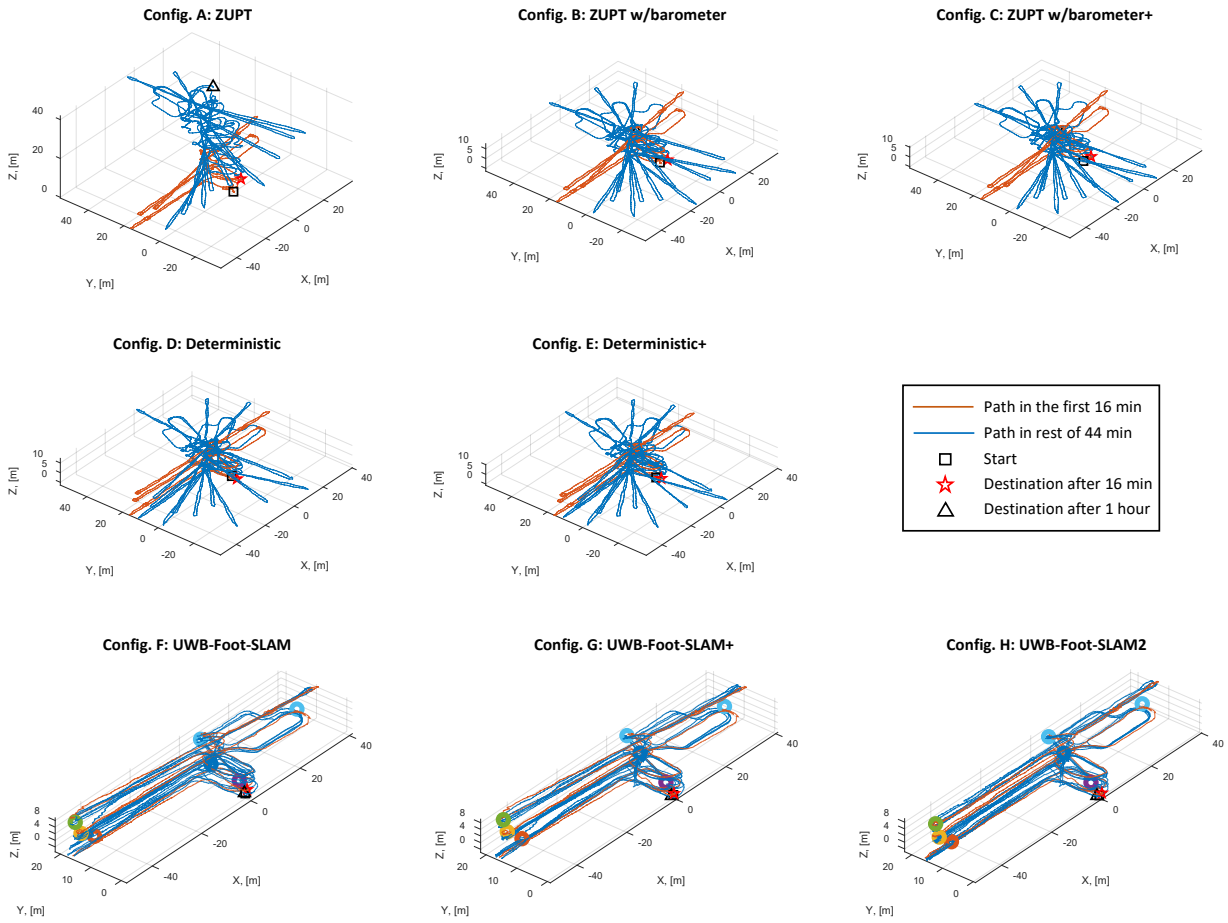


Figure 9.19: Navigation solutions using eight different algorithms in the experiment discussed in Section 9.3.3. The solutions listed from top left to bottom right are traditional ZUPT-aided INS (ZUPT), ZUPT-aided INS augmented with a barometric altimeter (ZUPT /w ALT), ZUPT-aided INS with a differential barometric altimeter (ZUPT w/ALT+), ZUPT-aided INS augmented with barometric altimeters and foot-to-foot ranging (Deterministic), ZUPT-aided INS augmented with differential barometric altimeters and foot-to-foot ranging (Deterministic+), the original UWB-Foot-SLAM, the original UWB-Foot-SLAM augmented with differential barometric altimeters (UWB-Foot-SLAM+), and the developed UWB-Foot-SLAM2. The orange trajectories represent the estimated positions of the first loop of the experiment, and the blue trajectories represent the rest of the estimated positions. The duration of the experiment was around 1 hour, and the trajectory length was around 3.5 [km]. The developed UWB-Foot-SLAM2 had the minimum 3D mean absolute error of 0.48 [m], equivalent to 0.013% traveling distance based on a 3.5-[km]-long trajectory. Quantitative evaluation of all the navigation solutions is presented in TABLE 9.4.

Experimental Results We compared the performance of eight different navigation solutions, including the traditional ZUPT-aided INS, ZUPT-aided INS augmented with a barometric altimeter (ALT), ZUPT-aided INS with a differential barometric altimeter (ALT+), ZUPT-

Table 9.4: Comparison of navigation performance using INS aided with different techniques. Orders of this table are sorted from lowest 3D MAE to highest 3D MAE.

Config.	Algorithm	ZUPT-aided INS Enhancement Method				Mean Absolute Error (MAE)			Loop-Closure Error (LCE)		
		ALT	DALT	F2F	UWB-SLAM	3D [m] (%)	2D [m]	$\perp$ [m]	3D [m] (%)	2D [m]	$\perp$ [m]
H	UWB-Foot-SLAM2		✓	✓	✓	0.48 (0.013)	0.41	0.21	0.62 (0.017)	0.62	0.06
G	UWB-Foot-SLAM+		✓		✓	0.55 (0.015)	0.42	0.28	0.54 (0.015)	0.49	0.23
F	UWB-Foot-SLAM[89]				✓	1.96 (0.055)	0.66	1.78	1.05 (0.030)	0.84	0.64
E	Deterministic+		✓	✓		9.65 (0.275)	9.62	0.65	19.80 (0.565)	19.78	0.94
D	Deterministic[83]	✓		✓		9.94 (0.284)	9.86	1.05	20.59 (0.588)	20.34	3.20
C	ZUPT w/ALT+		✓			12.76 (0.364)	12.74	0.76	21.28 (0.608)	21.25	1.11
B	ZUPT w/ALT[91]	✓				12.93 (0.369)	12.85	1.18	21.81 (0.623)	21.54	3.41
A	ZUPT[216]					25.86 (0.738)	12.37	22.63	46.76 (1.335)	20.92	41.82

aided INS augmented with barometric altimeters and foot-to-foot ranging (Deterministic), ZUPT-aided INS augmented with differential barometric altimeters and foot-to-foot ranging (Deterministic+), the original UWB-Foot-SLAM, the original UWB-Foot-SLAM augmented with differential barometric altimeters (UWB-Foot-SLAM+), and the developed UWB-Foot-SLAM2. TABLE 9.4 summarizes the description of each navigation solution. Six different performance metrics are used to evaluate the navigation solutions, including 3D, 2D, and vertical Mean Absolute Errors (MAEs) and 3D, 2D, and vertical LCEs. The MAEs were calculated based on the four estimated destinations at the end of each round of the close-loop path.

Figure 9.18 and Figure 9.19 present the navigation solutions of the eight different approaches. It should be mentioned that beacons' locations in Figure 9.18 and Figure 9.19 were not pre-surveyed but estimated by the corresponding algorithms. TABLE 9.4 shows quantitative errors of the estimated navigation results. The LCEs of different approaches at the end of each round are shown in Figure 9.20. Figure 9.21 shows the position uncertainties of different states predicted by the EKF in the experiment. The horizontal position uncertainties in Figure 9.21 were computed by summing the 3σ along the x- and the y-axis. Comparing all the navigation approaches, the developed UWB-Foot-SLAM2 had the minimum 3D MAE of 0.48 [m] in this experiment, equivalent to 0.013% traveling distance based on a 3.5-[km]-

long trajectory. Using 3D MAE as a comparison metric, the UWB-Foot-SLAM2 improved navigation accuracy by $\times 3.08$, $\times 19.7$, and $\times 52.9$, as compared to the original UWB-Foot-SLAM, Deterministic navigation solution, and traditional ZUPT-aided INS. Discussions of the experimental results are presented in the next subsection.

Discussion The following lessons can be learned by inspecting the experimental results presented in Figure 9.19, Figure 9.20, Figure 9.21, and TABLE 9.4.

- All beacon's locations could be estimated when the original UWB-Foot-SLAM, UWB-Foot-SLAM+, and UWB-Foot-SLAM2 algorithms are used, and the estimated vertical locations matched the floor heights where the beacons were deployed. However, we could see in Figure 9.19 that the locations of the beacons estimated by the three different approaches were slightly distinct. The locations estimated by the developed UWB-Foot-SLAM2 had the smallest errors, as compared to the other two solutions.
- Following the previous bullet point, integrating barometers into the navigation solutions can enhance both localization and mapping accuracy along the vertical direction of the UWB-Foot-SLAM algorithm. The rationale behind the improvements is that when the UWB-Foot-SLAM operates in the mapping mode, the algorithm's performance highly depends on the ZUP-aided INS. The ZUPT-aided INS is identified to have position drifts faster along the vertical than the horizontal direction, and the vertical drifts were contributed largely from insufficient FSR and bandwidth of COTS IMU in the case of foot-mounted application [87]. Moreover, a standalone ZUPT-aided INS has been pointed out to have difficulties in tracking vertical displacement when operating in a moving elevator. The stance phase detection mechanism optimized for walking motion often identifies a moving elevator as a stationary event [92]. Augmenting a ZUPT-aided INS with a barometer could greatly mitigate this issue.
- Reference barometers integrated into beacons could improve vertical displacement es-

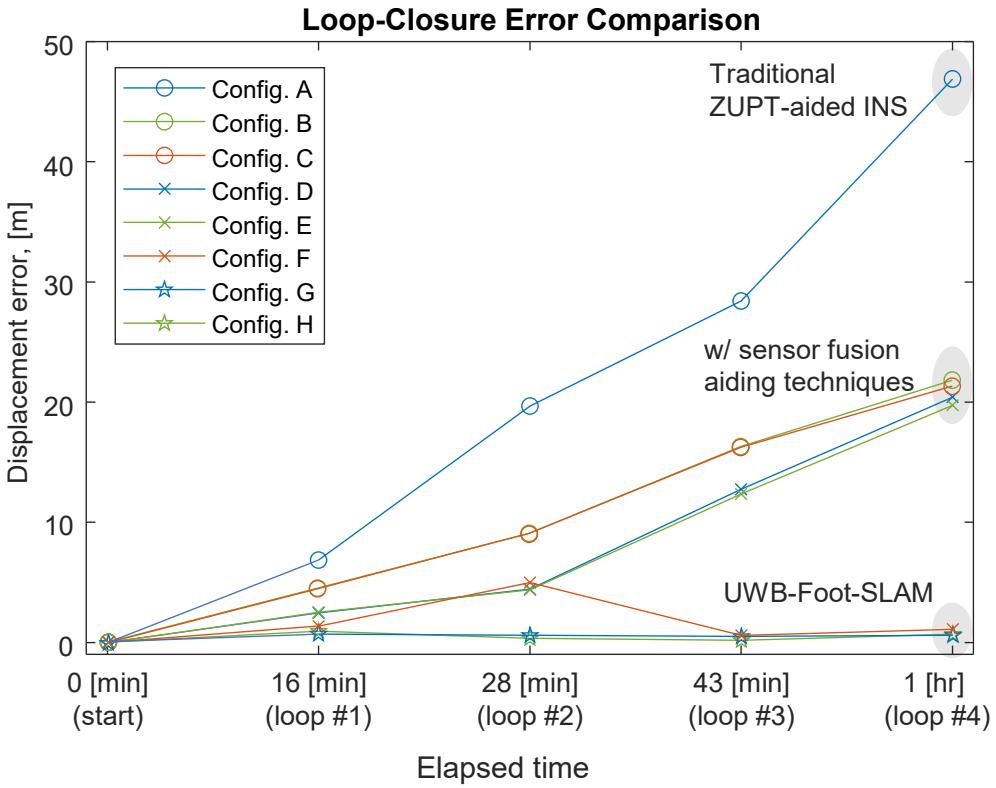


Figure 9.20: Propagation of LCEs of different navigation solutions. It could be observed that using self-contained aiding techniques for the ZUPT-aided INS could reduce the navigation errors increase rate. The UWB-Foot-SLAM framework could effectively bound error growth, allowing for high positioning accuracy in long-term navigation tasks.

timated by the foot-mounted barometer, and the improvements become significant in long-term tasks. In our experiment, we could observe that a standalone foot-mounted barometer had a height displacement error of around 1 [m] in the first 43 minutes of the experiment. In the last 17 minutes of the experiment, the height estimated by foot-mounted barometers had errors exceeding 3 [m]. These errors were reduced when the differential barometer mechanism was used in a navigation solution, as shown in TABLE 9.4. However, we would like to point out that using the reference barometer could not completely remove attitude estimation errors in this experiment. We believe the residual attitude errors were contributed by a combination of multiple factors, including sensor errors, such as scale factors and biases, and the fact that in some scenarios where foot-mounted UWB and seeded UWB had a LoS connection, the baseline

air pressure experienced by the reference barometer and the foot-mounted barometer may be different because the two barometers were physically far apart from each other. Future research is encouraged to investigate the error characteristics of the differential barometer mechanism.

- Inspecting the solutions without augmentation from beacons in Figure 9.19, which are labeled as configurations A-E, we could observe that the first 16 minutes of trajectories, colored in orange, have significantly smaller errors than the rest of the trajectories, colored in blue. This was due to the fact that position and yaw angle errors of configurations A-E are not globally bounded. The unbounded error growth could also be observed from another perspective in Figure 9.21, where the 3σ of the ZUPT-aided INS and the Deterministic solutions continue to grow until the end of the experiment.

- It can be seen in Figure 9.20 that after one hour of operation, the eight navigation solutions converge into three groups based on the levels of LCEs. The first group is the traditional ZUP-aided INS, which had the largest LCEs among all the solutions. The second group includes ZUPT-aided INS augmented with self-contained sensing modalities. Although altimeters provide vertical compensation, the unbounded position and yaw angle errors eventually lead to large navigation errors. The third group consists of different implementations of the UWB-Foot-SLAM, which had bounded position error growth, as compared other two groups.

- Observing the uncertainty propagation of beacons in Figure 9.21, we could see that beacons involved at a later stage of the experiment converge to a larger value, as compared to the ones that were involved at the early stage. This was because, in the early stage of navigation, foot-mounted IMU had smaller position uncertainties, allowing for updating positions of the beacon with a higher confidence level. At the later stage, the position uncertainty of the IMU increased, and the confidence level when estimating the beacon's position was also reduced.

- As compared to the original UWB-Foot-SLAM, the developed UWB-Foot-SLAM2 using augmentation with reference barometer and foot-to-foot ranging measurements achieves a smaller navigation error. This improvement was because, in the UWB-Foo-SLAM framework, the localization performance of the foot-mounted localization system directly affects the accuracy of both localization and mapping performance. In this section, the UWB-Foot-SLAM2 was demonstrated to improve the navigation accuracy of the UWB-Foot-SLAM. This improvement was expected because the Deterministic navigation approach was experimentally validated to have a better performance than a standalone ZUPT-aided INS [82]. Improving the ZUPT-aided INS can be done through other aspects, such as innovative design and fabrication of high-performance IMUs, machine-learning-enhanced stance phase detection, and additional sensor fusion techniques.

The experimental results presented in Section 9.3.3 and Section 9.3.3 demonstrate that the developed UWB-Foot-SLAM2 inherits the advantages of the original UWB-Foot-SLAM, which could simultaneously localize unknown beacons' positions with sufficiently high accuracy, and uses the augmentation with sensor fusion solutions to further improve both localization and mapping accuracy. The developed UWB-Foot-SLAM2 algorithm significantly outperformed the traditional ZUPT-aided INS in long-term pedestrian navigation accuracy by 52×, achieving loop-closure error of less than 1 [m] after navigating for 3.5 [km] in 1 hour. This section developed a UWB-Foot-SLAM2 algorithm that simultaneously localizes positions of a pedestrian and maps locations of unknown UWB beacons that are designed to be deployed during a navigation task. The developed UWB-Foot-SLAM2 algorithm preserved the characteristics of the original UWB-Foot-SLAM but is enhanced with multi-sensor fusion involving differential barometer and foot-to-foot ranging. We developed an experimental prototype, including the Lab-On-Shoe platform and the PEBBLE 2.0 system, and conducted two series of experiments to compare the performance of the developed UWB-Foot-SLAM2

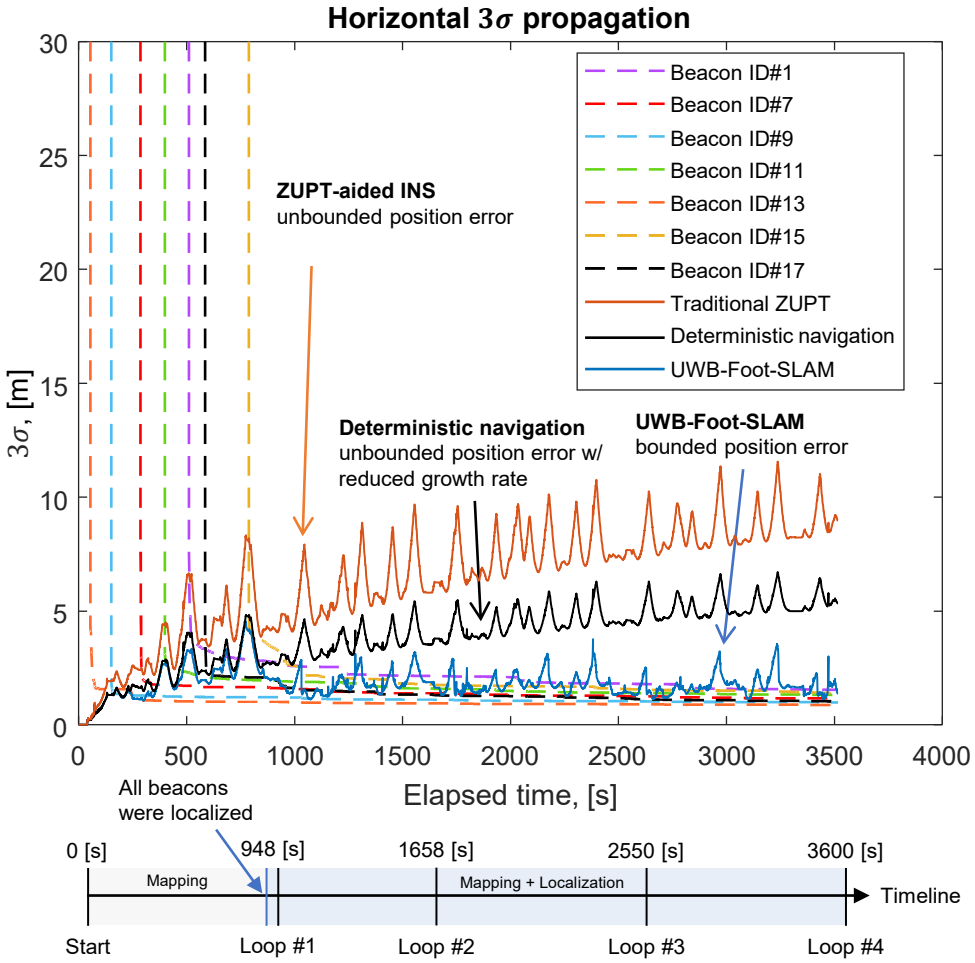


Figure 9.21: Propagation profile of the covariances associated with agent's and beacon's positions. It could be seen that the agent's position uncertainties were bounded in the case of the UWB-Foot-SLAM, while the uncertainties in the case of the ZUPT-aided INS followed an increasing trend.

algorithm with the original UWB-Foot-SLAM and different implementations of ZUPT-based INS. The first experiment involved evaluating the navigation solutions with a high-accuracy motion capture camera system and a subject walking along different patterns. In the worse case scenarios of this experiment, the ZUPT-aided INS had a position RMSE of 0.19 [m], an LCE of 0.3 [m], and a maximum error of 0.5 [m]. The original UWB-Foot-SLAM had an improved navigation performance with a position RMSE of 0.18 [m], an LCE of 0.27 [m], and a maximum error of 0.45 [m]. Positions of the two UWB beacons estimated by the original UWB-Foot-SLAM had displacement errors of 0.36 [m] and 0.25 [m], respectively. In the

second experiment, which involved a subject walking for approximately 3.5 [km] in one hour on three different floors with seven beacons, the ZUPT-aided INS, the Deterministic navigation, the original UWB-Foot-SLAM, and the developed UWB-Foot-SLAM2 had 3D MAE of 25.86 [m], 9.94 [m], 1.96 [m], and 0.48 [m]. It was demonstrated that the EKF covariances associated with pedestrian's positions in the case of the UWB-Foot-SLAM framework were bounded. The experimental results also show that the original UWB-Foot-SLAM algorithm could significantly improve the long-term positioning accuracy of a pedestrian inertial navigation system using foot-mounted IMUs, and the UWB-Foot-SLAM2 using sensor fusion enhancement further extends the high accuracy performance.

9.4 Conclusion

This chapter presents the theory, implementation, and experimental validation of the two UWB-Foot-SLAM frameworks, including the original UWB-Foot-SLAM algorithm and the UWB-Foot-SLAM2 algorithm. The developed UWB-Foot-SLAM frameworks use environment-deployed sensor-embedded beacons and foot-mounted inertial-based navigation systems. The beacons do not need to be pre-deployed or pre-surveyed. They are designed to be distributed during navigation tasks, and the foot-mounted navigation systems are used to determine the locations of the beacons. Once the beacons are localized, measurements of ranges between the foot and the beacons bound position error growth, enabling the foot-mounted navigation systems to operate in long-term navigation tasks. The developed original UWB-Foot-SLAM was experimentally demonstrated to map beacons' locations with displacement RMSEs of less than 0.3 [m] when evaluating with a motion capture camera system. In an experiment where a pedestrian navigated in a two-floor indoor environment for around 2.5 [km] in 25 minutes, the original UWB-Foot-SLAM achieved a loop-closure error of 1.49 [m] along the horizontal direction and 1.09 [m] along the vertical direction. In the developed UWB-

Foot-SLAM2, three additional enhancements were implemented to augment the original UWB-Foot-SLAM. The enhancements were a differential barometer mechanism, foot-to-foot ranging augmentation, and IMU-based beacon motion detection. Navigation results collected in an experiment involved a pedestrian walking for approximately 3.5 [km] in one hour on three different floors with seven beacons, UWB-Foot-SLAM2 had a 3D mean absolute error of 0.48 [m]. The research outlook along the direction of the developed UWB-Foot-SLAM framework is presented in Chapter 10.2.

Chapter 10

Conclusion

This Ph.D. dissertation focused on the development pedestrian navigation system implementing ZUPT-aided INS based on foot-mounted IMUs. New enhancement algorithmic and system-level techniques that address the ZUPT-aided INS on aspects of motion sensor, algorithm assumption, and estimation filter were developed, and a series of indoor pedestrian navigation experiment was conducted to investigate the validity of each developed enhancements. Specific contributions of the dissertation are summarized below.

10.1 Contribution of the Dissertation

Innovations on Motion Sensor

- Experimental evidence was provided in this thesis to demonstrate that COTS foot-mounted inertial sensors could have inaccurate accelerometer readings during the heel-strike phases due to insufficient FSR in daily pedestrian activities.
- This thesis developed a pedestrian inertial navigation simulation that combines a sim-

ulated IMU noise model and a relatively simple dynamic analytical model that mimics swing and contact during the foot motions based on an inverted pendulum dynamical system. The inverted pendulum model was developed by UCI Professor Alexander Voloshina. The author of this thesis, Chi-Shih Jao, developed a simulation approach generating foot-mounted IMU signals that considered deterministic noises, stochastic noises, and sensor limitation and combined the simulation approach with the pendulum mode. The developed pedestrian navigation simulation allowed for simulating the effects of the high-frequency sampling of sensor readings and large-magnitude signals during foot swing and contact on navigation performance. Navigation accuracy of the ZUPT-aided INS using simulated and experimental foot-mounted IMU measurements in walking scenarios had 6% different in position RMSEs. The developed model also accurately predicted the drift in the vertical direction, matching well the reported experiments. The simulation results show that insufficient FSR and bandwidth were factors causing the drifts along the vertical direction.

- An algorithmic reconstruction filter targeting to mitigate the issue of insufficient FSR and bandwidth was developed. This approach introduced an accelerometer measurement model that considers the effects of insufficient FSR, developed a Gaussian process regression to pre-process saturated accelerometer measurements for a ZUPT-aided INS, and verified the developed approach with real-world indoor pedestrian navigation experiments. The experimental results showed that when applying our developed reconstruction filter to the saturated accelerometer's measurements, the navigation accuracy of the ZUPT-aided INS along horizontal and vertical directions was increased by 5% and 50%, respectively, as compared to the case without using the developed approach.

- Prio-IMU was developed. The Prio-IMU was a systematic approach utilizing multiple IMUs to simultaneously increase sensor FSR and bandwidth while maintaining great noise performance. A Prio-IMU prototype was built, and it was integrated with

three different inertial sensors. The experimental results involving walking and running showed that both the horizontal and vertical RMSEs of the ZUPT-aided INS using the Prio-IMU prototype were improved by 79% and 82%, as compared to the case of using a single ICM−20948 IMU. The approach showed a method for resolving trade-offs in selection of inertial sensors for foot-mounted IMUs in pedestrian navigation scenarios.

- A BPNN-based temperature compensation method was developed for enhancing ZUPT-aided INS while operating in temperature-varying environments. Thermal hysteresis effects were experimentally demonstrated, and the temperature compensation method uses 12 separately trained feedforward NNs to predict thermal-induced errors, including bias drifts and noise variations of accelerometers and gyroscopes along the three reference axes of IMU. Pedestrian indoor walking experiments with a nominal length of 85.2 [m] were conducted, and the experimental results showed that when operating in environments where ambient temperature changed between 20°C and 50°C, a standalone ZUPT-aided INS had a position RMSE of 9.29 [m] while the developed temperature-compensated ZUPT-aided INS achieved a RMSE of 0.57 [m].

Innovations on Algorithm Assumption

- A zero-velocity detector using both an IMU and a DVS, referred to as the DVS-SHOE, was developed. We presented hardware design of a foot-mounted INS integrated with a DVS, and analyzed the properties of the DVS firing rate during indoor navigation experiments. The event-based camera-aided zero velocity detector was derived in a Generalized Likelihood Ratio Test (GLRT) framework, and this thesis evaluated the developed detector in terms of detection rate, false alarm rate, and navigation error. The experimental results showed that the CEP of the case using DVS-SHOE is reduced by around 25%, from 1.2 m to 0.9 m, as compared to the case of the traditional SHOE detector.

- This thesis developed a UA-SHOE detector. Measurements of a foot-mounted IMU and a downward-facing ultrasonic transducer were fused within a GLRT framework to enhance stance phase detection robustness. Indoor navigation experiments involving traveling at different speeds were conducted to evaluate the developed UA-SHOE detector. In the running experiments, the UA-SHOE detector outperformed the original SHOE detector by more than 50%.

- This dissertation introduced a log-likelihood ratio to quantify instability level of a pedestrian's foot, based on readings from inertial sensors. The foot instability metrics were used to adapt the covariance of the zero-velocity measurements in different scenarios for the ZUPT-aided INS, and the adaptive mechanism was referred to as the FIBA covariance. This thesis implemented the ZUPT-aided INS using the developed FIBA covariance in an EKF framework without a stance phase detector and investigated properties of EKF innovation sequences in the developed ZUPT-aided INS using the FIBA covariance. Experimental results showed that using the FIBA covariance showed a maximum improvement in navigation accuracy of 36% horizontally and 64% vertically, as compared to the conventional ZUPT-aided INS.

Innovations on Estimation Filter

- An closed-form analytical estimation of the displacement error standard deviation in the down direction was derived for ZUPT/altimeter-aided INS implemented in the EKF. The analytical expression was verified by numerical simulation and experimental, showing that the analytical estimation had an uncertainty of less than 20%.

- This thesis introduced for the first time a hybrid altimeter that uses both the ultrasonic altimeter and the barometer. This approach allowed a shoe-mounted ultrasonic sensor to be used as an altimeter in the case of walking on a flat plane or stairs, and an EKF framework was adopted to fuse the hybrid altimeter measurements with a

ZUPT-augmented Inertial Navigation System. The developed hybrid altimeter was experimentally verified to significantly enhanced the reliability of vertical position estimation in temperature and air pressure varying environments, as compared to a barometer.

- This dissertation presented a measurement model that uses self-contained vision measurements to enhance the yaw angle observability of the ZUPT-aided INS implemented in an EKF. A mechanism was provided to simulate the foot-to-foot relative position measurements obtained by shoe-mounted cameras. We verified the developed system with real-world experiments by comparing the results with a standalone ZUPT method and the ZUPT aided by foot-to-foot relative distance measurements. Experimental results showed a maximum improvement of 85% in accumulated errors, verifying the validity of the developed system in real-world environments.

- A PINDOC framework was developed in this thesis. The framework was a multi-agent indoor navigation solution that utilizes deterministic, opportunistic, and cooperative functionalities. A dedicated pedestrian navigation testbed was developed, integrating multiple sensors including IMUs, barometers, UWBs, and LTE receivers. This thesis compared the navigation accuracy of the developed systems implemented in different configurations using a real-world pedestrian indoor navigation experiment and analyzed the SWaP+C of the different configurations. The PINDOC was a collaborative contribution. Cooperative localization was developed by Ph.D. students Changwei Chen and Mingwon Soo under UCI Professor Solmaz Kia's supervision. The opportunistic approach were designed and independently tested by Ali Abdallah with Ohio State University Professor Zak Kassas's guidance. The author of this thesis, Chi-Shih Jao, supervised by UCI professor Andrei Shkel, developed the deterministic localization approach as well as integrated the deterministic, opportunistic, and cooperative approaches.

- Two SLAM frameworks, UWB-Foot-SLAM and UWB-Foot-SLAM2, utilizing foot-mounted IMUs and UWB were developed in this thesis. The developed UWB-Foot-SLAM simultaneously localized a pedestrian's positions and mapped positions of unknown beacons with measurements collected from all sensors integrated into a foot-mounted localization system and multiple beacons. This Thesis developed a hardware system that realizes the implementation of the developed UWB-Foot-SLAM framework and performed multiple real-world indoor pedestrian navigation experiments to validate the navigation performance of the developed algorithm. One experiment involved a subject walking for approximately 3.5 [km] in one hour on three different floors with seven beacons. The ZUPT-aided INS, the Deterministic navigation, the original UWB-Foot-SLAM, and the developed UWB-Foot-SLAM2 had 3D MAE of 25.86 [m], 9.94 [m], 1.96 [m], and 0.48 [m]. The experiments demonstrated that the UWB-Foot-SLAM solutions had significantly improved long-term accuracy and reliability, as compared to traditional pedestrian inertial navigation systems.

10.2 Future Research Directions

This part of the dissertation suggests future research directions corresponding to the developed approaches presented in the dissertation. The suggestions are discussed below.

10.2.1 Boosting FSR and Bandwidth of Inertial Sensors

Combination of Algorithmic and Syste-Level Approach

The developed approaches presented in Chapter 2 are suggested to be extended in the following direction. First, the chapter's reconstruction filter presented in Section 2.4 was designed

for only accelerometers. It would be beneficial to derive a counterpart of the reconstruction for gyroscopes. Second, the reconstruction filter and the prioritizable IMU array, presented in Section 2.5, can be combined to further increase the FSR and bandwidth of a foot-mounted IMU system. These two methods have complementary properties. On the one hand, the reconstruction filter could increase the FSR and bandwidth of a sensor to any value with an assumption on the signal structure. On the other hand, the prioritizable IMU array increases the FSR by using multiple sensors, but the increase is limited by the specification of the available COTS devices. The combined approach has the potential to capture violent motions experienced by the foot during navigation involving activities such as kicking doors, tripping, stamping, and jumping.

Component-Level Realization of Multi-IMU Array

The developed prioritizable IMU array presented in Section 2.5 of Chapter 2 had performance depend on the accuracy of the estimated relative positions between each on-board IMU, as the relative positions were needed to compensate for the Euler forces and the centrifugal forces experienced by IMUs mounted on different locations on a rigid body. In theory, the closer the placement of these IMUs, the less the differences in magnitudes of the forces. The developed prioritizable IMU array used inertial sensors with dimensions of a few millimeters. To reduce the relative positions with an order of magnitude, for example, to a few micrometers, integrating multiple sensors must be conducted on the component level before MEMS structures of inertial sensors are packaged. It is envisioned that the component-level realization of the multi-IMU array is less subjective to the accuracy of the estimated relative positions, which can improve the navigation performance of the ZUPT-aided INS using the prioritizable IMU array.

10.2.2 Enhancing Stance Phase Detection With Deep/Machine Learning

Moving Object Recognition for DVS/IMU Stance Phase Detection

During the experimental validation process reported in Section 4.3.4 of Chapter 4, the developed DVS-SHOE detector was found to have detection performance sensitive to moving objects presented within the FOV of the DVS and the complexity of visual patterns projected from navigating indoor environments. A potential technique to increase the reliability of the current DVS-SHOE detector is to use firing rates derived from a specific region of DVS's FOV. Furthermore, the DVS is envisioned to enhance a foot-mounted INS not only on the aspect of stance phase detection but also through other implementations, such as a SLAM framework that allows loop-closure detection based on visual features of DVS to be performed and bound position error growth.

Terrain Recognition for Downward-Ultrasonic/IMU Stance Phase Detection

The developed UA-SHOE detector, presented in Section 4.4 of Chapter 4, had a performance impacted by the terrains of operations in its current implementation. Ideally, the downward-facing ultrasonic range sensor is installed at a location that has contact with the ground. However, it was experimentally observed that the contacting location is different when performing different terrains. For example, when walking upstairs or climbing ladders, the toe side of a person's foot was usually the position in contact with the ground. In the case of going downstairs, this contacting position was often on the heel side of the foot. It is envisioned that this problem can be addressed through a system-level or machine-learning-aided approach. The system-level approach integrates multiple downward-facing ultrasonic sensors on different parts of a shoe. Likelihood statistics derived from measurements collected by the different ultrasonic sensors are compared, and stance phase detection could be achieved

by selecting the one with the highest likelihood. The machine-learning-aided approach can first classify pedestrian activities with machine learning or deep learning models, such as a support vector machine, random forest, or artificial neural network, and then adjust a hyper-parameter in the UA-SHOE the tune weighting ratios between measurements of the IMU and the downward-facing ultrasonic sensor.

Human-Activity-Recognition-Enhanced Stance Phase Detection

In Chapter 4, non-inertial sensing modalities were fused with measurements of foot-mounted IMU to modify the property of log-likelihood statistics in conventional IMU-based stance phase detector. The settings of hyper-parameters used in the developed stance phase detector may have different values when a person is performing different activities. It is envisioned that including a Human Activity Recognition (HAR) mechanism in the developed stance phase detectors could enhance detection performance. In such approaches, the HAR first classifies the type of activities a person is performing, and then the values of the hyper-parameters are adjusted to corresponding values to optimize detection performance.

10.2.3 Continuing FIBA Covariance

The following remarks are suggested for future research conducted along the direction of developed FIBA covariance presented in Chapter 5.

Hyper-Parameter Selection for Different Pedestrian Activities

The hyper-parameter selection for the developed FIBA covariance was conducted based on a series of walking-and-running experiments of 42.5 meters using a single subject. It is necessary for future research to investigate the validity of the hyper-parameter selection in

different cases, including performing other common pedestrian activities, such as sprinting, jumping, crawling, and side-stepping, on different terrains like stairs, sand, and grass. Moreover, in experiments discussed in Section 5.3, we could see that the innovation sequences along the horizontal and vertical directions had different behaviors. The different behaviors could be an indication that appropriate choices of the hyper-parameters, β mad γ, might not be the same for two different directions.

Combination of the Adaptive Covariance With Stance Phase Detection

The developed ZUPT-aided INS using the FIBA covariance feedbacked the zero-velocity measurements regardless of the stance phase or the swing phase. As discussed in Section 5.2.6, the zero-velocity measurements during the swing phases were highly correlated. The correlation violateed the fundamental assumption of EKF on uncorrelated measurements. In our developed FIBA covariance, the measurement covariance matrix was increased to a significantly large value to minimize the impact of the correlated measurements. A potential approach to further reduce the impact of violation of the EKF assumption on the correlation of measurements is to develop a hybrid approach combining the conventional ZUPT-aided INS with the developed FIBA covariance. The hybrid approach can work in a way that the zero-velocity measurements with the FIBA covariance are applied in the update step only when the likelihood of stance phase detection is below a high threshold and no measurement updates are performed when the foot is certainly moving.

FIBA Covariance for Multi-Agent Pedestrian Navigation

As discussed in Section 5.2, the developed FIBA covariance aimed to avoid applying the zero-velocity measurements with a high confidence when the foot was not completely stationary, and the covariance had low values when the foot was stable and significantly high values when

the foot was unstable. In some common pedestrian activities, for example, sprinting, the foot of a pedestrian is frequently unstable, even during the ground-contacting phases. In these situations, the FIBA covariance would give a low confidence to zero-velocity measurements and the accuracy of estimated velocities in the ZUPT-aided INS would start to decrease with time due to unavoidable sensor noise. This problem could be potentially mitigated through cooperative localization or sensor fusion solutions with other non-inertial sensing modalities.

Estimating Mean and Covariance of Stance-Phase Foot Residual Velocity

Chapter 5 presents the development of an adaptive mechanism to vary the covariance of the zero-velocity measurements used in the ZUPT-aided INS implemented in the EKF. The developed adaptive covariance mitigated the impact of unmodeled errors caused by residual velocities during the stance phases of a human gait cycle on navigation accuracy. It would be beneficial to extend this research direction to also estimate the statistical mean of the residual velocities. One potential approach to perform the estimation is to model the motion of a foot-mounted IMU during the stance phase as an inverted pendulum where the contact point of the foot on the ground is a non-slipping pivot point and the radius is the height of the IMU with respect to the ground.

10.2.4 Improving Hybrid Ultrasonic/Barometric Altimeters

It is envisioned that the approaches presented in Chapter 7 could be further extended in the following directions. First, factors of inertial sensor's FSR and bandwidth can be included in the derivation of the analytical expression predicting the covariance of the vertical displacement. Second, the developed hybrid altimeter, in its current implementation, had a larger vertical positioning error when operating on stairs, as compared to flat planes, ramps, and elevators. Additional development is needed to address this issue in order to utilize the

hybrid altimeter for long-term navigation tasks. Third, the size of the hardware developed to realize the real-time system architecture can be significantly reduced and ruggedized against heat and water. The miniaturized hardware can be embedded into the sole of a shoe. In such a configuration, investigations need to be conducted to 1) determine the optimal locations in a sole, 2) compensate for stress from human body weight applied on the hardware module, 3) develop system-level or component-level heat dissipation mechanisms, and 4) design water-resist protective cases that allow functional barometers.

10.2.5 Extending UWB-Foot-SLAM Framework

The developed UWB-Foot-SLAM framework, presented in Chapter 9, is envisioned to be further improved with the following suggested approaches.

Initialization Mechanisms

Initialization of UWB position states could affect the accuracy of the beacon's estimated position. In the developed approach, a beacon's position was initialized with an agent's current position. During the experiments presented in Section 9.3.3 and Section 9.3.3, the agent deployed a beacon within reach, matching the design of our approach. However, the deployment could be done in a more flexible manner, such as by throwing beacons to distant locations. In such cases, the initialization mechanism discussed in this section could lead to the estimated beacons' location being stuck in a statistical local minimum, degrading the navigation performance. One potential approach to address this issue is to use multiple initial guesses of a beacon's location, compute the likelihood of each guess, and select the one with the highest likelihood.

Multi-Model Kalman Filter Implementation

It was demonstrated in Section 9.3.3 that the pattern of a pedestrian's navigation trajectory could affect the estimation accuracy of beacons' locations, and we observed that, when a pedestrian traveled only horizontally along a straight line, the estimated beacons' positions had significantly larger errors along the axes perpendicular to the direction of travel than those in parallel to the direction. The large errors could exceed the associated covariances, indicating existence of unmodeled errors in the estimation filter. It would be beneficial for future research to develop a multi-model approach to mitigate this issue.

UWB Bias Compensation

The ability to identify and compensate for NLoS UWB range measurements directly affected both the mapping and localization performance of our developed UWB-Foot-SLAM. The experimental prototype discussed in Section 9.3.2 included foot-mounted UWB modules. This configuration was designed to avoid the need to estimate relative positions between a UWB and an IMU attached to a pedestrian. However, as compared to other mounting positions, such as head or shoulder, foot-mounted UWBs had more difficulties in receiving LoS measurements, as the modules were close to the ground and the direct signal path could be blocked by a pedestrian body part [32]. To improve the UWB range measurement accuracy, advanced LoS/NLoS detection and bias compensation approaches could be advantageous [262].

Analytical Prediction of Position Uncertainties

It was demonstrated in Section 9.3.3 that position uncertainties of a beacon estimated by the UWB-Foot-SLAM framework were not fixed values but varying values due to the stochastic

nature of the EKF used to realize the UWB-Foot-SLAM. The position uncertainties of beacons were affected by multiple factors, including settings of the process and measurement noise covariance matrix, timestamps of the first LoS foot-to-beacon UWB measurements, patterns of a pedestrian's trajectory, a pedestrian's activities, stance phase detection performance, and noise performance of inertial sensors and UWB modules. Future research is needed to determine an analytical relationship between the uncertainties and the contributing factors. A numerical simulation for the UWB-Foot-SLAM framework could enable isolating and characterizing error sources in the algorithm, facilitating designing corresponding error mitigation approaches.

De-Centralized Implementation

The UWB-Foot-SLAM2 algorithm developed in Section 9.3 and the original UWB-Foot-SLAM discussed in Section 9.2 were realized in a centralized framework, where all the states were updated in every iteration of the EKF, even if some of the beacons were not connected. The developed UWB-Foot-SLAM could be extended to a de-centralized framework [264], which is computationally less expensive and would be more friendly to be implemented in real-time on a microcontroller. Investigating the trade-offs between navigation performance and computational complexity of centralized and de-centralized realizations would be beneficial.

Hardware Upgrades for Real-Time Implementation

The developed UWB-Foot-SLAM2 algorithm can be implemented in real-time, although the experimental results presented in Section 9.3.3 were obtained in a post-processing fashion. The Lab-On-Shoe platform and PEBBLE 2.0 system presented in Section 9.3.2 were developed with a priority on flexibility, performance, and fast proof-of-concept prototyping. To

have a real-time implementation of the developed algorithm, modifications to the current testbed are needed, as our current approach needs sensor data that are stored locally on an SD card integrated into a beacon. One potential approach is to leverage the UWB modules such that messages are exchanged wireless between two UWB modules when range measurements between them are being collected. In such a case, each message packet should include a beacon's current estimated positions and corresponding uncertainties, onboard reference barometer readings, and motion detection results.

Minimizing Power Consumption for Idle Beacons

The power consumption performance of the PEBBLE 2.0 system can be further improved. In our current settings, a beacon after being deployed in a navigation environment only provided range measurements to the foot-mounted system briefly and remained idle for the majority of the time. During the idleness, the beacon can be designed to go into "sleep" mode, where much less power is consumed, and "wake-up" by signals with a frequency range lower than a UWB.

Enhancing UWB-Foot-SLAM With Augmentations for ZUPT-aided INS

The performance of localization of pedestrians and mapping unknown beacons in the UWB-Foot-SLAM framework depended on the performance of the built-in ZUPT-aided INS. We demonstrated in Section 9.3 that using self-contained enhancement for the ZUPT-aided INS could altogether augment the performance of the original UWB-Foot-SLAM. In the literature related to foot-mounted IMUs, many enhancement sources from multiple different aspects of the system have been investigated, including robust stance phase detection [178, 93, 223], additional sensor fusion solutions [1, 239], and IMU compensation [90, 87]. We believe that combining some of the techniques with our UWB-Foot-SLAM framework could lead to

improvement in navigation performance and reliability, but further research is needed to have experimental validation and quantitatively evaluate improvements brought by the additional sensing modalities.

10.2.6 Foot-mounted-INS-Enabled Mapping and Path Planning

The thesis focused on the localization aspect of a pedestrian navigation system. The developed foot-mounted localization system can be leveraged to combine with additional wearable devices performing mapping and guidance, such as handheld scanning platforms or head-mounted displays integrating Augmented/Virtual Reality technologies. As compared to LiDAR- or camera-based mapping systems, foot-mounted localization systems based on ZUPT-aided INS can operate through scenarios where lumination is low, environments are filled with smoke or airborne particles, spatial geometry is symmetric, or background contains insufficient visual feature landmarks. A combination of foot-mounted localization devices, handheld scanning platforms, and head-mounted guidance systems can enable navigate a person from point A to point B in a completely unknown and dynamic environment.

10.3 Commcercializable Solution: emergency Firefighter Indoor Navigation Systems (eFINS)

The approaches developed in this dissertation can be all integrated together to achieve an accurate and reliable pedestrian navigation system. Such a system can be utilized for the purposes of firefighter localization or other location-based services, including contact tracing, gaming, shopping, rehabilitation, personal health monitoring, etc. This part of the dissertation presents a system architecture that could potentially be commercialized.

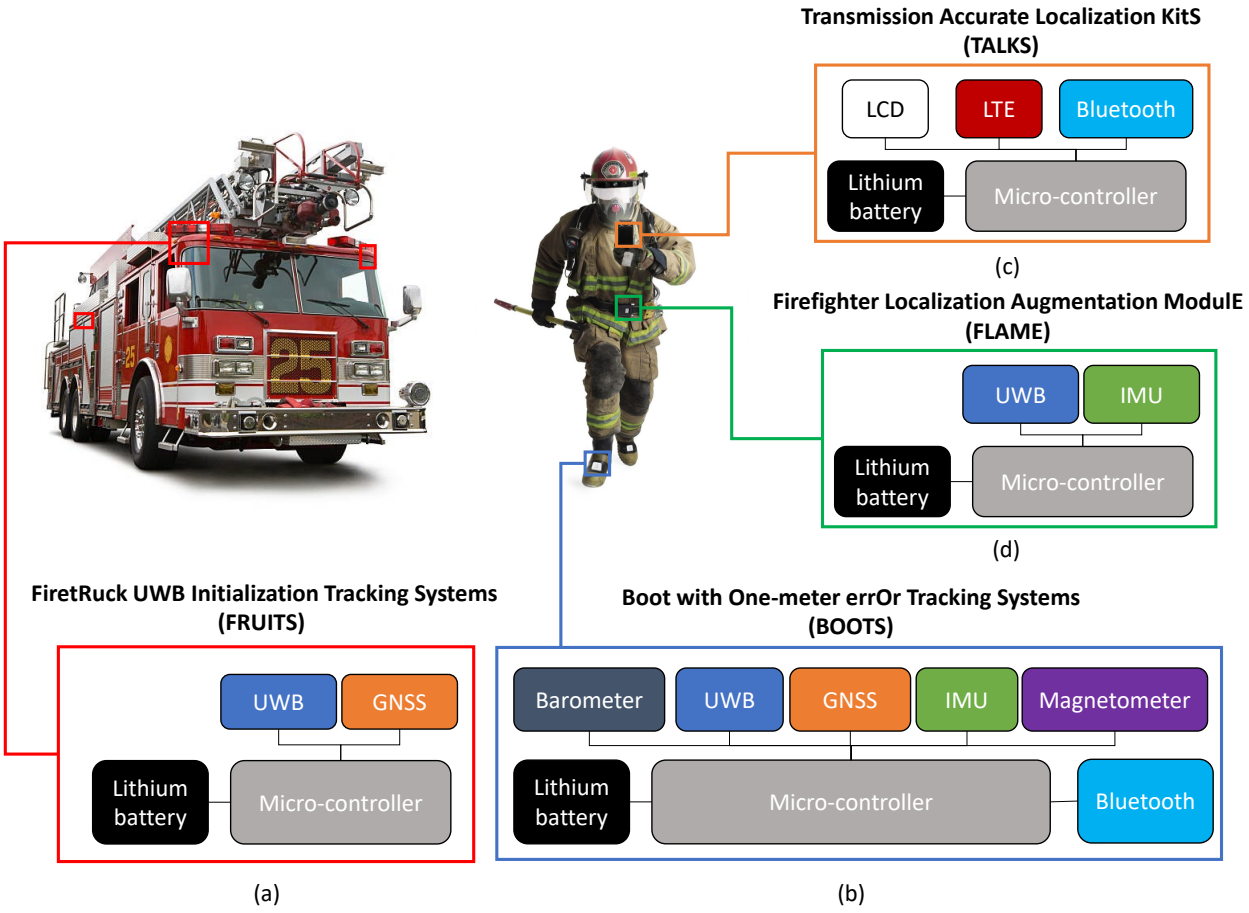


Figure 10.1: The concept of the proposed prototype eFINS. (a) FRUITS systems to be installed on a firetruck. (b) BOOTS to be embedded inside the sole of a firefighter boot. (c) TALKS for transmission and visualization of firefighter's current locations. (d) FLAME to be distributed in operating environments to further enhance navigation accuracy of the BOOTS.

a technology prototype, emergency Firefighter Indoor Navigation Systems (eFINS), aiming to address the technology gap in firefighter location tracking. The eFINS are designed to leverage the augmented ZUPT-aided INS using foot-mounted IMUs with additional approaches to address the six issues discussed in the previous paragraph. The eFINS is a wearable system that provides real-time position data of firefighters, allowing outside incident commanders or firefighters themselves to monitor their locations continuously.

The proposed eFINS provides real-time navigation solutions for firefighters and first responders. As compared to other indoor navigation technologies, our solutions have multiple advantages, including 1) operating in an infrastructure-free manner, 2) requiring very short to none installation time, 3) providing consistently available position data independent of operating building types and environments, 4) allowing augmentation from some navigational infrastructure when feasible, 5) having small form factors that do not affect firefighter's motion, 6) having low power consumptions, and 7) integrating approaches addressing issues in state-of-the-art foot-mounted INS technology to achieve navigation error of less than 1 meter for an extended period.

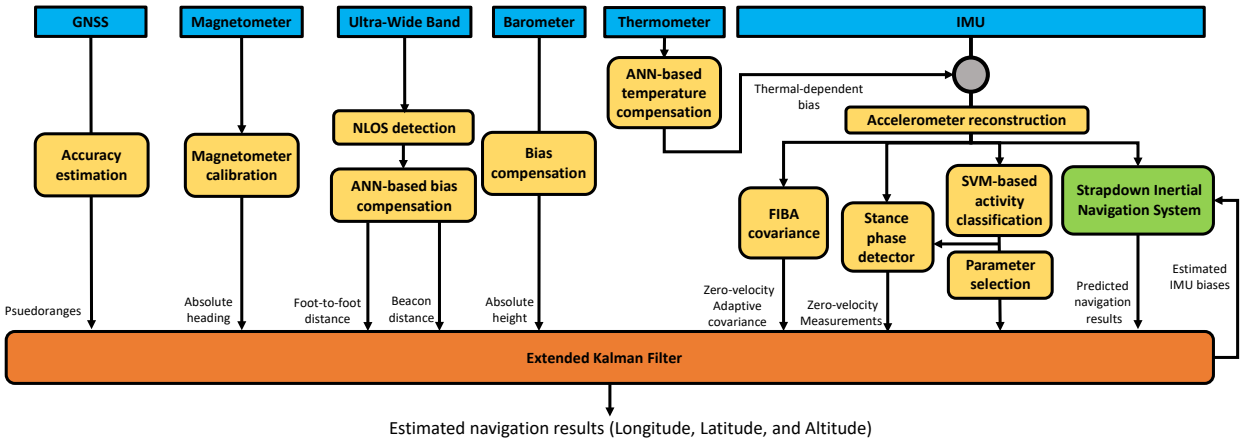


Figure 10.2: Navigation solutions implemented on the micro-controller in the BOOTS module.

System Design

The proposed system, eFINS, includes four different components: Boot with One-meter errOr Tracking System (BOOTS), FiretRuck UWB Initialization Tracking Systems (FRUITS), Firefighter Localization Augmentation ModulE (FLAME), and Transmission Accurate Localization Kits (TALKS). The concept of the proposed eFINS is illustrated in Figure 10.1. It is worth mentioning that, in the eFINS, the BOOTS module produces real-time position information of a person using self-contained foot-mounted INS technology, which operates without any external signals from pre-deployed infrastructures. Previous research has shown the technology to localize a person in a firefighter training center, which included different buildings types and various environmental conditions [3].

Boot with One-meter errOr Tracking System (BOOTS)

Hardware The hardware of BOOTS is shown in Figure 10.1(b). The module integrates a micro-controller, multiple COTS sensors, a Bluetooth transmitter on a PCB. The sensors include an industrial-grade IMU, a barometer, a UWB, a GNSS receiver, and a magnetometer. Industrial-grade IMU is selected to achieve great sensor noise performance while maintaining the cost of the system at a reasonable level. The module will be powered up with a lithium-ion battery and protected with an aluminum case. The size of the entire unit is expected to be on the order of 4 cm×4 cm×3 cm. The final form of this module will be embedded in the sole of the firefighter's boot. Each boot will have one such module.

Navigation algorithm The real-time navigation algorithm presented in Figure 10.2 is realized in an EKF framework. The prediction step of the EKF implements the strapdown INS algorithm. The update step of the EKF uses multiple aiding techniques. The ZUPT algorithm is used to bound velocity errors. Altimeter measurements are used to bound

vertical errors. When operating in outdoor environments, where GNSS signals are available, and magnetometer interferences are minimal, GNSS pseudoranges and magnetometer measurements are used to provide compensation of absolute positions and heading angles, respectively. GNSS and INS will be integrated in a tightly coupled manner. The UWB devices on two different boots of the same person will be paired to obtain foot-to-foot range measurements to improve heading angle and position uncertainty. Furthermore, the UWB devices on different firefighters will communicate to collect inter-agent ranging measurements.

There are issues in the ZUPT-aided INS that need to be addressed. We present solutions to each of the problems in the following paragraphs. The ZUPT-aided INS has a filter inconsistency issue when the EKF is used. This issue arises because, in reality, the velocity of a person's foot during the stance phase is not exactly zero, which violates the assumptions in the EKF. Previous research has demonstrated that using a Foot-Instability-Based Adaptive covariance with the ZUPT algorithm can reduce the error resulting from the filter inconsistency issue. In the proposed BOOTS, the FIBA covariance is implemented to enhance the performance of the ZUPT-aided INS [4].

The detection performance of the stance phase detector used in the ZUPT algorithm significantly affects the navigation accuracy. A detector optimized for the case of walking might not be optimal for other cases, such as running. To address this issue, a SVM-based activities classification algorithm can be included to identify what activity is being performed using IMU measurements. We will consider 14 common firefighter activities, including walking slowly, normally, fast, jogging, running, sprinting, jumping, side-stepping, walking backward, crawling, going up and downstairs. Based on the identified activities, parameters of the EKF and thresholds for the stance phase detector are adapted to values that minimize navigation errors. When the ZUPT-aided INS operates in activities involving foot striking the ground intensely, such as sprinting, forces generated by the strikes can be larger than 20 g, which exceeds many the FSR of most high-performance COTS IMUs. To minimize

the impact of this issue, we will include a GPR-based accelerometer readings reconstruction module. This technique has been experimentally proven to reduce the navigation error generated by the insufficient accelerometer FSR in a post-processing manner. We plan to implement this technique in the real-time implementation of the BOOTS.

In a rescuing mission, firefighters often navigate in environments with significant temperature variations. The temperature variations can degrade the accuracy of a traditional ZUPT-aided INS because the variation introduces additional thermal-dependent biases in IMU measurements. To solve this problem, we include an Artificial Neural Network (ANN) based temperature compensation approach to minimize the thermal-dependent errors in the ZUPT-aided INS. In this approach, the ANN will be trained in an offline manner. In the real-time implementation, the ANN will be performing only the feed-forward step to predict thermal-induced biases.

The proposed BOOTS are designed with the capability to operate without any pre-deployed navigational infrastructure or the other three modules in the eFINS, including FRUITS, FLAME, and TALKS. The final form of the BOOTS aims to provide navigation solutions with 1-meter accuracy when performing different activities in extreme weather environments for 15 minutes.

Firefighter Localization Augmentation ModulE (FLAME)

The FLAME is designed as add-on hardware to extend the 1-m navigation accuracy of the BOOTS from around 15 minutes to an hour. The FLAME is a ruggedized unit that can be distributed in environments during a navigation mission. The hardware of the module, illustrated with a block diagram in Figure 10.1 (d), integrates a micro-controller, a low-cost IMU, a UWB device, and a lithium-ion battery. The micron-controller handles sensor data acquisition, stationary status detection, and UWB communications. The UWB device

is paired with the UWB modules on the BOOTS. The IMU on the FLAME is used to determine if the module is stationary. The size of this module is estimated to be around 4 cm×4 cm×3 cm, and we will package the device with an aluminum shell.

After being distributed in an environment, our navigation algorithm initializes the location of the FLAME with very large uncertainty. When the module becomes stationary, location data and UWB ranging measurements will be transmitted to the BOOTS to perform a SLAM algorithm. The estimated location data will be transmitted back to the FLAME and stored in the micro-controller. When the IMU on the FLAME detects a motion, the micro-controller will reset the position uncertainty to a very high value, and the location of the FLAME will have to be re-determined by the BOOTS.

In a navigation task, the FLAME does not need to be pre-deployed, and its location does not require pre-surveying. Multiple FLAMEs can be distributed in the environment on the ground, wall, or civilian to the rescued. Existing COTS UWB devices, such as DWM1000, can provide a distance accuracy on the level of a couple of centimeters. The BOOTS enhanced with FLAME is expected to provide a position accuracy of less than 1 meter after navigating for more than 30 minutes to an hour.

FiretRuck UWB Initialization Tracking Systems (FRUITS)

When accurate longitude, latitude, and altitude information is needed, initial localization accuracy of a dead-reckoning navigation system becomes critical. FRUITS is a system designed to provide accurate initial location information in terms of longitude, latitude, and altitude for the BOOTS. Each module The FRUITS includes a micro-controller, a GNSS receiver, a UWB device, and a battery. A block diagram of the hardware is presented in Figure 10.1(a). The size of this module is estimated to be 5 cm×5 cm×3 cm.

To use the FRUITS, four or more modules will be placed on the top of a firetruck. In the

micro-controller of each module, we will obtain the location data based on the differential GPS mechanism. This location data will be transmitted to the BOOTS via the UWB module. In the initialization process of the BOOTS, the system performs trilateration based on the received location data and UWB range measurements to determine the initial location.

Transmission Accurate Localization Kits (TALKS)

Figure 10.1(c) shows a block diagram of the TALKS hardware. The TALKS is designed to enable extensibility and interoperability for the proposed eFINS, and this module allows firefighters to visualize and transmit location information estimated by the BOOTS. The TALKS module includes a micro-controller, a Liquid-Crystal Display (LCD), a LTE receiver, Bluetooth, and a lithium-ion battery. The TALKS will be attached to the firefighter's forearm and display current firefighter's 3D locations transmitted from the BOOTS via Bluetooth. When LTE signals become available, the location information can be transmitted to an outside incident commander to monitor all firefighters' positions in the line of duty. The size of this module is estimated to be around 15 cm×10 cm×1 cm.

Potential Technology Capacity

This part of the section presents the criteria and general metrics listed in Table 1 of the FRST Challenge Rules document and the technical capabilities discussed in the document.

Location Tracking Accuracy

The proposed eFINS aims to provide real-time navigation solutions for each firefighter with a position error of less than 1 meter along the x-, y-, and z-axes. In the eFINS, the BOOTS implemented a self-contained approach based on the ZUPT-aided INS augmented with sev-

eral enhancement techniques, including an altimeter, inter-agent ranging measurements, and pedestrian activity classification. In previous research, the standalone ZUPT-aided INS with a consumer-grade IMU has been analytically predicted and experimentally demonstrated with a navigation error of around 1% of total traveling distances in the case of walking. This result indicates that the system is more accurate when initially deployed, but accumulated errors decrease the system's accuracy over time. Moreover, it has been pointed out that the standalone ZUPT-aided INS has higher accuracy when standing still or walking than other activities.

Our team will address these problems by enhancing the foot-mounted INS technology with 1) a pedestrian activity classification algorithm and 2) a UWB-SLAM algorithm. The classification will be realized with machine learning approaches, such as SVM, Random Forest (RF), and ANN. We will consider activities of walking slowly, walking fast, jogging, running, sprinting, walking backward, side-stepping, climbing, jumping, and going up and downstairs. Parameters of the EKF and the stance phase detector used in the ZUPT-aided INS will be optimized to minimize the position errors in each activity. With the augmentation of the pedestrian activities classification, the accuracy of the ZUPT-aided INS in non-walking operations is expected to be significantly increased.

To bound the position error growth within the 1-m standard for an extended period of time, we use a UWB-SLAM algorithm based on the BOOTS and the FLAME in the eFINS. Since the BOOTS at initial deployment and the LOS UWB range measurements both have centimeter-level accuracy, the accuracy of the navigation solutions will be on the level of 50 cm to 1 meter when navigating for 30 minutes to an hour.

Ease of Deployment

The proposed eFINS is based on a self-contained and infrastructure-free solution. In the eFINS, the BOOTS system can provide position data of a person with respect to the initial position without any external devices or infrastructures. The final form of the BOOTS device will be embedded in the sole of firefighter boots, and therefore, the position data can be immediately collected in emergency situations. The FRUITS systems, designed to provide initial longitude, latitude, and altitude information for the BOOTS, will be installed on a firetruck. This module uses the GNSS as a dedicated navigational infrastructure. The FLAME device is designed to be distributed in the environments during a firefighting mission. We understand that a typical fire crew is comprised of three to five firefighters, including a fire captain, a fire equipment operator, and firefighters. While the BOOTS will be equipped on each crew member, the FLAME modules will be distributed by throwing or rolling in the environments by the fire captain or the fire equipment operator during a mission. The TALKS is intended to report current position data generated by the BOOTS in real-time to firefighters or incident commanders. It is designed in the form of a forearm-attached device. However, the design may be modified in other forms, such as integration with a hand-held thermal camera or a small display inside firefighter's gas masks.

Robustness/Ruggedization

The navigation algorithm of the eFINS system may be unreliable in two scenarios. First, in activities where the foot strikes the ground violently, the magnitude of the forces can exceed the full-scale range of most COTS IMUs. We address this problem by including a signal reconstruction mechanism to preprocess IMU readings in the algorithm. Second, in temperature-varying situations, IMU readings are contaminated by thermal-induced error. We minimize the impact of the error by including a temperature compensation algorithm.

The communication involved in the proposed eFINS include the I2C, SPI, UART, and LTE. The I2C and SPI protocol will be during sensor measurement acquisition processes with a micro-controller. The UART protocol will be used to transmit position data via Bluetooth from the BOOTS to the TALKS. With an assumption that the two devices will have a maximum separation of 2 m during operation, this Bluetooth communication will be reliable. The LTE protocol will be used to transmit the position data from the TALKS to an outside command-and-control center when the signals are available.

The BOOTS module will be shelled with an aluminum package and embedded inside the sole of a firefighter boot with one small opening for charging the battery. The other three modules, FRUITS, FLAME, and TALKS, will all be packaged with cases made of PPE material. Each of the modules includes a lithium-ion battery that can power up the device for more than 1 hour.

User Experience

When off-duty, the FRUITS module will be installed on top of the firetruck. A bag of FLAME devices will be attached to firefighting backpacks. At the beginning of a mission, each member in a fire crew turns on the BOOTS and the TALKS modules next to a firetruck and stands still for around 5 seconds. After this period, the system starts tracking firefighters' positions. During the mission, the person who has the FLAME modules will turn on the module and distribute, one at a time, the device in the operating environment. In the cases of the proposed eFINS system used by other professions or roles, such as police SWAT team or military personnel, the BOOTS, the FLAME, and the TALKS will be used in the same way as firefighters. The FRUITS module has to be mounted on associated vehicles, for example, police cars.

Extensibility & Interoperability

The proposed eFINS system can be used as a standalone and self-contained localization system. In the current design of the system, we plan to transmit estimated position data through Bluetooth or cellular LTE signals to an external device for extensibility and interoperability. The proposed BOOTS module also has a GNSS receive to improve navigation accuracy when operating in outdoor environments. If integrating additional functionalities is requireed, such as Wi-Fi, we will add the additional requested sensor module to the BOOTS system.

Bibliography

[1] A. Abdallah, C.-S. Jao, Z. Kassas, and A. M. Shkel. A pedestrian indoor navigation system using deep-learning-aided cellular signals and ZUPT-aided foot-mounted IMUs. *IEEE Sensors Journal*, 22(6):5188–5198, 2022.

[2] A. Abdallah and Z. Kassas. Deep Learning-Aided Spatial Discrimination for Multipath Mitigation. In *IEEE/ION Position, Location and Navigation Symposium (PLANS)*, pages 1324–1335, April, 2020.

[3] A. Abdallah and Z. Kassas. Multipath mitigation via synthetic aperture beamforming for indoor and deep urban navigation. *IEEE Transactions on Vehicular Technology*, 70(9):8838–8853, September, 2021.

[4] A. Abdallah, J. Khalife, and Z. Kassas. Experimental characterization of received 5G signals carrier-to-noise ratio in indoor and urban environments. In *93rd IEEE Vehicular Technology Conference*, Virtual conference, Apr. 25−19, 2021.

[5] A. Abdallah, K. Shamaei, and Z. Kassas. Indoor localization with LTE carrier phase measurements and synthetic aperture antenna array. In *32nd International Technical Meeting of the Satellite Division of The Institute of Navigation (ION GNSS+)*, St. Louis, MO, USA, Sep. 21−25, 2019.

[6] A. A. Abdallah, K. Shamaei, and Z. M. Kassas. Assessing real 5G signals for opportunistic navigation. In *33rd ION GNSS+*, Portland, OR, USA, Sep. 25−29, 2020.

[7] M. H. Afzal, V. Renaudin, and G. Lachapelle. Assessment of indoor magnetic field anomalies using multiple magnetometers. In *23rd ION GNSS+*, Portland, OR, USA, Sep. 21−24, 2010.

[8] D. B. Ahmed, E. M. Diaz, and S. Kaiser. Performance comparison of foot-and pocket-mounted inertial navigation systems. In *7th IEEE International Conference on Indoor Positioning and Indoor Navigation (IPIN)*, Alcalá de Henares, Madrid, Spain, Oct. 4−7, 2016.

[9] D. B. Ahmed, L. E. Díez, E. M. Diaz, and J. J. G. Domínguez. A survey on test and evaluation methodologies of pedestrian localization systems. *IEEE Sensors Journal*, 20(1):479–491, 2019.

[10] R. M. Alexander. Simple Models of Human Movement. *Applied Mechanics Reviews*, 48(8):461–470, 08 1995.

[11] R. Ali, R. Liu, A. Nayyar, B. Qureshi, and Z. Cao. Tightly Coupling Fusion of UWB Ranging and IMU Pedestrian Dead Reckoning for Indoor Localization. *IEEE Access*, 9:164206–164222, 2021.

[12] R. Alonso and M. D. Shuster. Complete linear attitude-independent magnetometer calibration. *The Journal of the Astronautical Sciences*, 50:477–490, 2002.

[13] J. L.-F. Ang, W.-K. Lee, B.-Y. Ooi, T. W.-M. Ooi, and S. O. Hwang. Pedestrian Dead Reckoning with correction points for indoor positioning and Wi-Fi fingerprint mapping. *Journal of Intelligent & Fuzzy Systems*, 35(6):5881–5888, 2018.

[14] M. Angermann and P. Robertson. FootSLAM: Pedestrian simultaneous localization and mapping without exteroceptive sensors—Hitchhiking on human perception and cognition. *Proceedings of the IEEE*, 100(Special Centennial Issue):1840–1848, 2012.

[15] M. Angermann, P. Robertson, T. Kemptner, and M. Khider. A high precision reference data set for pedestrian navigation using foot-mounted inertial sensors. In *1st IEEE IPIN*, Zurich, Switzerland, Sep. 15—17, 2010.

[16] G. Araghi et al. Temperature compensation model of MEMS inertial sensors based on neural network. In *IEEE/ION PLANS*, Monterey, CA, USA, Apr. 23—26, 2018.

[17] C. T. Ardito, J. J. Morales, J. Khalife, A. Abdallah, Z. M. Kassas, et al. Performance evaluation of navigation using LEO satellite signals with periodically transmitted satellite positions. In *International Technical Meeting of The Institute of Navigation (ION ITM)*, Reston, VA, USA, Jan. 28—31, 2019.

[18] S. Askari, C.-S. Jao, Y. Wang, and A. M. Shkel. A Laboratory Testbed for Self-Contained Navigation. In *IEEE International Symposium on Inertial Sensors and Systems (INERTIAL)*, Naples, FL, USA, Apr. 1—5, 2019.

[19] A. Bahillo, I. Angulo, E. Onieva, A. Perallos, and P. Fernández. Low-cost Bluetooth foot-mounted IMU for pedestrian tracking in industrial environments. In *IEEE International Conference on Industrial Technology (ICIT)*, Seville, Spain, Mar. 17—19, 2015.

[20] A. Bahillo, A. Arambarri, I. Angulo, E. Onieva, P. Elejoste, and A. Perallos. Implementing a Pedestrian Tracker Using Low-Cost Bluetooth Inertial Sensors. In *International Conference on Ubiquitous Computing and Ambient Intelligence*, Belfast, Northern Ireland, Dec. 2—5, 2014. Springer.

[21] J. B. Bancroft, G. Lachapelle, M. E. Cannon, and M. G. Petovello. Twin IMU-HSGPS integration for pedestrian navigation. In *21st ION GNSS+*, Savannah, GA, USA, Sep. 16—19, 2008.

[22] S. Beauregard. A helmet-mounted pedestrian dead reckoning system. In *3rd International Forum on Applied Wearable Computing*, Bremen, Germany, Mar. 15—16, 2006. VDE.

[23] J. Bird and D. Arden. Indoor navigation with foot-mounted strapdown inertial navigation and magnetic sensors [emerging opportunities for localization and tracking]. *IEEE Wireless Communications*, 18(2):28–35, 2011.

[24] J. Borenstein, L. Ojeda, and S. Kwanmuang. Heuristic reduction of gyro drift in imu-based personnel tracking systems. In *Optics and Photonics in Global Homeland Security V and Biometric Technology for Human Identification VI*, volume 7306, pages 244–254. SPIE, 2009.

[25] G. Bradski and A. Kaehler. *Learning OpenCV: Computer vision with the OpenCV library.* " O'Reilly Media, Inc.", 2008.

[26] T. J. Brand and R. E. Phillips. Foot-to-foot range measurement as an aid to personal navigation. In *59th Annual Meeting of The Institute of Navigation and CIGTF 22nd Guidance Test Symposium*, Albuquerque, NM, USA, Jun. 23—25, 2003.

[27] L. Bruno and P. Robertson. WiSLAM: Improving FootSLAM with WiFi. In *2nd IEEE IPIN*, Guimarães, Portugal, Sep. 21—23, 2011.

[28] D. Buhaiov, V. Shelever, and V. Avrutov. Artificial Neural Networks application to MMG temperature calibration. In *5th IEEE International Conference Actual Problems of Unmanned Aerial Vehicles Developments (APUAVD)*, Kyiv, Ukraine, Oct. 22—24, 2019.

[29] H. Carlsson, I. Skog, and J. Jaldén. Self-calibration of inertial sensor arrays. *IEEE Sensors Journal*, 21(6):8451–8463, 2021.

[30] W. Chai, C. Chen, E. Edwan, J. Zhang, and O. Loffeld. 2D/3D indoor navigation based on multi-sensor assisted pedestrian navigation in Wi-Fi environments. In *IEEE Ubiquitous Positioning, Indoor Navigation, and Location Based Service (UPINLBS)*, Helsinki, Finland, Oct. 3—4, 2012.

[31] C. Chen, C.-S. Jao, A. M. Shkel, and S. S. Kia. UWB sensor placement for foot-to-foot ranging in dual-foot mounted ZUPT-aided INS. *IEEE Sensors Letters*, 2022.

[32] C. Chen, C.-S. Jao, A. M. Shkel, and S. S. Kia. UWB Sensor Placement for Foot-to-Foot Ranging in Dual-Foot-Mounted ZUPT-Aided INS. *IEEE Sensors Letters*, 6(2):1–4, 2022.

[33] C. Chen, X. Lu, A. Markham, and N. Trigoni. IONet: Learning to cure the curse of drift in inertial odometry. *AAAI Conference on Artificial Intelligence*, Feb. 2—7, 2018.

[34] L. Chen, H. Kuusniemi, Y. Chen, L. Pei, T. Kröger, and R. Chen. Motion restricted information filter for indoor bluetooth positioning. *International Journal of Embedded and Real-Time Communication Systems (IJERTCS)*, 3(3):54–66, 2012.

[35] L.-H. Chen, E. H.-K. Wu, M.-H. Jin, and G.-H. Chen. Intelligent fusion of Wi-Fi and inertial sensor-based positioning systems for indoor pedestrian navigation. *IEEE Sensors Journal*, 14(11):4034–4042, 2014.

[36] P. Chen, Y. Kuang, and X. Chen. A UWB/improved PDR integration algorithm applied to dynamic indoor positioning for pedestrians. *Sensors*, 17(9):2065, 2017.

[37] S. Chen, Z. Lu, X. Xu, J. Liu, and Z. Bi. Foot-mounted Dual-sensor Single-board Pedestrian Inertial Navigation System Based on Position and Velocity Constraints. *Sensors and Materials*, 34(6):2075–2087, 2022.

[38] Y. Chen, R. Chen, L. Pei, T. Kröger, H. Kuusniemi, J. Liu, and W. Chen. Knowledge-based error detection and correction method of a multi-sensor multi-network positioning platform for pedestrian indoor navigation. In *IEEE/ION PLANS*, Indian Wells, CA, USA, May 3—6, 2010.

[39] Z. Chen, X. Pan, C. Chen, and M. Wu. Contrastive learning of zero-velocity detection for pedestrian inertial navigation. *IEEE Sensors Journal*, 22(6):4962–4969, 2021.

[40] B. Cinaz and H. Kenn. HeadSLAM-simultaneous localization and mapping with head-mounted inertial and laser range sensors. In *12th IEEE International Symposium on Wearable Computers*, Pittsburgh, PA, USA, Sep. 28—1, 2008.

[41] R. M. Coelho, J. Gouveia, M. A. Botto, H. I. Krebs, and J. Martins. Real-time walking gait terrain classification from foot-mounted Inertial Measurement Unit using Convolutional Long Short-Term Memory neural network. *Expert Systems with Applications*, 203:117306, 2022.

[42] U. Dauderstädt, P. Sarro, and P. French. Temperature dependence and drift of a thermal accelerometer. *Sensors and Actuators A: Physical*, 66(1—3):244–249, 1998.

[43] F. de Ponte Müller. Survey on ranging sensors and cooperative techniques for relative positioning of vehicles. *Sensors*, 17(2):271, 2017.

[44] E. M. Diaz, O. Heirich, M. Khider, and P. Robertson. Optimal sampling frequency and bias error modeling for foot-mounted IMUs. In *4th IEEE IPIN*, Montbeliard, France, Oct. 28—31, 2013.

[45] P. D. Duong and Y. S. Suh. Foot pose estimation using an inertial sensor unit and two distance sensors. *Sensors*, 15(7):15888–15902, 2015.

[46] M. El-Diasty, A. El-Rabbany, and S. Pagiatakis. Temperature variation effects on stochastic characteristics for low-cost MEMS-based inertial sensor error. *Measurement Science and Technology*, 18(11):3321, 2007.

[47] N. El-Sheimy and Y. Li. Indoor navigation: State of the art and future trends. *Satellite Navigation*, 2(1):1–23, 2021.

[48] J. Elwell. Inertial navigation for the urban warrior. In *Digitization of the Battlespace IV*, volume 3709, pages 196–204. SPIE, 1999.

[49] J. A. Farrell, F. O. Silva, F. Rahman, and J. Wendel. IMU Error Modeling for State Estimation and Sensor Calibration:A Tutorial. *IEEE Control Systems Magazine*, 42(6):40–66, 2021.

[50] B. Feng, W. Tang, G. Guo, and X. Jia. An improved pedestrian tracking method based on Wi-Fi fingerprinting and pedestrian dead reckoning. *Sensors*, 20(3):853, 2020.

[51] C. Fischer and H. Gellersen. Location and navigation support for emergency responders: A survey. *IEEE Pervasive Computing*, 9(1):38–47, 2010.

[52] S. Flynn. The perfect fire, Mar. 9, 2017. [Retrieved 01-July—2022].

[53] R. Fontanella, D. Accardo, E. Caricati, S. Cimmino, and D. De Simone. An extensive analysis for the use of back propagation neural networks to perform the calibration of MEMS gyro bias thermal drift. In *IEEE/ION PLANS*, Savannah, GA, USA, Apr. 11—16, 2016.

[54] R. Fontanella, D. Accardo, E. Caricati, S. Cimmino, D. De Simone, and G. Lucignano. Improving inertial attitude measurement performance by exploiting MEMS gyros and neural thermal calibration. In *AIAA Information Systems-AIAA Infotech @ Aerospace*, Grapevine, TX, USA, Jan. 9—13, 2017.

[55] R. Fontanella, D. Accardo, R. S. L. Moriello, L. Angrisani, and D. De Simone. MEMS gyros temperature calibration through artificial neural networks. *Sensors and Actuators A: Physical*, 279:553–565, 2018.

[56] R. Fontanella, D. Accardo, R. S. L. Moriello, L. Angrisani, and D. D. Simone. An innovative strategy for accurate thermal compensation of Gyro Bias in inertial units by exploiting a novel Augmented Kalman Filter. *Sensors*, 18(5):1457, 2018.

[57] E. Foxlin. Pedestrian tracking with shoe-mounted inertial sensors. *IEEE Computer Graphics and Applications*, 25(6):38–46, 2005.

[58] J. Fuentes-Pacheco, J. Ruiz-Ascencio, and J. M. Rendón-Mancha. Visual simultaneous localization and mapping: a survey. *Artificial Intelligence Review*, 43(1):55–81, 2015.

[59] K. Gądek and M. Jaraczewski. Novel ultrasonic distance measuring system based on correlation method. *Archives of Electrical Engineering*, 63(3), 2014.

[60] G. Gallego, T. Delbrück, G. Orchard, C. Bartolozzi, B. Taba, A. Censi, S. Leutenegger, A. J. Davison, J. Conradt, K. Daniilidis, et al. Event-based vision: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(1):154–180, 2020.

[61] E. García, P. Poudereux, Á. Hernández, J. Ureña, and D. Gualda. A robust UWB indoor positioning system for highly complex environments. In *IEEE ICIT*, Seville, Spain, Mar. 17—19, 2015.

[62] M. Garcia, A. Chatterjee, A. Ruina, and M. Coleman. The simplest walking model: stability, complexity, and scaling. *ASME Journal Biomechanical Engineering*, 120(2):281–288, 1998.

[63] M. Garcia Puyol, P. Robertson, and O. Heirich. Complexity-reduced FootSLAM for indoor pedestrian navigation using a geographic tree-based data structure. *Journal of Location Based Services*, 7(3):182–208, 2013.

[64] A. Geiger, F. Moosmann, Ö. Car, and B. Schuster. Automatic camera and range sensor calibration using a single shot. In *IEEE International Conference on Robotics and Automation (ICRA)*, St Paul, MN, USA, May 14—19, 2007.

[65] R. Girisha, G. Prateek, K. Hari, and P. Händel. Fusing the navigation information of dual foot-mounted zero-velocity-update-aided inertial navigation systems. In *International Conference on Signal Processing and Communications (SPCOM)*. Bangalore, India, Jul. 22—25, 2014.

[66] S. Godha and G. Lachapelle. Foot mounted inertial system for pedestrian navigation. *Measurement Science and Technology*, 19(7):075202, 2008.

[67] S. Godha, G. Lachapelle, and M. E. Cannon. Integrated GPS/INS system for pedestrian navigation in a signal degraded environment. In *19th ION GNSS+*, Fort Worth, TX, USA, Sep. 26—29, 2006.

[68] E. Gökalp, O. Güngör, and Y. Boz. Evaluation of different outlier detection methods for GPS networks. *Sensors*, 8(11):7344–7358, 2008.

[69] P. Groves. *Principles of GNSS, Inertial, and Multisensor Integrated Navigation Systems, Second Edition*. Artech, 2013.

[70] Y. Gu, Q. Song, M. Ma, Y. Li, and Z. Zhou. Using iBeacons for trajectory initialization and calibration in foot-mounted inertial pedestrian positioning systems. In *7th IEEE IPIN*, Alcalá de Henares, Spain, Oct. 4—7, 2016.

[71] Y. Gu, C. Zhou, A. Wieser, and Z. Zhou. Pedestrian positioning using WiFi fingerprints and a foot-mounted inertial sensor. In *IEEE European Navigation Conference (ENC)*, Lausanne, Switzerland, May 9—12, 2017.

[72] Y. Gu, C. Zhou, A. Wieser, and Z. Zhou. WiFi based trajectory alignment, calibration and crowdsourced site survey using smart phones and foot-mounted IMUs. In *8th IEEE IPIN*, Sapporo, Japan, Sep. 18—21, 2017.

[73] Y. Gu, C. Zhou, A. Wieser, and Z. Zhou. Trajectory estimation and crowdsourced radio map establishment from foot-mounted IMUs, Wi-Fi fingerprints, and GPS positions. *IEEE sensors journal*, 19(3):1104–1113, 2018.

[74] O. Guclu and A. B. Can. A comparison of feature detectors and descriptors in RGB-D SLAM methods. In *17th International Conference Image Analysis and Recognition (ICIAR)*, Niagara Falls, Canada, Jul. 22—24, 2015. Springer.

[75] R. Harle. A survey of indoor inertial positioning systems for pedestrians. *IEEE Communications Surveys & Tutorials*, 15(3):1281–1293, 2013.

[76] J. Haverinen and A. Kemppainen. Global indoor self-localization based on the ambient magnetic field. *Robotics and Autonomous Systems*, 57(10):1028–1035, 2009.

[77] J. Heikkila, O. Silven, et al. A four-step camera calibration procedure with implicit image correction. In *Computer Vision and Pattern Recognition Conference (CVPR)*, volume 97, page 1106. Citeseer, 1997.

[78] W. Hess, D. Kohler, H. Rapp, and D. Andor. Real-time loop closure in 2D LIDAR SLAM. In *IEEE ICRA*, Stockholm, Sweden, May 16—21, 2016.

[79] G. P. Horn, R. M. Kesler, S. Kerber, K. W. Fent, T. J. Schroeder, W. S. Scott, P. C. Fehling, B. Fernhall, and D. L. Smith. Thermal response to firefighting activities in residential structure fires: impact of job assignment and suppression tactic. *Ergonomics*, 61(3):404–419, 2018.

[80] S. House, S. Connell, I. Milligan, D. Austin, T. L. Hayes, and P. Chiang. Indoor localization using pedestrian dead reckoning updated with RFID-based fiducials. In *Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, Boston, MA, USA, Aug. 30—3, 2011. IEEE.

[81] M. Jacquelin Perry. Gait analysis: normal and pathological function. *New Jersey: SLACK*, 2010.

[82] C.-S. Jao, A. A. Abdallah, C. Chen, M. Seo, S. S. Kia, Z. M. Kassas, and A. M. Shkel. Sub-meter accurate pedestrian indoor navigation system with dual ZUPT-aided INS, machine learning-aided LTE, and UWB signals. In *International Technical Meeting of the Satellite Division of The Institute of Navigation (ION GNSS+ 2022)*, Denver, CO, USA, Sep 19—23, 2022.

[83] C.-S. Jao, A. A. Abdallah, C. Chen, M.-W. Seo, S. S. Kia, Z. M. Kassas, and A. M. Shkel. PINDOC: Pedestrian Indoor Navigation System Integrating Deterministic, Opportunistic, and Cooperative Functionalities. *IEEE Sensors Journal*, 2022.

[84] C.-S. Jao, A. Parrish, and A. M. Shkel. "Sugar-Cube" PLT: A Real-time Pedestrian Localization Testbed Utilizing Foot-mounted IMU/Barometer/Ultrasonic Sensors. In *IEEE Sensors*, Virtual Conference, Oct. 31—4, 2021. IEEE.

[85] C.-S. Jao, E. Sangenis, P. Simo, A. Voloshina, and A. M. Shkel. An Inverted Pendulum Model of Walking for Predicting Navigation Uncertainty of Pedestrian in Case of Foot-mounted Inertial Sensors. In *IEEE INERTIAL*, Kauai, HI, USA, Mar. 28—31, 2023.

[86] C.-S. Jao and A. M. Shkel. Zupt-aided ins bypassing stance phase detection by using foot-instability-based adaptive covariance. *IEEE Sensors Journal*, 21(21):24338–24348, 2021.

[87] C.-S. Jao and A. M. Shkel. A reconstruction filter for saturated accelerometer signals due to insufficient FSR in foot-mounted inertial navigation system. *IEEE Sensors Journal*, 22(1):695–706, 2022.

[88] C.-S. Jao, K. Stewart, J. Conradt, E. Neftci, and A. Shkel. Zero Velocity Detector for Foot-mounted Inertial Navigation System Assisted by a Dynamic Vision Sensor. In *2020 DGON Inertial Sensors and Systems (ISS)*, Virtual Conference, Sep. 15—16, 2020.

[89] C.-S. Jao, D. Wang, J. Grasso, and A. M. Shkel. UWB-Foot-SLAM: Bounding Position Error of Foot-mounted Pedestrian INS with Simultaneously Localized UWB Beacons. In *IEEE/ION Position, Location and Navigation Symposium (PLANS)*, Monterey, CA, USA, Apr 24—27, 2023 (Accepted).

[90] C.-S. Jao, D. Wang, A. R. Parrish, and A. M. Shkel. A Neural Network Approach to Mitigate Thermal-Induced Errors in ZUPT-aided INS. In *9th IEEE INERTIAL*, Avignon, France, May 8—11, 2022.

[91] C.-S. Jao, Y. Wang, S. Askari, and A. M. Shkel. A Closed-Form Analytical Estimation of Vertical Displacement Error in Pedestrian Navigation. In *IEEE/ION PLANS*, Portland, OR, USA, Apr. 20—23, 2020.

[92] C.-S. Jao, Y. Wang, Y.-W. Lin, and A. M. Shkel. A Hybrid Barometric/Ultrasonic Altimeter for Aiding ZUPT-based Inertial Pedestrian Navigation Systems. In *33rd ION GNSS+*, Virtual Conference, Sep. 21—25, 2020.

[93] C.-S. Jao, Y. Wang, and A. M. Shkel. A Zero Velocity Detector for Foot-mounted Inertial Navigation Systems Aided by Downward-facing Range Sensor. In *IEEE Sensors Conference*, Virtual Conference, Oct. 25—28, 2020.

[94] C.-S. Jao, Y. Wang, and A. M. Shkel. Pedestrian Inertial Navigation System Augmented by Vision-Based Foot-to-foot Relative Position Measurements. In *IEEE/ION PLANS*, Portland, OR, USA, Apr. 20—23, 2020.

[95] A. R. Jimenez, F. Seco, C. Prieto, and J. Guevara. A comparison of pedestrian dead-reckoning algorithms using a low-cost MEMS IMU. In *IEEE International Symposium on Intelligent Signal Processing*, Budapest, Hungary, Aug. 26—28, 2009.

[96] A. R. Jiménez, F. Seco, J. C. Prieto, and J. Guevara. Indoor pedestrian navigation using an INS/EKF framework for yaw drift reduction and a foot-mounted IMU. In *7th IEEE Workshop on Positioning, Navigation and Communication (WPNC)*, Dresden, Germany, Mar. 11—12, 2010.

[97] A. R. Jiménez, F. Seco, F. Zampella, J. C. Prieto, and J. Guevara. PDR with a foot-mounted IMU and ramp detection. *Sensors*, 11(10):9393–9410, 2011.

[98] Y. Jiong, Z. Lei, D. Jiangping, S. Rong, and W. Jianyu. GPS/SINS/BARO integrated navigation system for UAV. In *IEEE International Forum on Information Technology and Applications (IFITA)*, Kunming, China, Jul. 16—18, 2010.

[99] J. Johnson and C. Taylor. Relative Magnetic Position and Rotation Sensor Assisted Dual Foot Pedestrian Dead Reckoning. In *ION ITM*, Mexico City, Mexico, Jan. 25—27, 2022.

[100] H. Ju, J. H. Lee, and C. G. Park. Pedestrian dead reckoning system using dual IMU to consider heel strike impact. In *International Conference on Control, Automation and Systems (ICCAS)*, PyeongChang, GangWon Province, Korea, Oct. 17—20, 2018.

[101] H. Ju, M. S. Lee, S. Y. Park, J. W. Song, and C. G. Park. A pedestrian dead-reckoning system that considers the heel-strike and toe-off phases when using a foot-mounted IMU. *Measurement Science and Technology*, 27(1):015702, 2015.

[102] H. Ju and C. G. Park. A pedestrian dead reckoning system using a foot kinematic constraint and shoe modeling for various motions. *Sensors and Actuators A: Physical*, 284:135–144, 2018.

[103] S. Kaiser. Integrating known locations in FootSLAM and investigating the influence of different prior information. In *8th IEEE IPIN*, Sapporo, Japan, Sep. 18—21, 2017.

[104] S. Kaiser and E. M. Diaz. PocketSLAM based on the principle of the FootSLAM algorithm. In *IEEE International Conference on Localization and GNSS (ICL-GNSS)*, Gothenburg, Sweden, Jun. 22—24, 2015.

[105] S. Kaiser and C. Lang. Integrating moving platforms in a SLAM algorithm for pedestrian navigation. *Sensors*, 18(12):4367, 2018.

[106] S. Kaiser, M. G. Puyol, and P. Robertson. Maps-based angular PDFs used as prior maps for FootSLAM. In *IEEE PLANS*, Myrtle Beach, SC, USA, Apr. 23—26, 2012.

[107] J. Käppi and K. Alanen. Pressure altitude enhanced AGNSS hybrid receiver for a mobile terminal. In *18th ION GNSS+*, Long Beach, CA, USA, Sep. 13—16, 2005.

[108] Z. Kassas. Navigation with cellular signals of opportunity. In J. Morton, F. van Diggelen, J. Spilker, Jr., and B. Parkinson, editors, *Position, Navigation, and Timing Technologies in the 21st Century*, volume 2, pages 1171–1223. Wiley-IEEE, 2021.

[109] Z. Kassas, J. Morales, K. Shamaei, and J. Khalife. LTE steers UAV. *GPS World Magazine*, 28(4):18–25, April, 2017.

[110] J.-H. Kim, J. W. Starr, and B. Y. Lattimer. Firefighting robot stereo infrared vision and radar sensor fusion for imaging through smoke. *Fire Technology*, 51(4):823–845, 2015.

[111] M. Klann. Tactical navigation support for firefighters: The LifeNet ad-hoc sensor-network and wearable system. In *International Workshop on Mobile Information Technology for Emergency Response*, pages 41–56. Springer, 2008.

[112] L. Koval, J. Vaňuš, and P. Bilík. Distance measuring by ultrasonic sensor. *IFAC-PapersOnLine*, 49(25):153–158, 2016.

[113] A. Kuo. Energetics of actively powered locomotion using the simplest walking model. *ASME Journal Biomechanical Engineering*, 124:113, 2002.

[114] A. D. Kuo. A simple model of bipedal walking predicts the preferred speed–step length relationship. *ASME Journal Biomechanical Engineering*, 123(3):264–269, 2001.

[115] A. D. Kuo, J. M. Donelan, and A. Ruina. Energetic consequences of walking like an inverted pendulum: step-to-step transitions. *Exercise and sport sciences reviews*, 33(2):88–97, 2005.

[116] R. B. Langley et al. Dilution of precision. *GPS world*, 10(5):52–59, 1999.

[117] M. Laverne, M. George, D. Lord, A. Kelly, and T. Mukherjee. Experimental validation of foot-to-foot range measurements in pedestrian tracking. In *24th ION GNSS+*, Portland, OR, USA, Sep. 20—23, 2011.

[118] F. Le Blancq. Diurnal pressure variation: the atmospheric tide. *Weather*, 66(11):306–307, 2011.

[119] J. Le Scornec, M. Ortiz, and V. Renaudin. Foot-mounted pedestrian navigation reference with tightly coupled GNSS carrier phases, inertial and magnetic data. In *8th IEEE IPIN*, Sapporo, Japan, Sep. 18—21, 2017.

[120] J. H. Lee and C. G. Park. Mitigation of a Heading Drift in Pedestrian Dead-reckoning Caused by the Sensor Bandwidth. *International Journal of Control, Automation and Systems*, 19(8):2882–2890, 2021.

[121] M. S. Lee, C. Park, and C. W. Shim. A movement-classification algorithm for pedestrian using foot-mounted IMU. In *ION ITM*, Newport Beach, CA, USA, Jan. 30—1, 2012.

[122] T. Lemaire, C. Berger, I.-K. Jung, and S. Lacroix. Vision-based SLAM: Stereo and monocular approaches. *International Journal of Computer Vision*, 74(3):343–364, 2007.

[123] X. Li, Y. Wang, and D. Liu. Research on extended kalman filter and particle filter combinational algorithm in uwb and foot-mounted imu fusion positioning. *Mobile Information Systems*, 2018, 2018.

[124] Y. Li and J. J. Wang. A robust pedestrian navigation algorithm with low cost IMU. In *3rd IEEE IPIN*, Sydney, Australia, Nov. 13—15, 2012.

[125] Z. Li, C. Song, J. Cai, R. Hua, and P. Yu. An improved pedestrian navigation system using imu and magnetometer. In *IEEE International Conference on Computer Systems, Electronics and Control (ICCSEC)*, Dalian, China, Dec. 19—21, 2017.

[126] P. Lichtsteiner, C. Posch, and T. Delbruck. A 128 $\times$ 128 120 dB 15 μs Latency Asynchronous Temporal Contrast Vision Sensor. *IEEE Journal of Solid-State Circuits*, 43(2):566–576, 2008.

[127] H.-f. Liu, W. Ren, T. Zhang, J. Gong, J.-m. Liang, B. Liu, J.-w. Shi, and Z. Huang. An adaptive selection algorithm of threshold value in zero velocity updating for personal navigation system. In *33rd Chinese Control Conference*, Nanjing, China, Jul. 28—30, 2014.

[128] R. Liu, C. Yuen, T.-N. Do, M. Zhang, Y. L. Guan, and U.-X. Tan. Cooperative positioning for emergency responders using self IMU and peer-to-peer radios measurements. *Information Fusion*, 56:93–102, 2020.

[129] D. Lymberopoulos, J. Liu, X. Yang, R. R. Choudhury, V. Handziski, and S. Sen. A realistic evaluation and comparison of indoor location technologies: Experiences and lessons learned. In *14th International Symposium on Information Processing in Sensor Networks (IPSN)*, Seattle, WA, USA, Apr. 11—14, 2015.

[130] M. Ma, Q. Song, Y. Gu, Y. Li, and Z. Zhou. An adaptive zero velocity detection algorithm based on multi-sensor fusion for a pedestrian navigation system. *Sensors*, 18(10):3261, 2018.

[131] M. Ma, Q. Song, Y. Li, and Z. Zhou. A zero velocity intervals detection algorithm based on sensor fusion for indoor pedestrian navigation. In *2nd IEEE Information Technology, Networking, Electronic and Automation Control Conference (ITNEC)*, Chengdu, China, Dec. 15—17, 2017.

[132] M. Maaref and Z. Kassas. Autonomous integrity monitoring for vehicular navigation with cellular signals of opportunity and an IMU. *IEEE Transactions on Intelligent Transportation Systems*, 2021. accepted.

[133] L. Mainetti, L. Patrono, and I. Sergi. A survey on indoor positioning systems. In *22nd IEEE International Conference on Software, Telecommunications and Computer Networks (SoftCOM)*, Split, Croatia, Sep. 17—19, 2014.

[134] J. Morales and Z. Kassas. Optimal collaborative mapping of terrestrial transmitters: receiver placement and performance characterization. *IEEE Transactions on Aerospace and Electronic Systems*, 54(2):992–1007, April, 2018.

[135] J. Morales, J. Khalife, and Z. Kassas. Information fusion strategies for collaborative inertial radio SLAM. *IEEE Transactions on Intelligent Transportation Systems*, 2021. accepted.

[136] A. I. Mourikis and S. I. Roumeliotis. A multi-state constraint Kalman filter for vision-aided inertial navigation. In *IEEE ICRA*, Rome, Italy, Apr. 10—14, 2007.

[137] E. Mueggler, B. Huber, and D. Scaramuzza. Event-based, 6-DOF pose tracking for high-speed maneuvers. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Chicago, IL, USA, Sep. 14—18, 2014.

[138] E. Munoz Diaz. Inertial pocket navigation system: Unaided 3D positioning. *Sensors*, 15(4):9156–9178, 2015.

[139] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardos. ORB-SLAM: a versatile and accurate monocular SLAM system. *IEEE Transactions on Robotics*, 31(5):1147–1163, 2015.

[140] R. Mur-Artal and J. D. Tardós. Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras. *IEEE Transactions on Robotics*, 33(5):1255–1262, 2017.

[141] J.-O. Nilsson, A. K. Gupta, and P. Händel. Foot-mounted inertial navigation made easy. In *5th IEEE IPIN*, Busan, Korea, Oct. 27−30, 2014.

[142] J.-O. Nilsson and I. Skog. Inertial sensor arrays—A literature review. In *IEEE ENC*, Helsinki, Finland, May 30−2, 2016.

[143] J.-O. Nilsson, I. Skog, and P. Händel. A note on the limitations of ZUPTs and the implications on sensor error modeling. In *3rd IPIN*, Sydney, Australia, Nov. 13−15, 2012.

[144] J.-O. Nilsson, I. Skog, P. Händel, and K. Hari. Foot-mounted INS for everybody-an open-source embedded implementation. In *IEEE/ION PLANS*, Myrtle Beach, SC, USA, Apr. 23−26, 2012.

[145] J.-O. Nilsson, D. Zachariah, I. Skog, and P. Händel. Cooperative localization by dual foot-mounted inertial sensors and inter-agent ranging. *EURASIP Journal on Advances in Signal Processing*, 2013(1):1–17, 2013.

[146] X. Niu, Y. Li, J. Kuang, and P. Zhang. Data Fusion of Dual Foot-Mounted IMU for Pedestrian Navigation. *IEEE Sensors Journal*, 19(12):4577–4584, 2019.

[147] X. Niu, Y. Li, H. Zhang, Q. Wang, and Y. Ban. Fast thermal calibration of low-grade inertial sensors and inertial measurement units. *Sensors*, 13(9):12192–12217, 2013.

[148] A. Norrdine, Z. Kasmi, and J. Blankenbach. Step detection for ZUPT-aided inertial pedestrian navigation system using foot-mounted permanent magnet. *IEEE Sensors Journal*, 16(17):6766–6773, 2016.

[149] L. Ojeda and J. Borenstein. Non-GPS navigation for security personnel and first responders. *The Journal of Navigation*, 60(3):391–407, 2007.

[150] F. Olsson, J. Rantakokko, and J. Nygårds. Cooperative localization using a foot-mounted inertial navigation system and ultrawideband ranging. In *5th IEEE IPIN*, Busan, South Korea, Oct. 27−30, 2014.

[151] M. Osman, F. Viset, and M. Kok. Indoor SLAM Using a Foot-mounted IMU and the local Magnetic Field. *arXiv preprint arXiv:2203.15866*, 2022.

[152] S. Y. Park, H. Ju, and C. G. Park. Stance phase detection of multiple actions for military drill using foot-mounted IMU. In *7th IEEE IPIN*, Alcalá de Henares, Spain, Oct. 4−7, 2016.

[153] A. R. Parrish, C.-S. Jao, and A. M. Shkel. Stance Phase Detection for ZUPT-Aided INS Using Knee-Mounted IMU in Crawling Scenarios. *IEEE Sensors Letters*, 6(5):1–4, 2022.

[154] J. Parviainen, J. Kantola, and J. Collin. Differential barometry in personal navigation. In *IEEE/ION PLANS*, Monterey, CA, USA, May 5—8, 2008.

[155] P. Pascacio, S. Casteleyn, J. Torres-Sospedra, E. S. Lohan, and J. Nurmi. Collaborative indoor positioning systems: A systematic review. *Sensors*, 21(3):1002, 2021.

[156] P. Peltola, C. Hill, and T. Moore. Particle filter for context sensitive indoor pedestrian navigation. In *ICL-GNSS*, Barcelona, Spain, Jun. 28—30, 2016. IEEE.

[157] A. Peruzzi, U. Della Croce, and A. Cereatti. Estimation of stride length in level walking using an inertial measurement unit attached to the foot: A validation of the zero velocity assumption during stance. *Journal of Biomechanics*, 44(10):1991–1994, 2011.

[158] D. D. Pham and Y. S. Suh. Pedestrian navigation using foot-mounted inertial sensor and LIDAR. *Sensors*, 16(1):120, 2016.

[159] J. Pieniazek and P. Ciecinski. Thermal hysteresis in inertial sensors. In *7th IEEE International Workshop on Metrology for AeroSpace (MetroAeroSpace)*, Virtual Conference, Jun. 22—24, 2020.

[160] M. Placer and S. Kovačič. Enhancing indoor inertial pedestrian navigation using a shoe-worn marker. *Sensors*, 13(8):9836–9859, 2013.

[161] G. Puyol and M. Jesus. *Crowdsourcing Motion Maps based on FootSLAM for Reliable Indoor Pedestrian Navigation in Multistory Environments*. PhD thesis, Technische Universität München, 2017.

[162] M. G. Puyol. Merging of maps obtained with human odometry based on footslam for pedestrian navigation. *Master's Thesis, http://www. kn-s. dlr. de/indoomav/thesis-MariaGarciaPuyol. pdf, University of Malaga, Spain*, 2011.

[163] M. G. Puyol, D. Bobkov, P. Robertson, and T. Jost. Pedestrian simultaneous localization and mapping in multistory buildings using inertial sensors. *IEEE Transactions on Intelligent Transportation Systems*, 15(4):1714–1727, 2014.

[164] M. G. Puyol, P. Robertson, and O. Heirich. Complexity-reduced FootSLAM for indoor pedestrian navigation. In *3rd IEEE IPIN*, Sydney, Australia, Nov. 13—15, 2012.

[165] J. Rantakokko, J. Rydell, P. Strömbäck, P. Händel, J. Callmer, D. Törnqvist, F. Gustafsson, M. Jobs, and M. Gruden. Accurate and reliable soldier and first responder indoor positioning: multisensor systems and cooperative localization. *IEEE Wireless Communications*, 18(2):10–18, 2011.

[166] J. Rantanen, L. Ruotsalainen, M. Kirkko-Jaakkola, and M. Mäkelä. Height measurement in seamless indoor/outdoor infrastructure-free navigation. *IEEE Transactions on Instrumentation and Measurement*, 68(4):1199–1209, 2018.

[167] K. D. Rao. Integration of GPS and baro-inertial loop aided strapdown INS and radar altimeter. *IETE Journal of Research*, 43(5):383–390, 1997.

[168] V. Renaudin, O. Yalak, P. Tomé, and B. Merminod. Indoor navigation of emergency agents. *European Journal of Navigation*, 5(ARTICLE):36–45, 2007.

[169] G. Retscher. Altitude determination of a pedestrian in a multistorey building. In *Location Based Services and Telecartography*, pages 119–130. Springer, 2007.

[170] G. Retscher and Q. Fu. Integration of RFID, GNSS and DR for ubiquitous positioning in pedestrian navigation. In *20th ION GNSS+*, Fort Worth, Texas, Sep. 25—28, 2007.

[171] P. Robertson, M. Angermann, and B. Krach. Simultaneous localization and mapping for pedestrians using only foot-mounted inertial sensors. In *11th International Conference on Ubiquitous Computing (UbiComp)*, Orlando, FL, Sep. 30—3, 2009.

[172] P. Robertson, M. Frassl, M. Angermann, M. Doniec, B. J. Julian, M. G. Puyol, M. Khider, M. Lichtenstern, and L. Bruno. Simultaneous localization and mapping for pedestrians using distortions of the local magnetic field intensity in large indoor environments. In *4th IEEE IPIN*, Montbeliard-Belfort, France, Oct. 28—31, 2013.

[173] P. Robertson, M. G. Puyol, and M. Angermann. Collaborative pedestrian mapping of buildings using inertial sensors and FootSLAM. In *24th ION GNSS+*, Portland, OR, USA, Sep. 20—23, 2011.

[174] A. R. J. Ruiz and F. S. Granja. Comparing ubisense, bespoon, and decawave uwb location systems: Indoor performance analysis. *IEEE Transactions on instrumentation and Measurement*, 66(8):2106–2117, 2017.

[175] A. R. J. Ruiz, F. S. Granja, J. C. P. Honorato, and J. I. G. Rosas. Accurate pedestrian indoor navigation by tightly coupling foot-mounted IMU and RFID measurements. *IEEE Transactions on Instrumentation and measurement*, 61(1):178–189, 2011.

[176] A. R. J. J. Ruiz, F. S. Granja, J. C. P. Honorato, and J. I. G. Rosas. Pedestrian indoor navigation by aiding a foot-mounted IMU with RFID signal strength measurements. In *1st IEEE IPIN*, Zurich, Switzerland, Sep. 15—17, 2010.

[177] A. M. Sabatini and V. Genovese. A sensor fusion method for tracking vertical velocity and height based on inertial and barometric altimeter measurements. *Sensors*, 14(8):13324–13347, 2014.

[178] E. Sangenis, C.-S. Jao, and A. M. Shkel. SVM-based Motion Classification Using Foot-mounted IMU for ZUPT-aided INS. In *IEEE Sensors*, Dalla, TX, USA, Oct 30—2, 2022.

[179] E. Schubert and M. Scholz. Evaluation of wireless sensor technologies in a firefighting environment. In *7th IEEE International Conference on Networked Sensing Systems (INSS)*, pages 157–160, 2010.

[180] Y. Shu, Y. Huang, J. Zhang, P. Coué, P. Cheng, J. Chen, and K. G. Shin. Gradient-based fingerprinting for indoor localization and tracking. *IEEE Transactions on Industrial Electronics*, 63(4):2424–2433, 2015.

[181] I. Skog, P. Handel, J.-O. Nilsson, and J. Rantakokko. Zero-velocity detection—An algorithm evaluation. *IEEE Transactions On Biomedical Engineering*, 57(11):2657–2666, 2010.

[182] I. Skog, G. Hendeby, and F. Gustafsson. Magnetic odometry-a model-based approach using a sensor array. In *21st IEEE 2018 21st International Conference on Information Fusion (FUSION)*, Cambridge, United Kingdom, Jul. 10—13, 2018.

[183] I. Skog, J.-O. Nilsson, and P. Händel. An open-source multi inertial measurement unit (MIMU) platform. In *IEEE International Symposium on Inertial Sensors and Systems (ISISS)*, Dana Point, USA, Feb. 25—26, 2014.

[184] I. Skog, J.-O. Nilsson, and P. Händel. Pedestrian tracking using an IMU array. In *IEEE International Conference on Electronics, Computing and Communication Technologies (CONECCT)*, Bangalore, India, Jul. 2—4, 2014.

[185] I. Skog, J.-O. Nilsson, P. Händel, and A. Nehorai. Inertial sensor arrays, maximum likelihood, and cramér–rao bound. *IEEE Transactions on Signal Processing*, 64(16):4218–4227, 2016.

[186] I. Skog, J.-O. Nilsson, D. Zachariah, and P. Händel. Fusing the information from two navigation systems using an upper bound on their maximum spatial separation. In *3rd IEEE IPIN*, Sydney, Australia, Nov. 13—15, 2012.

[187] V. S. Sokolovic, G. Dikic, and R. Stancic. Integration of INS, GPS, magnetometer and barometer for improving accuracy navigation of the vehicle. *Defence Science Journal*, 63(5):451–455, 2013.

[188] A. Solin, M. Kok, N. Wahlström, T. B. Schön, and S. Särkkä. Modeling and interpolation of the ambient magnetic field by Gaussian processes. *IEEE Transactions on Robotics*, 34(4):1112–1127, 2018.

[189] R. Stirling, J. Collin, K. Fyfe, and G. Lachapelle. An innovative shoe-mounted pedestrian navigation system. *ENC GNSS*, Apr. 22—24, 2003.

[190] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers. A benchmark for the evaluation of RGB-D SLAM systems. In *IEEE/RSJ IROS*, Algarve, Portugal, Oct. 7—12, 2012.

[191] W. Sun, W. Ding, H. Yan, and S. Duan. Zero velocity interval detection based on a continuous hidden markov model in micro inertial pedestrian navigation. *Measurement Science and Technology*, 29(6):065103, 2018.

[192] Q. Tang, X. Wang, Q. Yang, and C. Liu. Static temperature analysis and compensation of MEMS gyroscopes. *International Journal of Metrology and Quality Engineering*, 4(3):209–214, 2013.

[193] M. Tanigawa, H. Luinge, L. Schipper, and P. Slycke. Drift-free dynamic height sensor using mems imu aided by mems pressure sensor. In *5th IEEE WPNC*, Hannover, Germany, Mar. 27, 2008.

[194] X. Tao, F. Zhu, X. Hu, W. Liu, and X. Zhang. An enhanced foot-mounted PDR method with adaptive ZUPT and multi-sensors fusion for seamless pedestrian navigation. *GPS Solutions*, 26(1):1–13, 2022.

[195] D. Tazartes. An historical perspective on inertial navigation systems. In *IEEE ISISS*, Dana Point, USA, Feb. 25—26, 2014.

[196] D. A. Tazartes. Inertial navigation: From gimbaled platforms to strapdown sensors, 2011.

[197] X. Tian, J. Chen, Y. Han, J. Shang, and N. Li. A novel zero velocity interval detection algorithm for self-contained pedestrian navigation system with inertial sensors. *Sensors*, 16(10):1578, 2016.

[198] D. Titterton, J. L. Weston, and J. Weston. *Strapdown inertial navigation technology*, volume 17. IET, 2004.

[199] I. Vallivaara, J. Haverinen, A. Kemppainen, and J. Röning. Magnetic field-based SLAM method for solving the localization problem in mobile robot floor-cleaning task. In *15th IEEE International Conference on Advanced Robotics (ICAR)*, Tallinn, Estonia, Jun. 20 - 23, 2011.

[200] A. R. Vidal, H. Rebecq, T. Horstschaefer, and D. Scaramuzza. Ultimate SLAM? Combining events, images, and IMU for robust visual SLAM in HDR and high-speed scenarios. *IEEE Robotics and Automation Letters*, 3(2):994–1001, 2018.

[201] F. Viset, J. T. Gravdahl, and M. Kok. Magnetic field norm SLAM using Gaussian process regression in foot-mounted sensors. In *IEEE European Control Conference (ECC)*, Virtual Conference, Jun. 29—2, 2021.

[202] B. Wagstaff and J. Kelly. LSTM-based zero-velocity detection for robust inertial navigation. In *9th IEEE IPIN*, Nantes, France, Sep. 24—27, 2018.

[203] B. Wagstaff, V. Peretroukhin, and J. Kelly. Improving foot-mounted inertial navigation through real-time motion classification. In *8th IEEE IPIN*, Sapporo, Japan, Sep. 18—21, 2017.

[204] J. Wahlström, A. Markham, and N. Trigoni. FootSLAM meets adaptive thresholding. *IEEE Sensors Journal*, 20(16):9351–9358, 2020.

[205] J. Wahlström and I. Skog. Fifteen years of progress at zero velocity: A review. *IEEE Sensors Journal*, 21(2):1139–1151, 2020.

[206] J. Wahlström, I. Skog, F. Gustafsson, A. Markham, and N. Trigoni. Zero-velocity detection—A Bayesian approach to adaptive thresholding. *IEEE Sensors Letters*, 3(6):1–4, 2019.

[207] U. Walder and T. Bernoulli. Context-adaptive algorithms to improve indoor positioning with inertial sensors. In *1st IEEE IPIN*, Zurich, Switzerland, Sep. 15—17, 2010.

[208] J. Wang, A. Hu, X. Li, and Y. Wang. An improved PDR/magnetometer/floor map integration algorithm for ubiquitous positioning using the adaptive unscented Kalman filter. *ISPRS International Journal of Geo-Information*, 4(4):2638–2659, 2015.

[209] L. Wang, Y. Hao, Z. Wei, and F. Wang. Thermal calibration of MEMS inertial sensors for an FPGA-based navigation system. In *3rd IEEE International Conference on Intelligent Networks and Intelligent Systems (ICINIS)*, Shenyang, China, Nov. 1—3, 2010.

[210] Q. Wang, Z. Guo, Z. Sun, X. Cui, and K. Liu. Research on the forward and reverse calculation based on the adaptive zero-velocity interval adjustment for the foot-mounted inertial pedestrian-positioning system. *Sensors*, 18(5):1642, 2018.

[211] Q. Wang, X. Zhang, X. Chen, R. Chen, W. Chen, and Y. Chen. A novel pedestrian dead reckoning algorithm using wearable EMG sensors to measure walking strides. In *UPINLBS*, Kirkkonummi, Finland, Oct. 14—15, 2010.

[212] Y. Wang. *Pedestrian Inertial Navigation - Development of Sensors and Algorithms*. PhD thesis, University of California, Irvine, 2020.

[213] Y. Wang, S. Askari, C.-S. Jao, and A. M. Shkel. Directional ranging for enhanced performance of aided pedestrian inertial navigation. In *IEEE INERTIAL*, Naples, FL, USA, Apr. 1—5, 2019.

[214] Y. Wang, S. Askari, and A. M. Shkel. Study on Mounting Position of IMU for Better Accuracy of ZUPT-Aided Pedestrian Inertial Navigation. In *IEEE INERTIAL*, Naples, FL, USA, Apr. 1—5, 2019.

[215] Y. Wang, H. Cheng, and M. Q.-H. Meng. Inertial Odometry Using Hybrid Neural Network with Temporal Attention for Pedestrian Localization. *IEEE Transactions on Instrumentation and Measurement*, 2022.

[216] Y. Wang, A. Chernyshoff, and A. M. Shkel. Error analysis of ZUPT-aided pedestrian inertial navigation. In *9th IEEE IPIN*, Nantes, France, Sep. 24—27, 2018.

[217] Y. Wang, A. Chernyshoff, and A. M. Shkel. Study on estimation errors in zupt-aided pedestrian inertial navigation due to imu noises. *IEEE Transactions on Aerospace and Electronic Systems*, 56(3):2280–2291, 2019.

[218] Y. Wang, C.-S. Jao, and A. M. Shkel. Scenario-dependent zupt-aided pedestrian inertial navigation with sensor fusion. *Gyroscopy and Navigation*, 12(1):1–16, 2021.

[219] Y. Wang and X. Li. The IMU/UWB fusion positioning algorithm based on a particle filter. *ISPRS International Journal of Geo-Information*, 6(8):235, 2017.

[220] Y. Wang and X. Li. Graph-optimization-based zupt/uwb fusion algorithm. *ISPRS International Journal of Geo-Information*, 7(1):18, 2018.

[221] Y. Wang, X. Li, K. S. Khoshelham, and P. Li. Robust iterated extended kalman filter algorithm for foot-mounted inertial measurement units/ultrawideband fusion positioning. *Journal of Applied Remote Sensing*, 13(2):024510, 2019.

[222] Y. Wang, Y.-W. Lin, S. Askari, C.-S. Jao, and A. M. Shkel. Compensation of Systematic Errors in ZUPT-Aided Pedestrian Inertial Navigation. In *IEEE/ION PLANS*, Portland, OR, USA, Apr. 20—23, 2020.

[223] Y. Wang and A. M. Shkel. Adaptive threshold for zero-velocity detector in ZUPT-aided pedestrian inertial navigation. *IEEE Sensors Letters*, 3(11):1–4, 2019.

[224] Y. Wang and A. M. Shkel. A review on zupt-aided pedestrian inertial navigation. In *27th Saint Petersburg International Conference on Integrated Navigation Systems (ICINS)*, Saint Petersburg, Russia, May 25—27, 2020.

[225] Y. Wang and A. M. Shkel. Learning-Based Floor-Type Identification in the ZUPT-Aided Pedestrian Inertial Navigation. *IEEE Sensors Letters*, 5(3):1–4, 2021.

[226] Y. Wang, D. Vatanparvar, A. Chernyshoff, and A. M. Shkel. Analytical Closed-Form Estimation of Position Error on ZUPT-Augmented Pedestrian Inertial Navigation. *IEEE Sensors Letters*, 2(4):1–4, 2018.

[227] Y.-H. Wang, C.-P. Chen, C.-M. Chang, C.-P. Lin, C.-H. Lin, L.-M. Fu, and C.-Y. Lee. Mems-based gas flow sensors. *Microfluidics and nanofluidics*, 6:333–346, 2009.

[228] Z. Wang, X. Cheng, and J. Du. Thermal Modeling and Calibration Method in Complex Temperature Field for Single-Axis Rotational Inertial Navigation System. *Sensors*, 20(2):384, 2020.

[229] D. Weikersdorfer, D. B. Adrian, D. Cremers, and J. Conradt. Event-based 3D SLAM with a depth-augmented dynamic vision sensor. In *IEEE ICRA*, Hong Kong, May 31—5, 2014.

[230] D. Weikersdorfer, R. Hoffmann, and J. Conradt. Simultaneous localization and mapping for event-based vision systems. In *International Conference on Computer Vision Systems (ICVS)*, St. Petersburg, Russia, Jul. 16—18, 2013. Springer.

[231] K. Wen, K. Yu, Y. Li, S. Zhang, and W. Zhang. A new quaternion Kalman filter based foot-mounted IMU and UWB tightly-coupled method for indoor pedestrian navigation. *IEEE Transactions on Vehicular Technology*, 69(4):4340–4352, 2020.

[232] K. Witrisal, S. Hinteregger, J. Kulmer, E. Leitinger, and P. Meissner. High-accuracy positioning for indoor applications: RFID, UWB, 5G, and beyond. In *IEEE International Conference on RFID*, shunde, China, Sep. 21—23, 2016.

[233] O. Woodman and R. Harle. Pedestrian localisation for indoor environments. In *10th International Conference on UbiComp*, Seoul, Korea, Sep. 21—24, 2008.

[234] X. Wu, Y. Wang, and G. Pottie. A non-ZUPT gait reconstruction method for ankle sensors. In *36th Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, Chicago, IL, USA, Aug. 26—30, 2014.

[235] Y. Wu and W. Shi. On calibration of three-axis magnetometer. *IEEE Sensors Journal*, 15(11):6424–6431, 2015.

[236] www.trxsystems.com.

[237] D. Xia, S. Chen, S. Wang, and H. Li. Temperature effects and compensation-control methods. *Sensors*, 9(10):8349–8376, 2009.

[238] H. Xia, X. Wang, Y. Qiao, J. Jian, and Y. Chang. Using multiple barometers to detect the floor location of smart phones with built-in barometric sensors for indoor positioning. *Sensors*, 15(4):7857–7877, 2015.

[239] N. Xiaoji, W. Yan, and K. Jian. A pedestrian pos for indoor mobile mapping system based on foot-mounted visual–inertial sensors. *Measurement*, 199:111559, 2022.

[240] Z. Xinxi, Z. Rong, G. Meifeng, C. Gaofeng, N. Shulai, and L. Jinglong. The performance impact evaluation on bias of gyro and accelerometer for foot-mounted INS. In *12th IEEE International Conference on Electronic Measurement & Instruments (ICEMI)*, Nanjing, Jiangsu Province, China, Nov. 2—4, 2015.

[241] D. Xu, Z. Yang, H. Zhao, and X. Zhou. A temperature compensation method for MEMS accelerometer based on LM_BP neural network. In *IEEE SENSORS*, Orlando, FL, USA, Oct. 30—2, 2016.

[242] G. Xu, T. Meng, and H. Zhang. Height Estimation of Ultrasonic Array Based on Integrated Navigation for UAVs. In *IEEE Chinese Control And Decision Conference (CCDC)*, Nanchang, China, Jun. 3—5, 2019.

[243] Y. Xu, T. Shen, X.-Y. Chen, L.-L. Bu, and N. Feng. Predictive adaptive Kalman filter and its application to INS/UWB-integrated human localization with missing UWB-based measurements. *International Journal of Automation and Computing*, 16(5):604–613, 2019.

[244] Y. Xu, G. Tian, and X. Chen. Enhancing INS/UWB integrated position estimation using federated EFIR filtering. *IEEE Access*, 6:64461–64469, 2018.

[245] A. Yaakov and J. L. Gruver. Dynamic hysteresis calibration algorithm for inertial grade accelerometers. In *3rd IEEE ISISS*, Laguna Beach, CA, USA, Feb. 23—25, 2016.

[246] C. Yang and H.-R. Shao. WiFi-based indoor positioning. *IEEE Communications Magazine*, 53(3):150–157, 2015.

[247] D. Yang, J.-K. Woo, S. Lee, J. Mitchell, A. D. Challoner, and K. Najafi. A micro oven-control system for inertial sensors. *Journal of Microelectromechanical Systems*, 26(3):507–518, 2017.

[248] T. Yang, K. Kaji, and N. Kawaguchi. Elevator acceleration sensing: Design and estimation recognition algorithm using crowdsourcing. In *37th IEEE Annual Computer Software and Applications Conference Workshops*, Kyoto, Japan, Jul. 22—26, 2013.

[249] A. Yassin, Y. Nasser, M. Awad, A. Al-Dubai, R. Liu, C. Yuen, R. Raulefs, and E. Aboutanios. Recent advances in indoor localization: A survey on theoretical approaches and applications. *IEEE Communications Surveys & Tutorials*, 19(2):1327–1346, 2016.

[250] Q. Yuan, I.-M. Chen, and A. Caus. Human velocity tracking and localization using 3 imu sensors. In *6th IEEE Conference on Robotics, Automation and Mechatronics (RAM)*, Manila, Philippines, Dec. 12—15, 2013.

[251] F. Zampella, A. De Angelis, I. Skog, D. Zachariah, and A. Jimenez. A constraint approach for UWB and PDR fusion. In *3rd IEEE IPIN*, Sydney, Australia, Nov. 13—15, 2012.

[252] F. Zampella, F. Seco, et al. Robust indoor positioning fusing PDR and RF technologies: The RFID and UWB case. In *4th IEEE IPIN*, Montbeliard-Belfort, France, Oct. 28—31, 2013.

[253] J. Zhang, E. Edwan, J. Zhou, W. Chai, and O. Loffeld. Performance investigation of barometer aided GPS/MEMS-IMU integration. In *IEEE/ION PLANS*, Myrtle Beach, SC, USA, Apr. 23—26, 2012.

[254] K. Zhang, M. Zhu, G. Retscher, F. Wu, and W. Cartwright. Three-dimension indoor positioning algorithms using an integrated RFID/INS system in multi-storey buildings. In *Location Based Services and TeleCartography II*, pages 373–386. Springer, 2009.

[255] R. Zhang, F. Hoeflinger, O. Gorgis, and L. M. Reindl. Indoor localization using inertial sensors and ultrasonic rangefinder. In *International Conference on Wireless Communications and Signal Processing (WCSP)*, Nanjing, China, Nov. 9—11, 2011.

[256] R. Zhang, H. Yang, F. Höflinger, and L. M. Reindl. Adaptive zero velocity update based on velocity classification for pedestrian tracking. *IEEE Sensors Journal*, 17(7):2137–2145, 2017.

[257] W. Zhang, X. Li, D. Wei, X. Ji, and H. Yuan. A foot-mounted PDR system based on IMU/EKF+ HMM+ ZUPT+ ZARU+ HDR+ compass algorithm. In *8th IEEE IPIN*, Sapporo, Japan, Sep. 18—21, 2017.

[258] Y. Zhang, X. Tan, and C. Zhao. UWB/INS integrated pedestrian positioning for robust indoor environments. *IEEE Sensors Journal*, 20(23):14401–14409, 2020.

[259] Z. Zhang. A flexible new technique for camera calibration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22:1330–1334, 2000.

[260] C. Zhou, J. Downey, D. Stancil, and T. Mukherjee. A low-power shoe-embedded radar for aiding pedestrian inertial navigation. *IEEE Transactions on Microwave Theory and Techniques*, 58(10):2521–2528, 2010.

[261] J. Zhu and S. S. Kia. A loosely coupled cooperative localization augmentation to improve human geolocation in indoor environments. In *9th IEEE IPIN*, Nantes, France, Sep. 24—27, 2018.

[262] J. Zhu and S. S. Kia. Bias compensation for UWB ranging for pedestrian geolocation applications. *IEEE Sensors Letters*, 3(9):1–4, 2019.

[263] J. Zhu and S. S. Kia. An UWB-based communication protocol design for an infrastructure-free cooperative navigation. In *IEEE/ION PLANS*, Portland, OR, USA, Apr. 20—23, 2020.

[264] J. Zhu and S. S. Kia. Decentralized cooperative localization with LoS and NLoS UWB inter-agent ranging. *IEEE Sensors Journal*, 22(6):5447–5456, 2021.

[265] Y. Zhuang, J. Yang, Y. Li, L. Qi, and N. El-Sheimy. Smartphone-based indoor localization with bluetooth low energy beacons. *Sensors*, 16(5):596, 2016.

[266] Z. Zuo, L. Liu, L. Zhang, and Y. Fang. Indoor positioning based on Bluetooth low-energy beacons adopting graph optimization. *Sensors*, 18(11):3736, 2018.

[267] L. Zwirello, C. Ascher, G. F. Trommer, and T. Zwick. Study on UWB/INS integration techniques. In *8th IEEE WPNC*, Dresden, Germany, Apr. 7—8, 2011.

[268] L. Zwirello, X. Li, T. Zwick, C. Ascher, S. Werling, and G. F. Trommer. Sensor data fusion in UWB-supported inertial navigation systems for indoor navigation. In *IEEE ICRA*, Karlsruhe, Germany, May 6—10, 2013.

Appendix A

Pedestrian Navigation Testbeds

This appendix presents the hardware, firmware, and interface development of two multi-sensor pedestrian navigation platforms, the Lab-On-Shoe platform and the Sugar-Cube platform. Corresponding MATLAB, LabVIEW, and C/C++ Sources codes are included. Section A.1 discusses different variations of the developed Lab-On-Shoe platform, and Section A.2 describes designs and implementations of the developed Sugar-Cube platform.

A.1 Lab-On-Shoe Platform

This section discusses the development of the Lab-On-Shoe platform. The Lab-On-Shoe platform was designed to be a flexible system allowing fast investigation of the impacts of different sensing modalities. There were two implementations of the Lab-On-Shoe platforms, referred to as Lab-On-Shoe 1 and Lab-On-Shoe 2, respectively. Lab-On-Shoe 1 is presented in Section A.1.1 and Lab-On-Shoe 2 is discussed in Section A.1.2.

A.1.1 Lab-On-Shoe 1: Packpack and Shoe Implementation

The Lab-On-Shoe 1 platform has nine different versions, including Lab-On-Shoe 1.0, Lab-On-Shoe 1.1, Lab-On-Shoe 1.2, Lab-On-Shoe 1.3, Lab-On-Shoe 1.4, Lab-On-Shoe 1.5, Lab-On-Shoe 1.6, Lab-On-Shoe 1.7, and Lab-On-Shoe 1.8. The version denoted with a larger includes all the functionalities of the version denoted with a smaller number. In this section, all eight versions of the Lab-On-Shoe 1 platform are discussed.

Lab-On-Shoe 1.0: FPGA Integrated With IMUs and Ultrasonic Transducers

Table A.1 lists COTS components used on the Lab-On-Shoe 1.0 system. This system included a backpack and a pair of shoes. The processing unit, battery, and voltage regulators were soldered and connected on a customized mother PCB, discussed later in this section, that was mounted inside a backpack with a dimension of around 20 [in] ×15 [in] ×10 [in]. One IMU and one pair of SONARs were mounted on each shoe of the system. The shoe-mounted sensors were connected to the mother PCB with 1-[m]-long cables.

Table A.1: COTS components used in Lab-On-Shoe 1.0 platform.

Component	Manufacturer	Model	Quantity	Purpose
FPGA	National Instruments	CompactRIO−9039	1	Processing Unit
IMU	Analog Device	ADIS16497−3	2	Acceleration and angular rate
SONAR	Devantech	SRF08	4	Ranging
Battery	Powerizer	LiFePO4	1	Power source
Voltage Regulator	Texas Instruments	LM3100EVAL	2	Power management
Voltage Regulator	Texas Instruments	TPS7A1601EVM−046	2	Power management

Motherboard PCB The motherboard PCB (see Figure A.1 for schematic and Figure A.2 for the assembled board) inputs power source from the LiFePO4 rechargeable battery and distributes power to the following circuit blocks: battery indicator, IMUs, and SONAR sen-

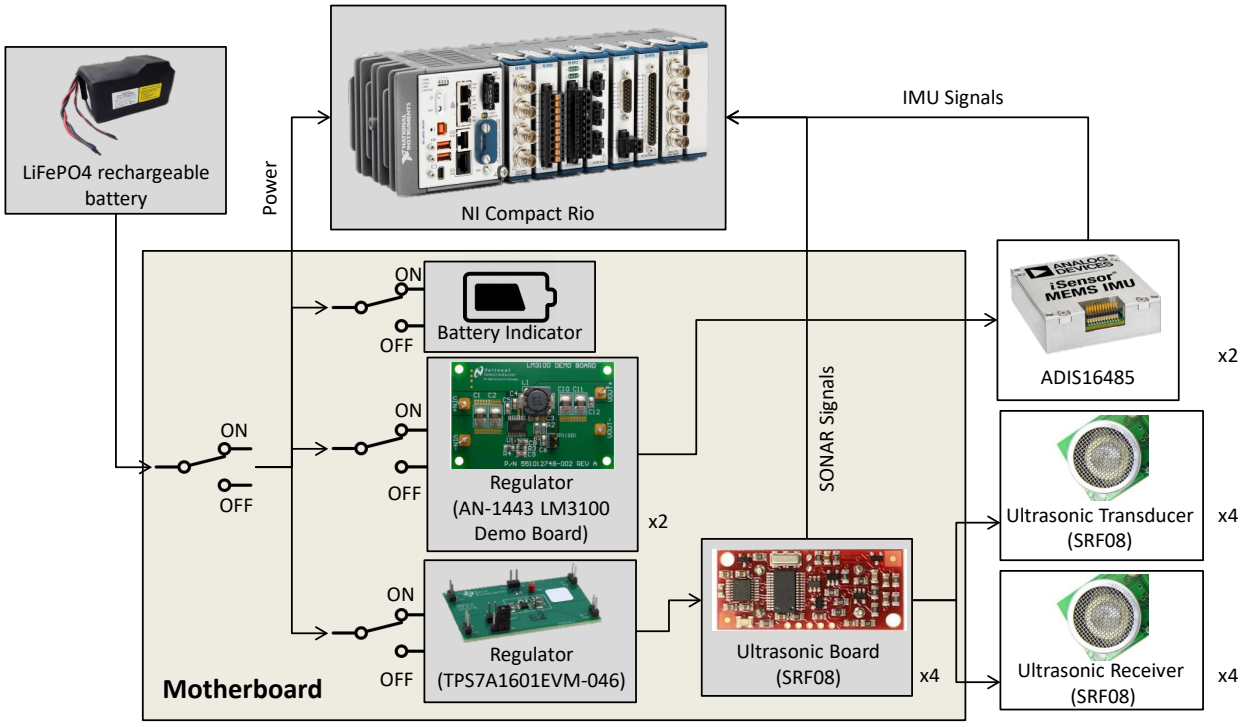


Figure A.1: Schematic of the motherboard PCB

sors. We placed two 3-port power switches (NKK Switch MS23BNW03-ND). One connects to the battery and controls the operation of the entire motherboard, the other connects to the battery indicator circuit and controls the operation of the indicator. We also placed three 2-port power switches (NKK Switch AS12AP) to turn on or off the operation of the IMU blocks and SONAR blocks, respectively.

The battery indicator circuit has five LEDs to show the battery level. Five LEDs illuminating simultaneously for 100% battery level, four for 75%, and so on. The IMU blocks use a 3.3 [V] DC voltage and maximum 1.5A DC current regulator (AN−1443 LM3100 Demo Board) for each of the IMUs. Each of the regulators was connected to two-port terminal blocks to provide power for IMU. The Sonar sensor blocks use a 5 [V] DC voltage and a maximum of 100 [mA] DC current regulator (TPS7A1601EVM−046) to provide power to all the SONAR sensors. This regulator was connected to four SRF08 SONAR sensor evaluation boards. The SONAR sensor board has four input pins, one pair for transducers and the other for

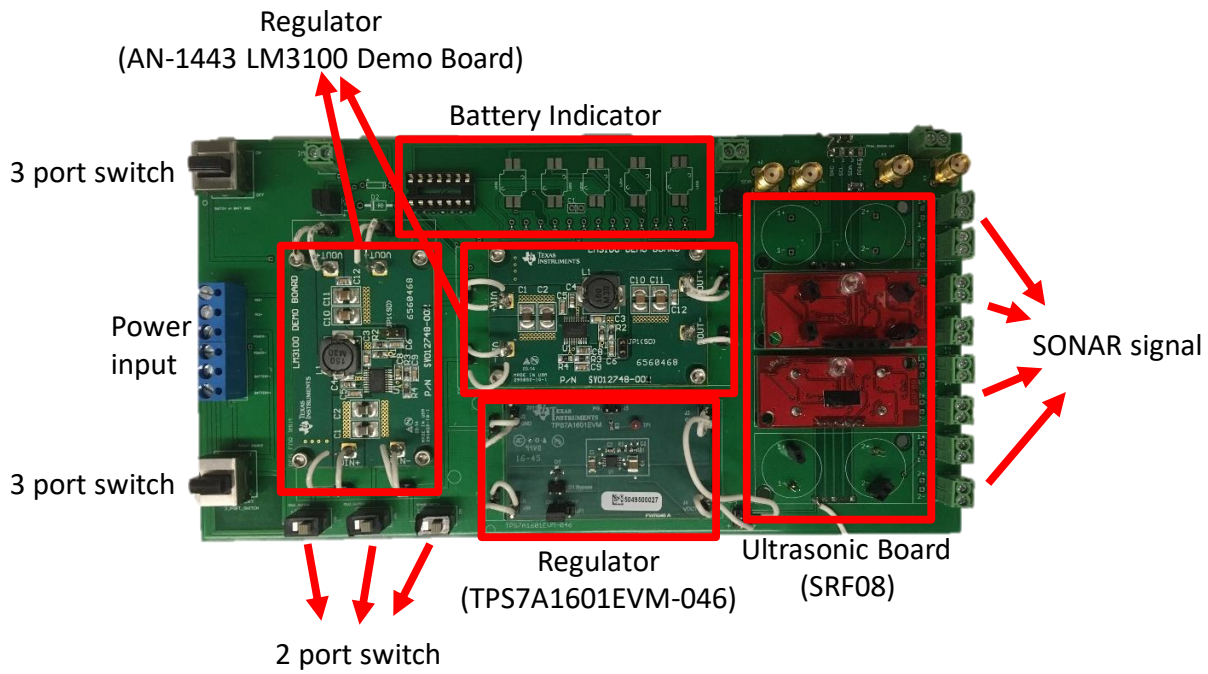


Figure A.2: The assembled motherboard PCB

receivers. We connected these pins on the four evaluation boards to 8 two-port terminal blocks. The power, SCL, SDA, and GND pins on the four evaluation boards were connected, respectively, and SCL and SDA were output to Compact-Rio via two SMA ports.

(a) (b) (c)

Figure A.3: (a) SONAR Sensor PCB layout. (b) SONAR Sensor PCB front view. (c) SONAR Sensor PCB back view

PCB for SONAR Sensor This PCB was developed for the placement transducer and receiver of the SRF08 SONAR sensor. We manually detached the transducer and the receiver

from the SRF08 evaluation board and placed each on this PCB. The assembled part was fixed on a specific face of the shoe fixture. On this PCB (see Figure A.3b and Figure A.3c), we have two ports for the sensors. These two ports were connected to a two-pin jumper to connect to the motherboard PCB via cables.

Lab-On-Shoe 1.1: Integrating With Altimeters

Table A.2: COTS components used in Lab-On-Shoe 1.1 platform.

Component	Manufacturer	Model	Quantity	Purpose
FPGA	National Instruments	CompactRIO−9039	1	Processing Unit
IMU	Analog Device	ADIS16497−3	2	Acceleration and angular rate
SONAR	Devantech	SRF08	4	Ranging
Altimeter	TE Connectivity	MS5803−01BA	2	Height
Battery	Powerizer	LiFePO4	1	Power source
Voltage Regulator	Texas Instruments	LM3100EVAL	2	Power management
Voltage Regulator	Texas Instruments	TPS7A1601EVM−046	2	Power management

Table A.2 lists COTS components used on the Lab-On-Shoe 1.1 system, which was integrated with two additional barometric altimeters, as compared to the Lab-On-Shoe 1.0 system. The altimeters were included to investigate the algorithm of the ZUPT-aided INS augmented with altimeters.

Altimeter Selection An altimeter (MS5803-series) was selected to be incorporated into the Lab-On-Shoe 1.1 platform. The altimeter was designed to have SPI and I2C communication protocol, which provides easy integration with the flexible platform that includes NI Compact-Rio, IMUs, and SONARs, presented in Figure A.4. In the case of the Lab-On-Shoe 1.1 platform, I2C communication protocol was ideal since it could be implemented on the two ports of a NI−9402 module of the CompactRIO. The operating voltage of the MS5803 altimeter was 3.3 [V], provided by the voltage regulator (AN−1443 LM3100 demo board).

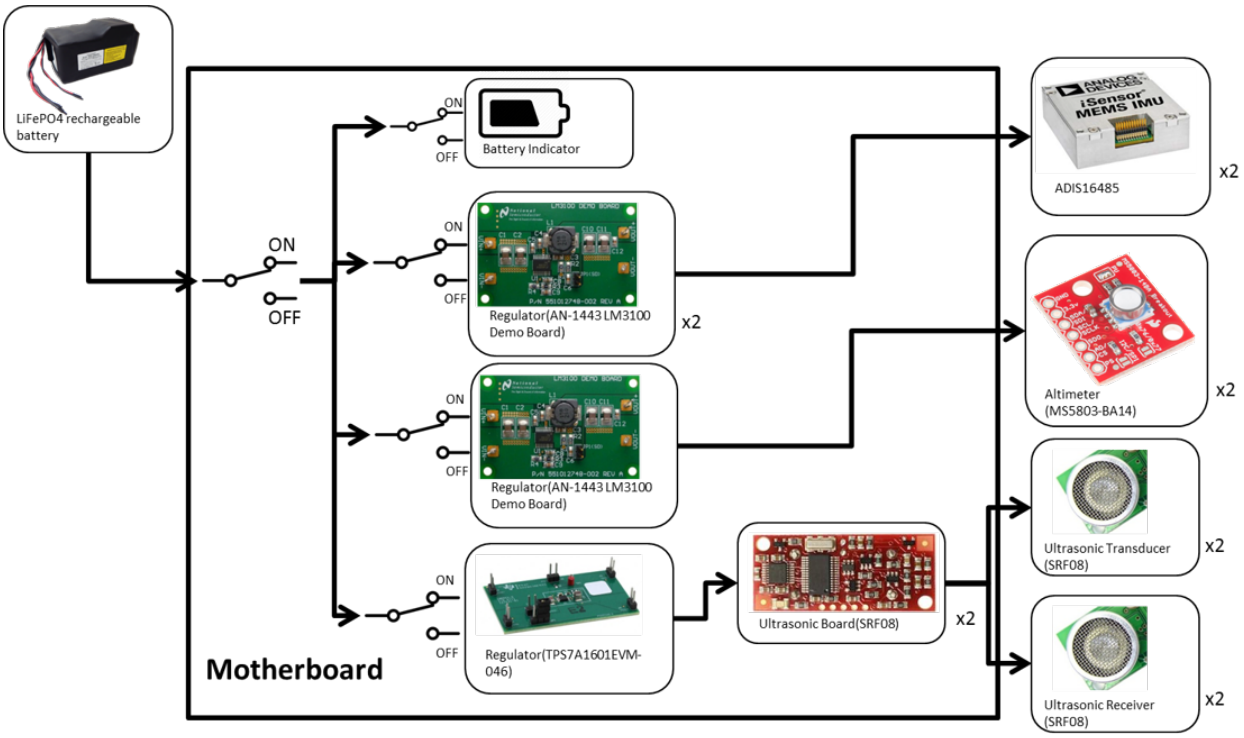


Figure A.4: Connection schematic of Lab-On-Shoe 1.1 platform.

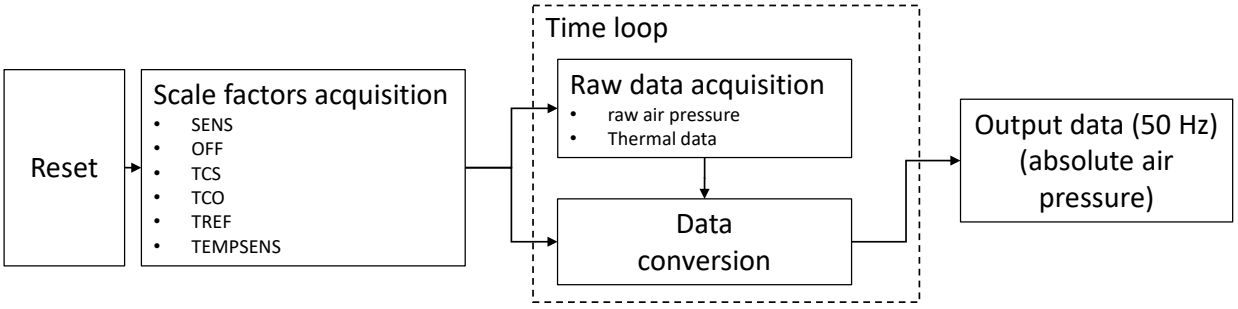


Figure A.5: I2C communication flow for MS5803 altimeter.

Altimeter Firmware An I2C implementation flow chart for the MS5803−01BA altimeter is shown in Figure A.5. Initialization was required for the I2C communication protocol to ensure that both the SDA and SCL lines were in an idle mode. The initialization was achieved by sending a reset command before implementing all the other commands. After the initialization, six scale factors must be read from the sensor at least once for each operating altimeter. These scale factors were used to calibrate the altimeter. Once the calibration coefficients were acquired, we sent analog-to-digital conversion command to read raw air pressure and temperature data in each iteration. These two data were then used with the calibration coefficients in the calculation of temperature-compensated measurements of air pressure.

The I2C for the MS5803 altimeter was implemented via LabVIEW on the Lab-On-Shoe 1.1 platform. Altimeter communication was implemented parallel to SONAR communication (On the NI−9402 module, port #0 and port #1 were used by SONARs, and port #3 and port #4 by altimeters). There was also a choice to connect altimeters in series with SONARs. However, such a configuration would limit the overall sampling frequency of I2C to 10 [Hz], which was considered insufficient in pedestrian navigation since each gait cycle was only approximately 0.5 seconds. Using the parallel connection, the system achieved sampling rates of 25 [Hz] for the integrated SONARs and 50 [Hz] for the altimeters, respectively.

PCB Design One D-sub cable of 25 pins was used to connect each shoe to the developed mother PCB. Among these 25 pins, six pins were allocated for communication with IMUs via SPI protocol, eight for SONAR sensors, and six for altimeter I2C communication.

The motherboard PCB, presented in Figure A.6, had four regulators to power IMUs, SONARs, and altimeters. Regulator TPS7A1601EVM−046 provided power for SONARs, and Regulator AN−1443 LM3100 Eval Board was used for IMUs and altimeters. Another PCB was designed to be placed on the shoe, serving as a relay to connect the CompactRIO to foot-

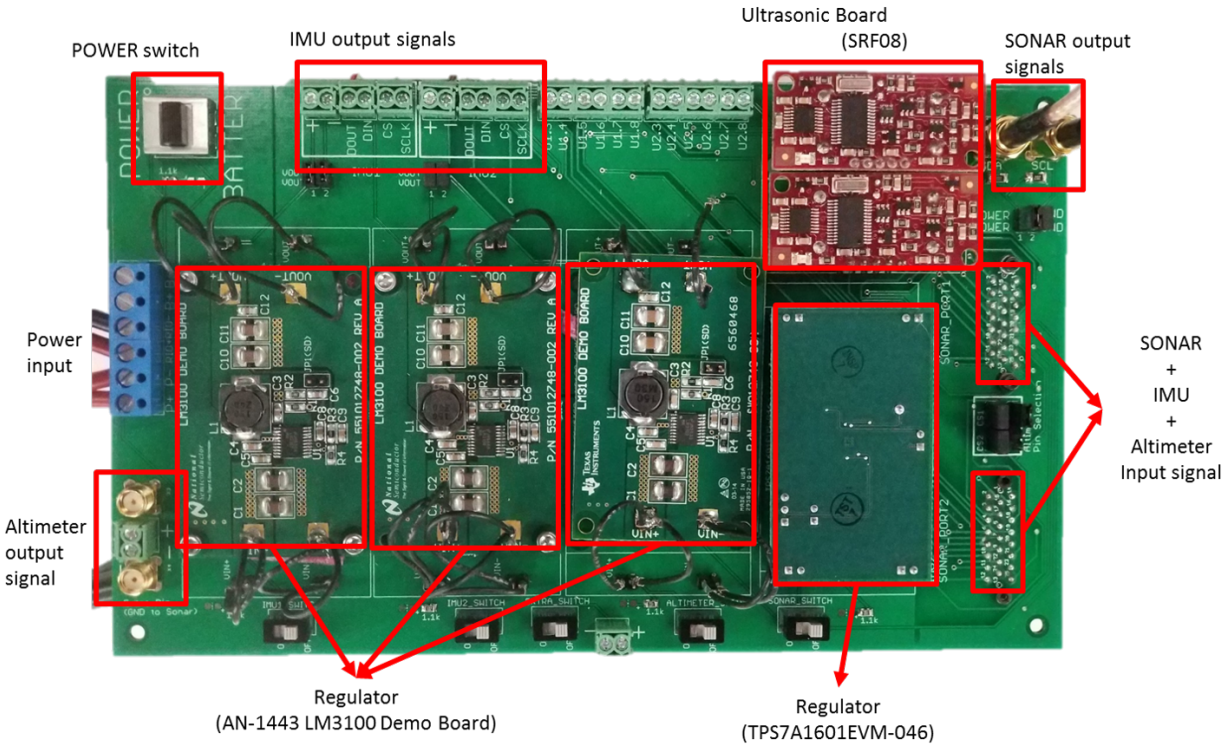


Figure A.6: A picture shown the assembled motherboard PCB used on the Lab-On-Shoe 1.1 platform.

mounted sensors. The motherboard PCB gathered all the clock signals of the I2C and SPI, sending them via the D-sub cable to the shoe PCB, and the shoe PCB then distributed power and clock signals to the sensors. The sensor data was then transmitted back to the shoe PCB, sending the data first back to the motherboard PCB via the D-sub cable, then to the CompactRIO.

Altimeter Performance Testing According to the MS5803−01BA datasheet, measurements of air pressure had a resolution of 0.2 [mbar]. To verify the resolutions of the altimeter, two series of experiments were conducted. In the first series of experiments, the MS5803−01BA altimeter remained stationary for approximately 4 hours to record data. These data were used to perform the Allan deviation analysis of the sensor. The Allan deviation is shown in Figure A.7.

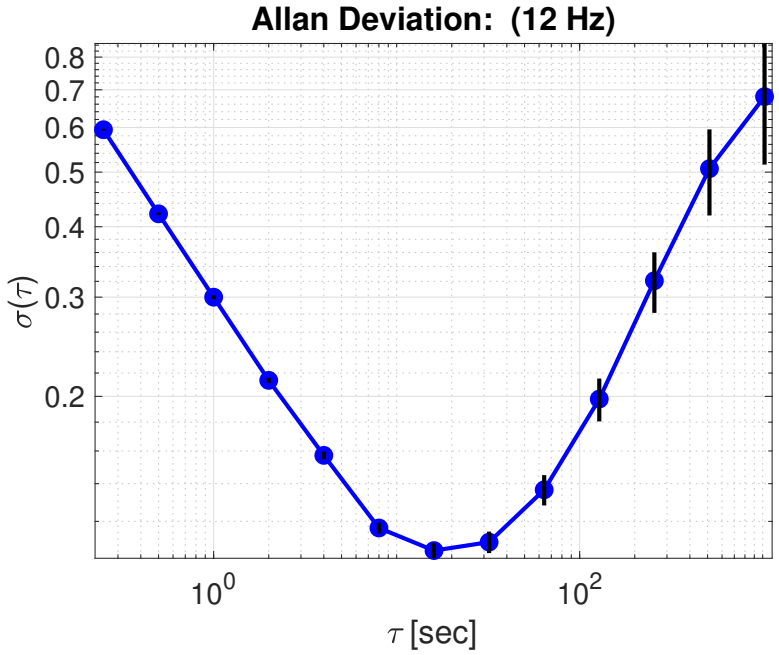


Figure A.7: Allan deviation of MS5803-BA14.

The second series of experiments involved a subject carrying the altimeter, performing three different trials: walking on indoor stairs, standing in a moving elevator, and walking on indoor stairs. The results are shown in Figure A.8. The height of the building where the experiment

was performed was approximately 15 [m]. In the experiment of walking on indoor stairs, the subject walked upstairs from the first floor to the fourth floor, stayed on the fourth floor for two seconds, and then walked back down to the first floor. This procedure was repeated three times. In the experiment of standing in the elevator, the subject stood inside an elevator, and the elevator went from the first floor to the fourth floor. The procedure was repeated three times. In the outdoor experiment, the subject repeated the same activities as in the indoor experiment. It could be observed in Figure A.8 that the collected data appears to be messy, as compared to the experiment conducted in indoors. The possible explanation was that when the subject was performing the outdoor experiments, air flow due to wind had a larger effect than the indoor environment.

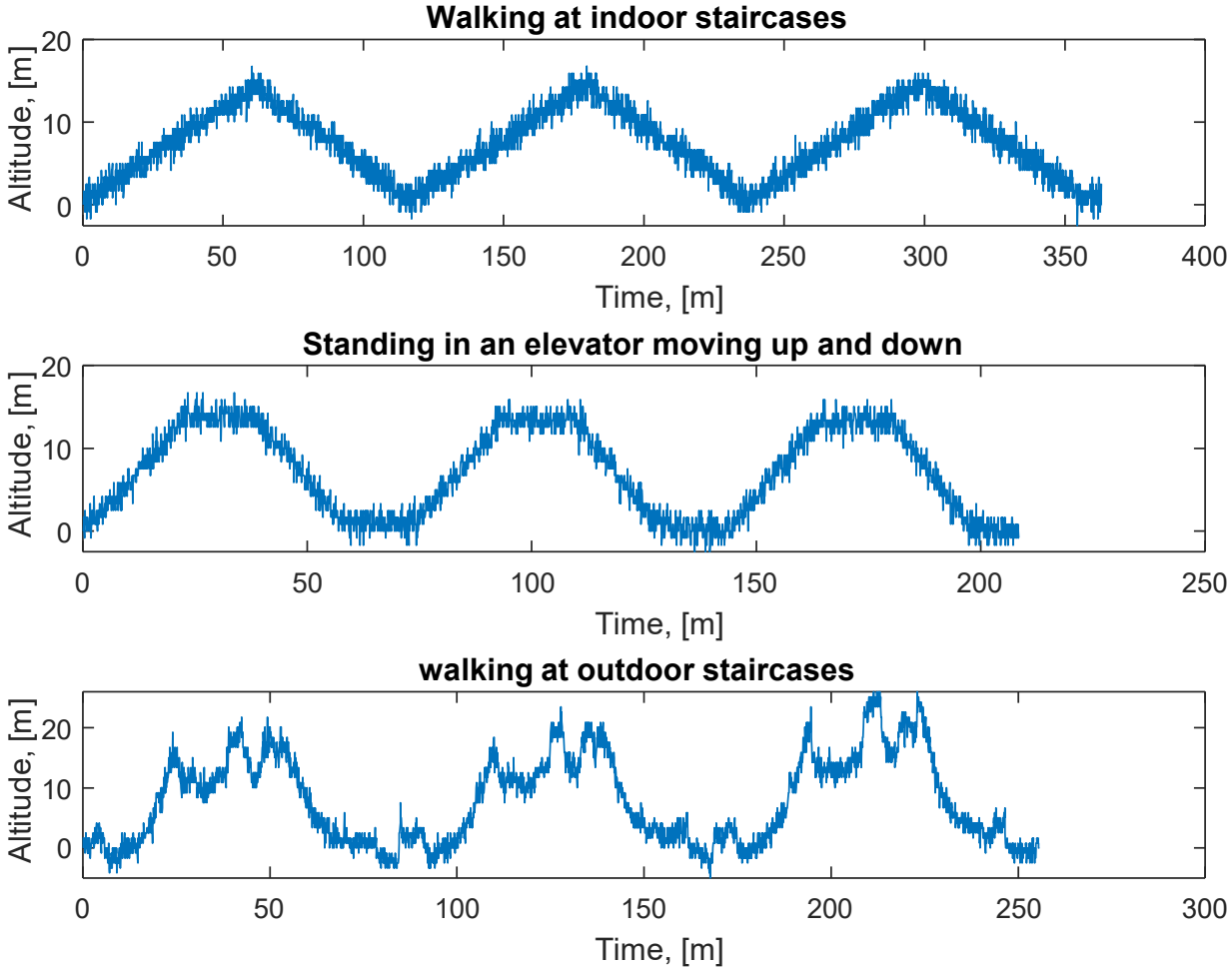


Figure A.8: Altimeter data collected from walking experiments.

Lab-On-Shoe 1.2: Real-time Navigation Using ZUPT/Altimeters/Foot-to-foot-aided INS

The Lab-On-Shoe 1.2 platform had the same hardware architecture as the Lab-On-Shoe 1.1 platform and was upgraded with an additional capability to perform real-time navigation based on a ZUPT-aided INS augmented with altimeters and SONAR-based foot-to-foot ranging.

Firmware Architecture The architecture of the Lab-On-Shoe 1.2 system is given in Figure A.9. The acquisitions of the sensors, including IMU, SONAR, and altimeter, were achieved in the FPGA module of the NI CompactRIO. The collected sensor measurements were transmitted to the Real-Time module of the NI CompactRIO. In this module, strap-down inertial navigation and extended Kalman Filter were performed to estimate the current location of an agent in a real-time fashion. The estimated location was then sent to a laptop or a mobile tablet for visualization.

LabVIEW User Interface A front-end user interface was developed using LabVIEW to control the navigation system and visualize data. The interface is shown in figure A.10. The green button was the start button of the entire system. Once it was clicked, the navigation system started the initialization step. In this step, it first collects gyroscope bias from the IMUs for 5 seconds and then estimates the accelerometer bias from EKF for 30 seconds. Both bias information were used to compensate for IMU measurements in the inertial navigation algorithm. The system was initialized for navigation after this step.

Next to the start button was a red stop button, used to terminate the entire system. The blue button next to the stop button was a reset button. It was used to reset the location of the agent to its initial location at the origin. The estimated location was presented on the plot titled North-East trajectory in Figure A.10. In the plot, the initial position was

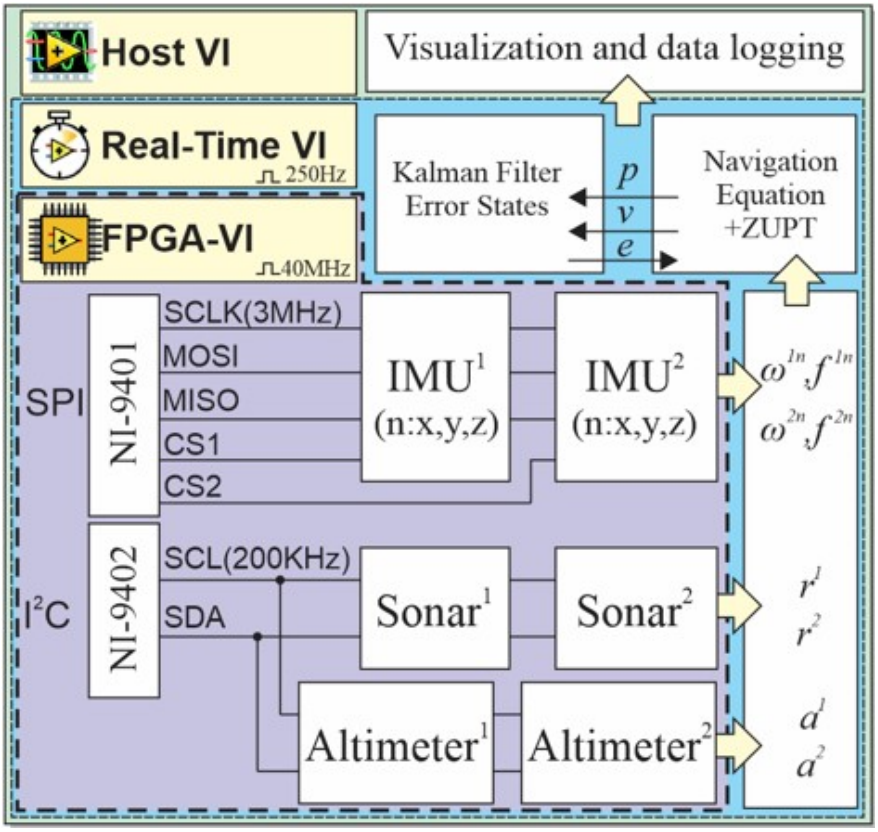


Figure A.9: Architecture of the navigation system

the origin of the trajectory, and the current location was the end of the trajectory. The bar indicator next to the North-East trajectory plot demonstrated the estimated altitude. The position of the firefighter icon indicates the current altitude of the agent.

At the bottom of the front-end interface were four plots showing the altimeter readouts, gyroscope readouts, accelerometer readouts, and ZUPT status, respectively. The elapsed time of the system is shown in the box above the stop button. The ZUPT switch and Altimeter switch were to determine whether EKF includes ZUPT or altimeter. For example, if both are turned to ON condition, the EKF uses both ZUPT and altimeter measurement. If ZUPT is ON, but the altimeter is OFF, the EKF uses only ZUPT, and the other way around is the same. If both are turned OFF, the system performs standalone inertial navigation without aiding techniques. Under the altimeter switch and ZUPT switch is an ON and OFF indicator. This indicator shows the current status of ZUPT. For example, if the foot is on

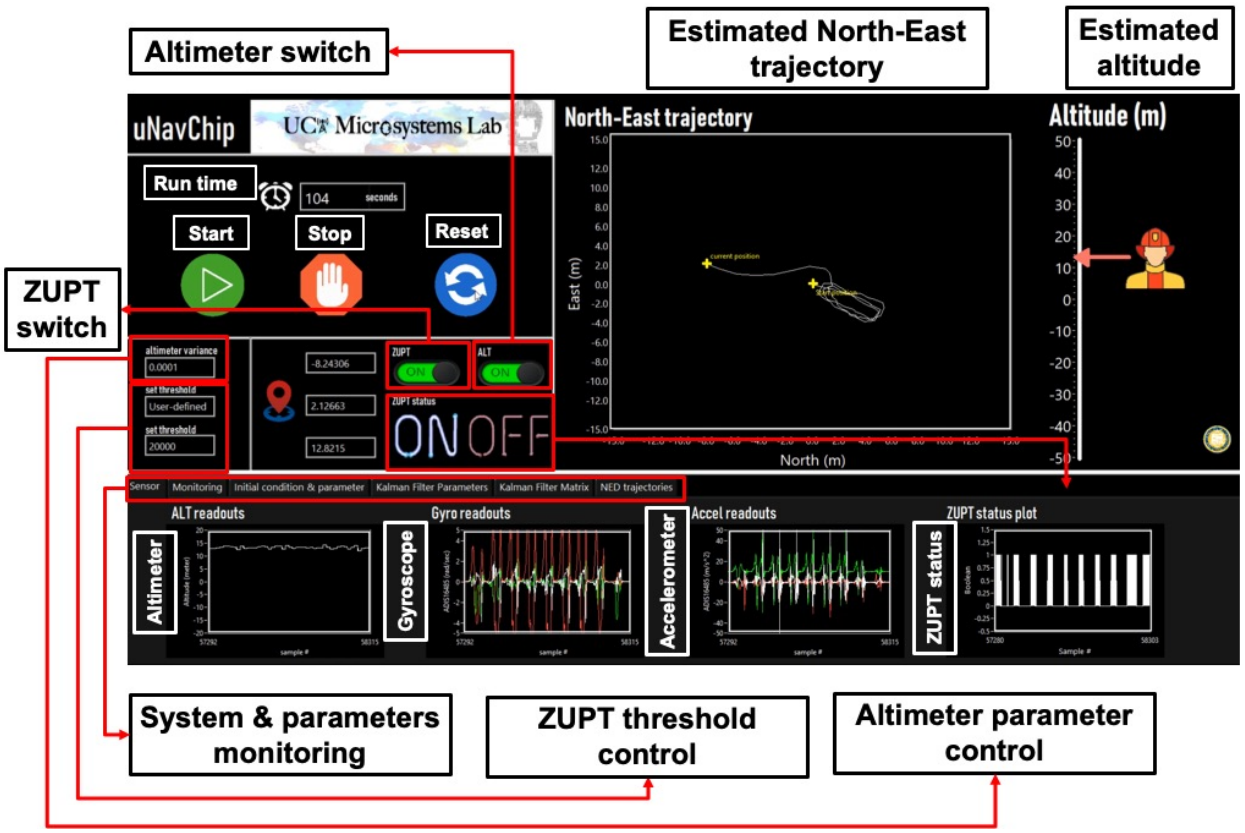


Figure A.10: Front panel of the navigation system

the ground, it would show ON. Otherwise, it shows OFF.

The current position is shown on the left-hand side of the ZUPT indicator. The altimeter parameter control determines the variance of the altimeter sensor. According to MS5803−01BA data sheets, the variance is 0.1 [m]. The LabVIEW interface was programmed to have the option to set variance to different numbers. The smaller this number is, the more the navigation solution relies on altimeter readouts, as in the EKF, the correction of position estimates is inverse proportional to the variance of the altimeter measurement.

Below the altimeter parameter control is the ZUPT threshold control. It is a scroll-down window of six items: 1, 2, 3, 4, 5, and the user-defined. Options 1, 2, 3, 4, and 5 correspond to set the ZUPT threshold as 200, 2000, 20000, 100000, and 200000, respectively. The user-defined option is to set a number as the ZUPT threshold. The system and parameter

monitoring is designed to keep track of the status of the Inertial Navigation algorithm and the EKF, ensuring the system operates under the correct conditions.

Lab-On-Shoe 1.3: Integrating With CMOS Monocular Cameras

The Lab-On-Shoe 1.3 system was upgraded based on the Lab-On-Shoe 1.1 platform and integrated with two additional Complementary Metal-Oxide Semiconductor (CMOS) monocular cameras. Table A.3 lists COTS components integrated on the Lab-On-Shoe 1.3 platform. This section discusses the specification of the selected camera module, implementation of object detection algorithms with the camera to extract positions and orientations of an object, and validation of the performance of the camera as a ranging sensor to aid INS.

Table A.3: COTS components used in Lab-On-Shoe 1.3 platform.

Component	**Manufacturer**	**Model**	**Quantity**	**Purpose**
FPGA	National Instruments	CompactRIO−9039	1	Processing Unit
IMU	Analog Device	ADIS16497−3	2	Acceleration and angular rate
SONAR	Devantech	SRF08	4	Ranging
Altimeter	TE Connectivity	MS5803−01BA	2	Height
Camera	Basler	acA800−200gc	2	Image
Battery	Powerizer	LiFePO4	1	Power source
Voltage Regulator	Texas Instruments	LM3100EVAL	2	Power management
Voltage Regulator	Texas Instruments	TPS7A1601EVM−046	2	Power management

Selection of Camera A Gigabit Ethernet (GigE) camera acA800−200gc manufactured by the Basler Camera was selected, based on consideration of image resolutions, Frame Per Second (FPS), and power consumption, to perform the task. The camera has a resolution of 600×800, a maximum FPS of 200, and a Power over Ethernet (PoE) port of 12/24V. A 4mm focal length lens with a Field Of View (FOV) of 73° was selected, and the lens has the least radial distortion among the short focal length lenses compatible with the chosen camera.

Camera Installation on the Shoes The camera was installed on top of the IMU mounted on the left shoe and a planar checkerboard pattern (referred to as the checkerboard in this section) next to the IMU of the right shoe (more on the reason for placing the checkerboard is discussed later in this section). Figure A.11 illustrates the physical characteristics of the system with a SOLIDWORKS 3D model. This arrangement was chosen to minimize the displacement between the left IMU and the camera, as well as the displacement between the right IMU and the checkerboard. The camera was placed at $[3, 0, 1]$ [cm] in the left IMU frame, and the checkerboard is placed at $[0, -1, 0]$ [cm] in the right IMU frame.

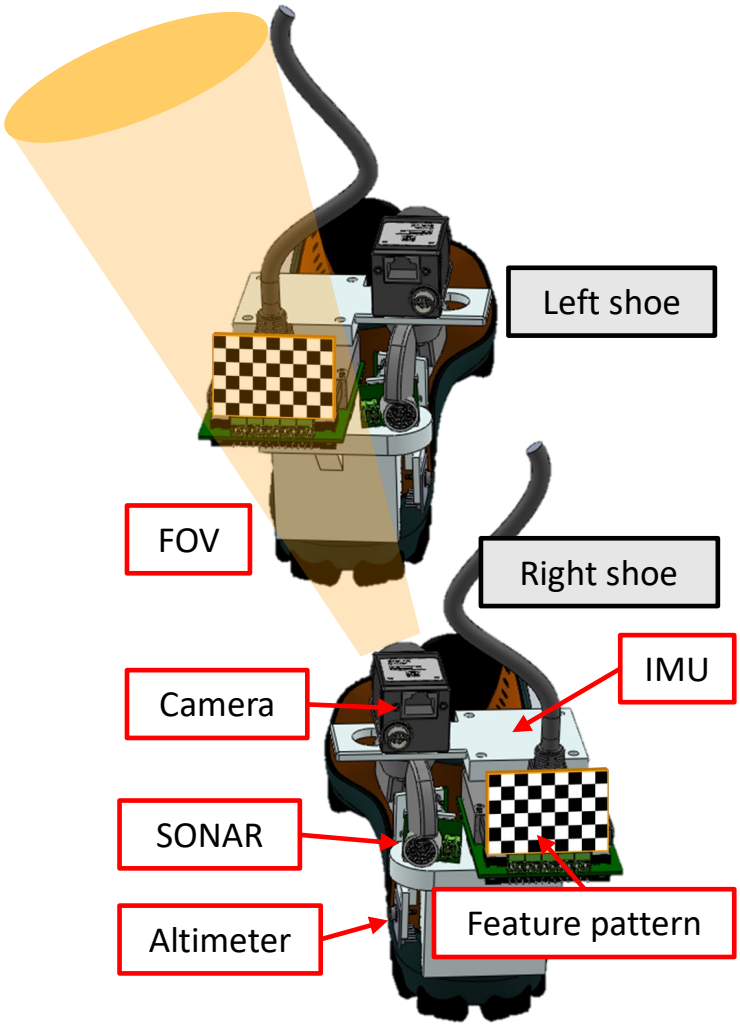


Figure A.11: A 3D model of the Lab-On-Shoe 1.3 system.

LabVIEW Firmware Implementation A LabVIEW interface was developed to simultaneously collect and save a sequence of images and IMU measurements in experiments. The cameras were interfaced with the LabVIEW installed on the computer (referred to as the computer vi in this section). The computer vi was designed to save and visualize collected data as well as control the camera operation.

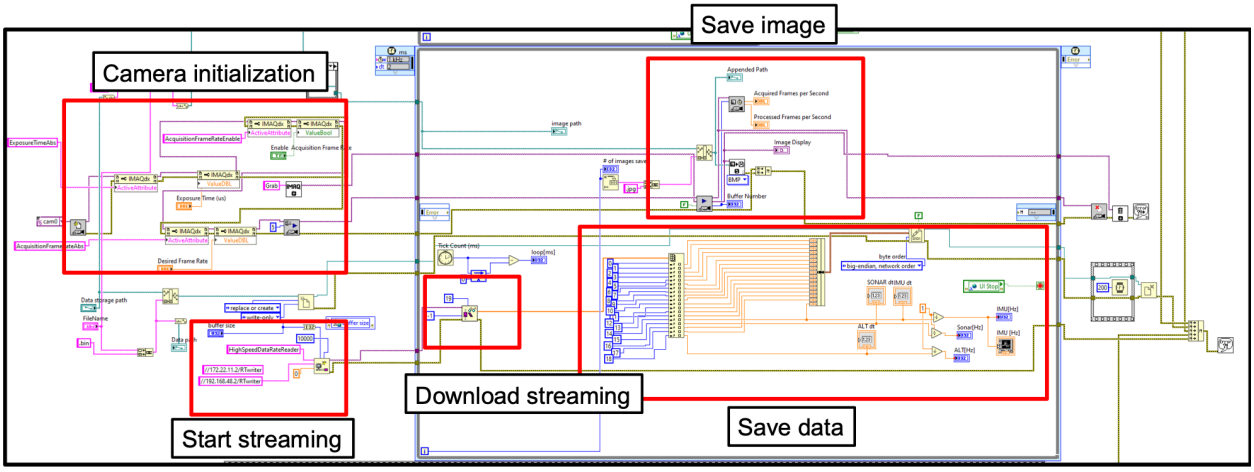


Figure A.12: Firmware architecture of the computer vi streaming data in a parallel implementation.

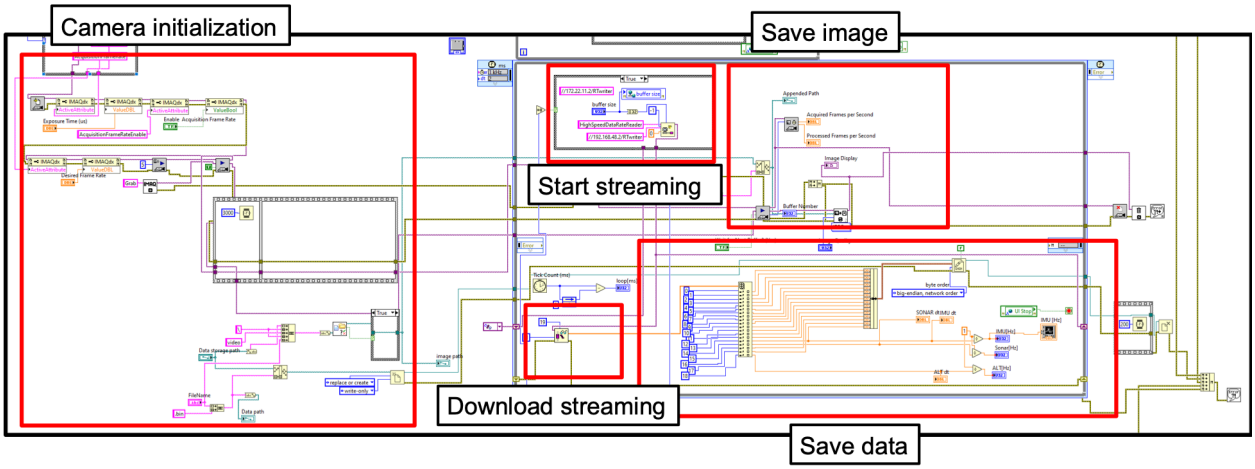


Figure A.13: Firmware architecture of the computer vi streaming data in a serial implementation.

One challenging part of the integration was that the time stamps of video data needed to be aligned with the sensor measurements. The alignment was important when fusing visual information with the inertial navigation system. In a parallel data streaming mechanism,

the Real-time module of the CompactRIO continuously uploaded the data collected from the FPGA module to the stream network, and the computer vi continuously downloaded data from the stream network once the data became available. Figure A.12 shows a part of the LabVIEW program in the computer vi. This implementation performed the initialization of the camera firmware and the creation of the stream network in parallel. The problem with this implementation was that the stream network started to download data to its buffer right after the stream was created at the computer vi while the camera was being initialized. The initialization takes around three seconds, so the time stamp of the first image being saved would correspond to the sensor data collected three seconds ago. This would lead to misalignment of the video data and measurements of sensors. A serial data streaming mechanism, shown in figure A.13, could avoid the misalignment issue by first initializing camera firmware and then creating the data stream at the first iteration of each implementation loop.

Figure A.14 shows a collection of snapshots from one of a walking experiment conducted with the Lab-On-Shoe 1.3 system. The camera mounted on the left foot captured images of the right foot. It is worth mentioning that the checkerboard was covered completely within the FOV after frame 4 in figure A.14.

Object Detection A 6×9 grid checkerboard, shown in figure A.15, was used as an object for detection. The checkerboard was symmetric, and each grid had a physical size of a 5×5 [mm] square, but other reference geometries or features, such as ApriTags, could also be used for detection. The object detection first identified the intersection points on the checkerboard and then estimated the position and orientation information of the checkerboard.

To identify the points on the checkerboard in each captured image, the checkerboard detection technique described in [64] was used. In such a method, the Harris corner detection is first applied to identify the pixel locations of the corners. Next, an edge detection technique

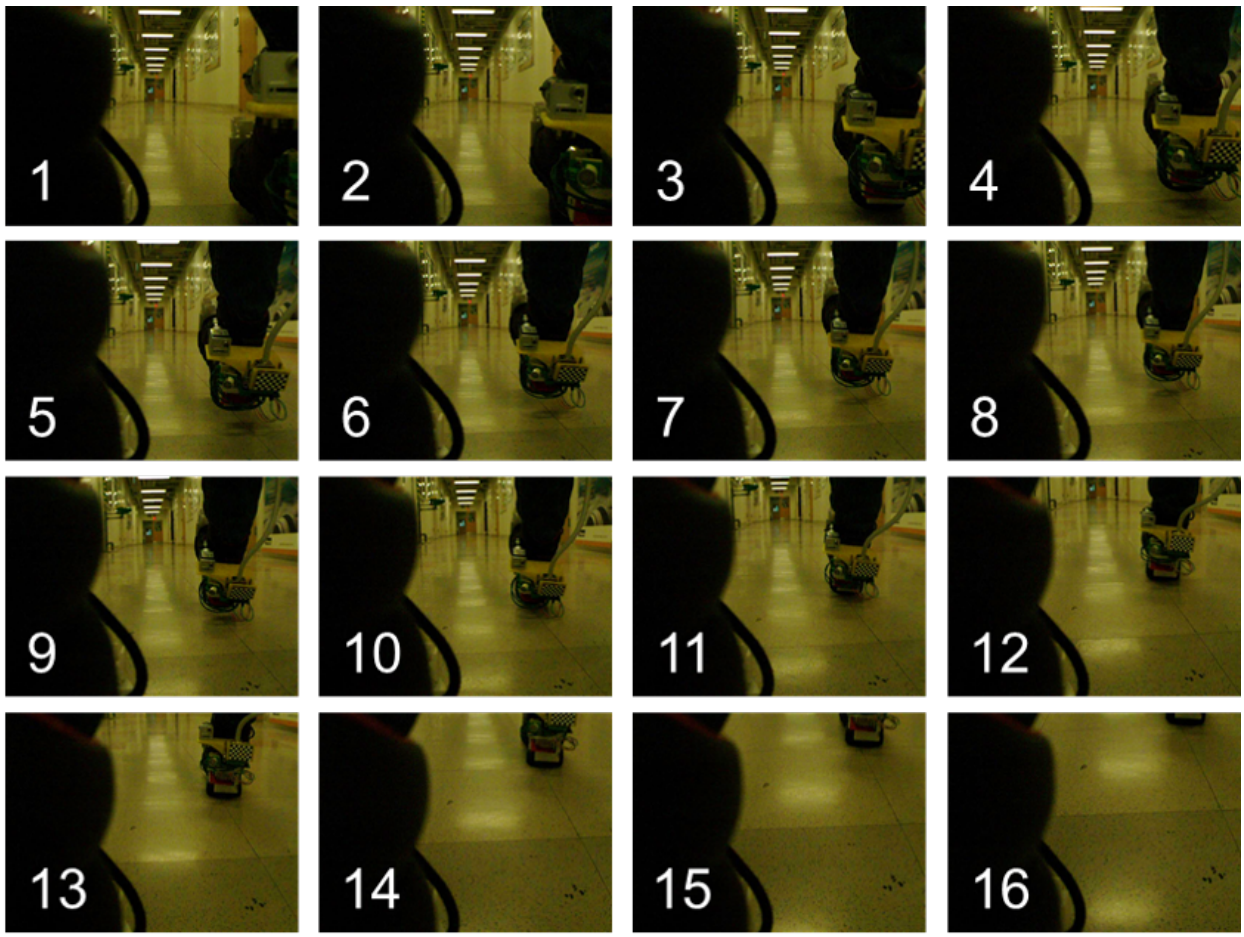


Figure A.14: An example of the video collected during a walking experiment. The number in each picture represents the frame number.

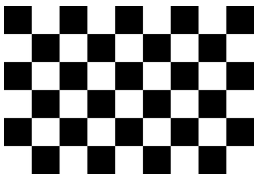


Figure A.15: A 6×9 checkerboard for object detection.

is applied to the original image to refine the corners identified in the corner detection step. Likelihoods of the corners are calculated in the Harris corner detection and the edge detection steps. Locations of the corners on the checkerboard are identified by selecting corners with likelihoods higher than a pre-defined threshold.

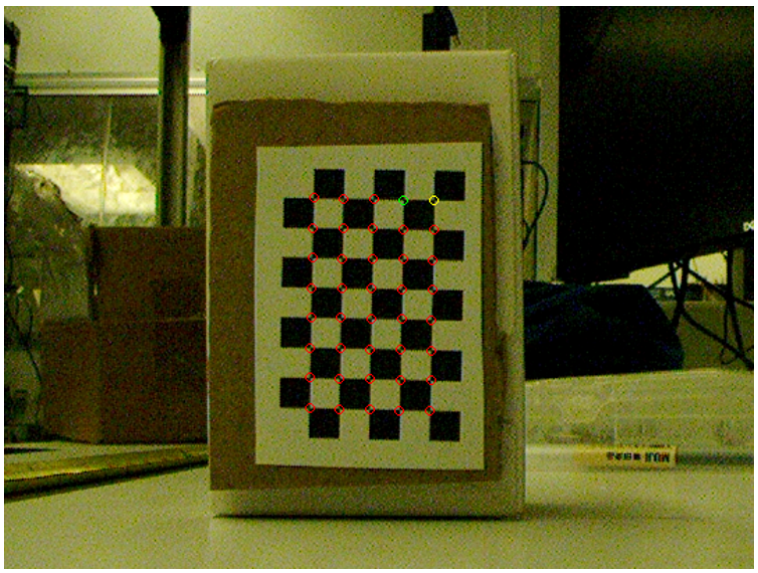


Figure A.16: An example of detected intersecting points on the checkerboard. The first and second points are marked with a yellow and a green circle, respectively, and the rest with red circles.

Figure A.16 shows an example of the detected intersecting points on the checkerboard. For this checkerboard, 40 points were identified. The first and second points are marked with a yellow and a green circle, respectively, and the rest with red circles. The centers of the circles are the pixel locations of the identified corners.

Object Geometric Information Extraction Camera extrinsic parameters were extracted from the corner locations on the pixel level identified in the object detection step. The

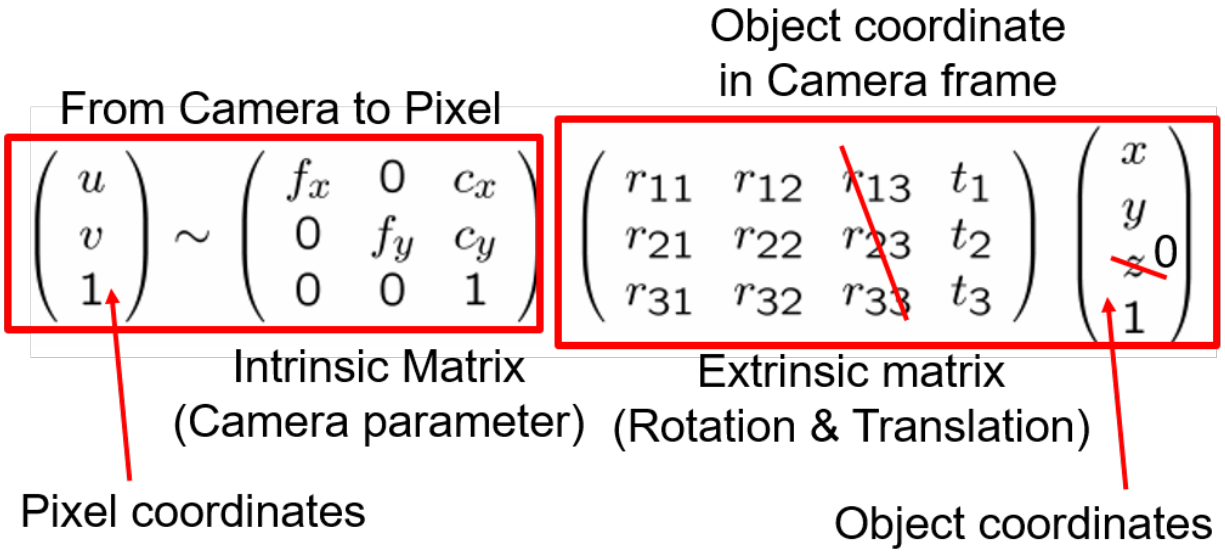


Figure A.17: Camera extrinsic parameter extraction algorithm. The variables u, v indicate the pixel located at u_{th} row and v_{th} column. f_x and f_y denotes parameters used to compensate for the radial distortion. c_x and c_y are the parameters used to compensate for center point offset. r_{ij}'s and t_i's in the extrinsic matrix are the parameters to determine the object's rotation and translation in the camera frame. The variables x, y, z denote the coordinates in the real world. Note that the extrinsic matrix consists of a rotation matrix and a translation vector.

extraction method is described in [259]. In this method, the pixel and real-world corner locations are related by the relationship shown in A.17. Since the lens of the camera integrated on the Lab-On-Shoe 1.3 platform had low distortion, the effect of center point offset and radial distortion were omitted, and the intrinsic matrix was set to a 3×3 identity matrix. It was assumed that the checkerboard lies on a perfect plane in the real world, so the z-coordinates of the corners on the checkerboard in the real-world frame were set to zero. As a result, the extrinsic matrix is reduced to a 3×3 matrix. With this assumption, the corner coordinates in the pixel and in the real world are known. All we need to do is to find the nine unknowns in the extrinsic matrix, which could be estimated through optimization techniques. Note that the outcome of the multiplication of the extrinsic matrix and the corner coordinates in the real world are the corner coordinates in the camera frame. Since a rotation matrix is an orthogonal matrix, the rotation matrix could be reconstructed from the 3×3 extrinsic matrix.

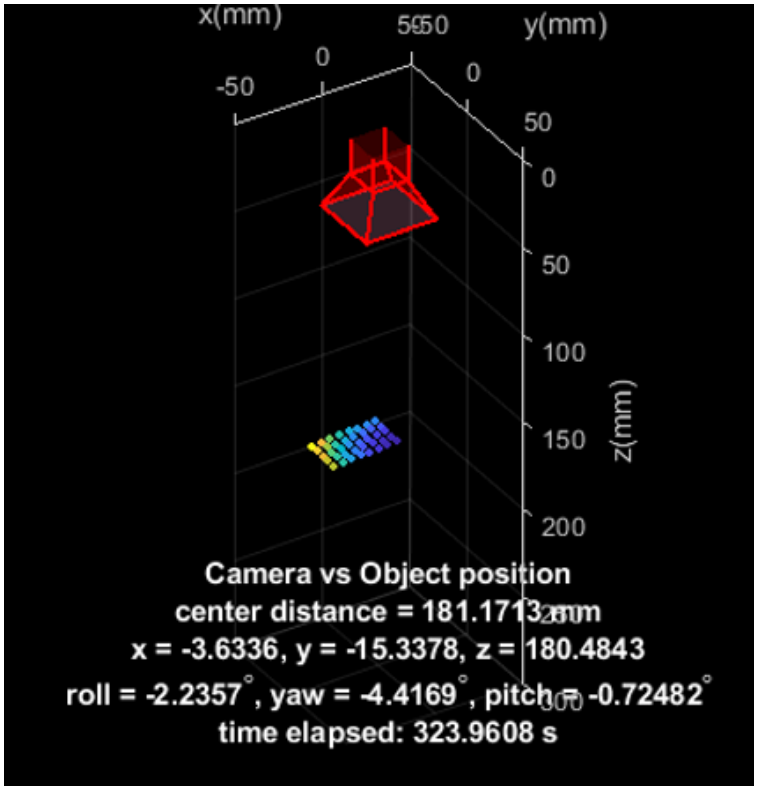


Figure A.18: Estimated position and orientation of a checkerboard.

A MATLAB program was developed to estimate the position and orientation of the checkerboard with a monocular camera. Figure A.18 shows a screenshot of the MATLAB program estimating the location of the checkerboard in figure A.16. The red object in the figure indicates the location of the camera, which is set at the origin, and the colorful points marked the estimated locations of the corners on the checkerboard. The errors in such estimation are contributed by several factors, including optimization errors, imprecise measurement of the physical size of the grid on a checkerboard, the flatness of the checkerboard, and electronic noise in the optical sensors.

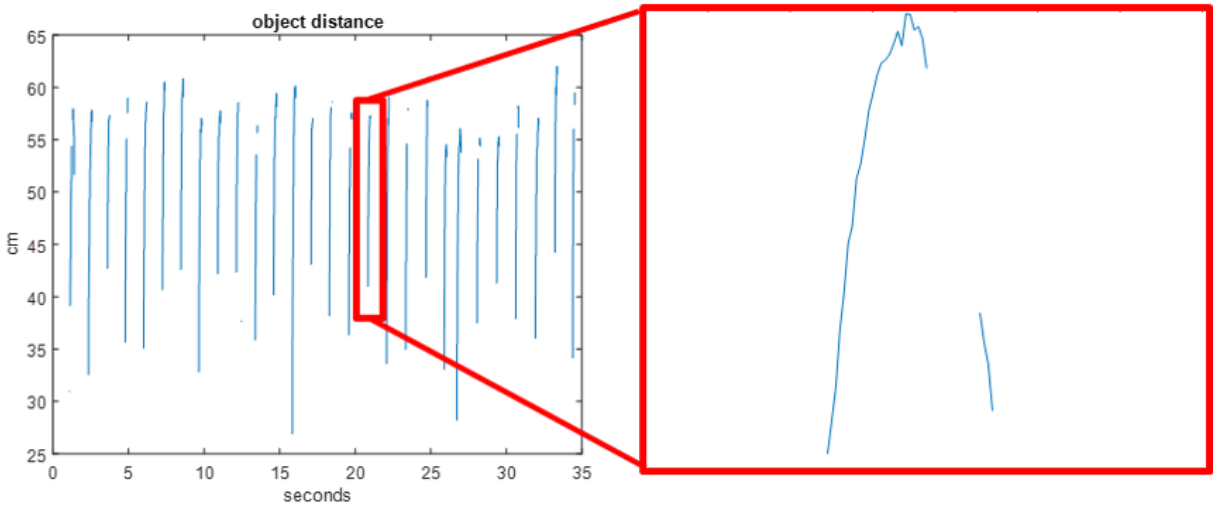


Figure A.19: Profile of the distance between two shoes captured by the camera.

Camera foot-to-foot Ranging Measurements An indoor experiment was conducted to investigate the accuracy of the poistion and orientation extracted based on images captured by the foot-mounted cameras. The experiment involved a subject equipped with the Lab-On-Shoe 1.3 platform walking straight along the north for 53 meters. The true distance between the two shoes of the Lab-On-Shoe 1.3 platform at the end of the experiment was 0.15 [m]. The developed camera ranging approach was used to measure the distances between two shoes when the checkerboard was fully presented in the FOV of the camera. The readouts of the camera ranging, shown in figure A.19, showed that at some of the time instances, there was no data point. The missed data points were due to either the checkerboard was not

completely covered in the camera FOV or the corresponding image being so blurry that the corners were failed to be identified.

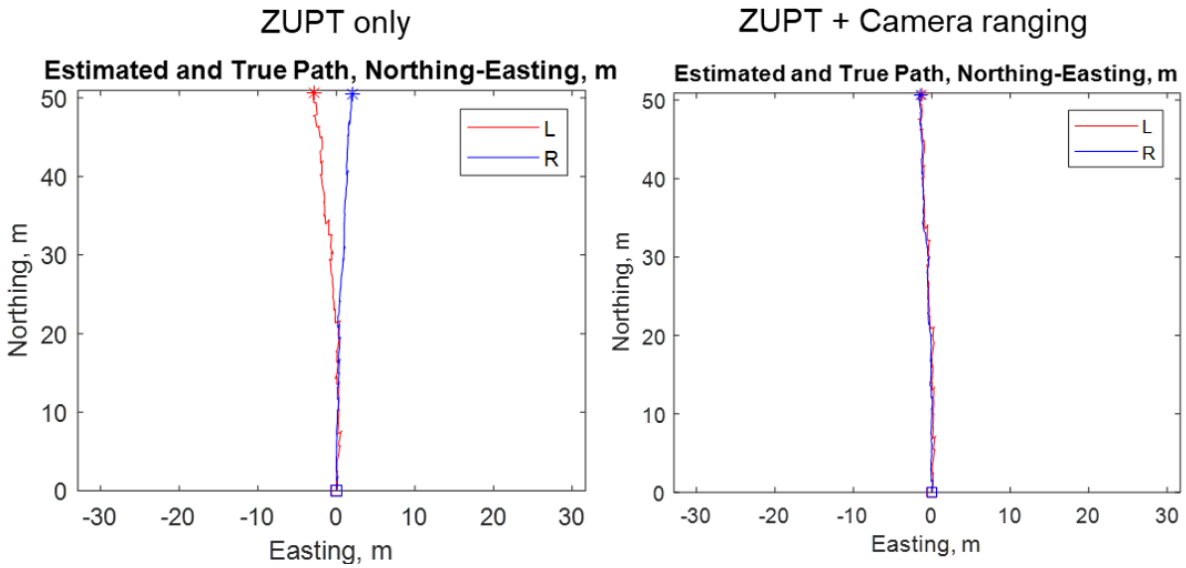


Figure A.20: Navigation results of using ZUPT and ZUPT + ranging. The distance between the two shoes at the end of the experiment using only the ZUPT algorithm is 4.88 [m] while that using ZUPT and camera ranging is 0.186 [m].

The ranging measurements were used to augment the ZUPT-augmented INS in the EKF framework. Figure A.20 compares the results of using a standalone ZUPT-aided INS (ZUPT only) and ZUPT-aided INS augmented with the ranging measurements (ZUPT + Camera). The red curve and the blue curve in Figure A.20 represent the trajectories of the left foot and right foot, respectively. Note that in the case of the ZUPT + Camera ranging algorithm, two trajectories almost overlap each other. In this experiment, using camera ranging as an additional measurement improved the estimated distance between two shoes at the end of the experiment from 4.88 [m] to 0.186 [m].

In the experiment reported in figure A.20, it was observed that the exposure time setting of the camera was a key factor when using the camera as a ranging sensor. On the one hand, a long exposure time would lead to a quite blurry image since the foot velocity could go up to as fast as 3 [m/s]. Figure A.21 compares two frames taken using the exposure time of 10000 [μs] and 2000 [μs], respectively. The checkerboard detection failed in the 10000 [μs]

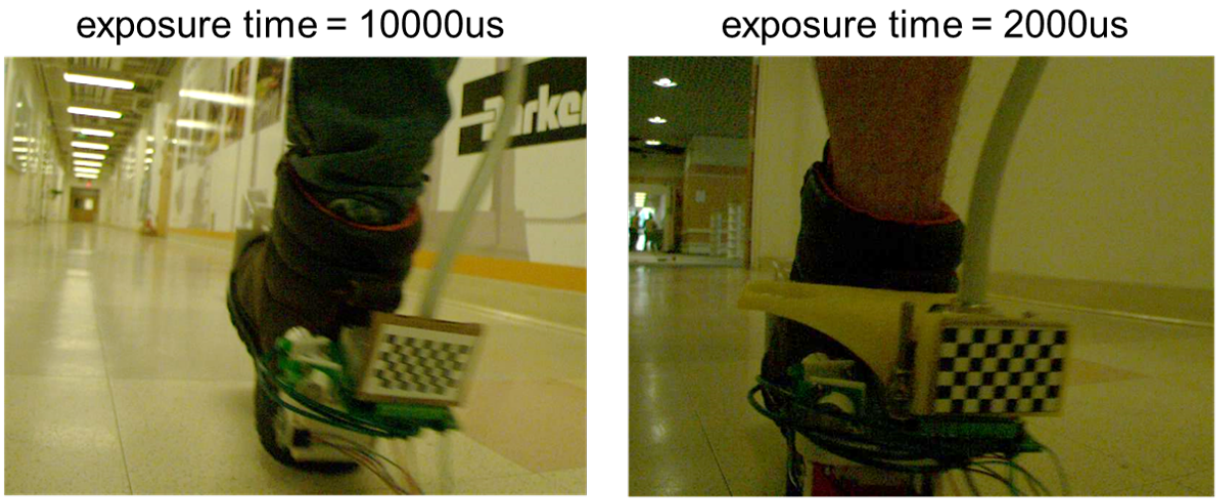


Figure A.21: Effect of exposure time of the camera in the walking experiment.

case and worked well in the 2000 [μs] case. On the other hand, if the exposure time is set too short, the sensor would not capture enough light, and the resulting images do not have clear corner features.

Controlling and Monitoring via an Android Mobile App An Android mobile App, Sensor GraphX, for the Lab-On-Shoe 1.3 platform was developed by UCI undergraduate student Anthony Skoury under the supervision of the author of this thesis Chi-Shih Jao. The App displays the location of the platform in real-time and sends commands to reset its initial location in event of a request for new navigation task.

The data transmission mechanism between the Lab-On-Shoe 1.3 platform and the Sensor GraphX App is shown in figure A.22. On Lab-On-Shoe 1.3 platform, the CompactRIO calculates the current location of its user, and the location data are saved as Shared Variables in LabVIEW. These Shared Variables are published to a customized website with a specified address in a local network. On the Sensor GraphX App, the App downloads data from the website and displays it on the mobile phone. In the case of sending a command from the phone to the platform, a reversed counterpart of the data transmission mechanism is sued.

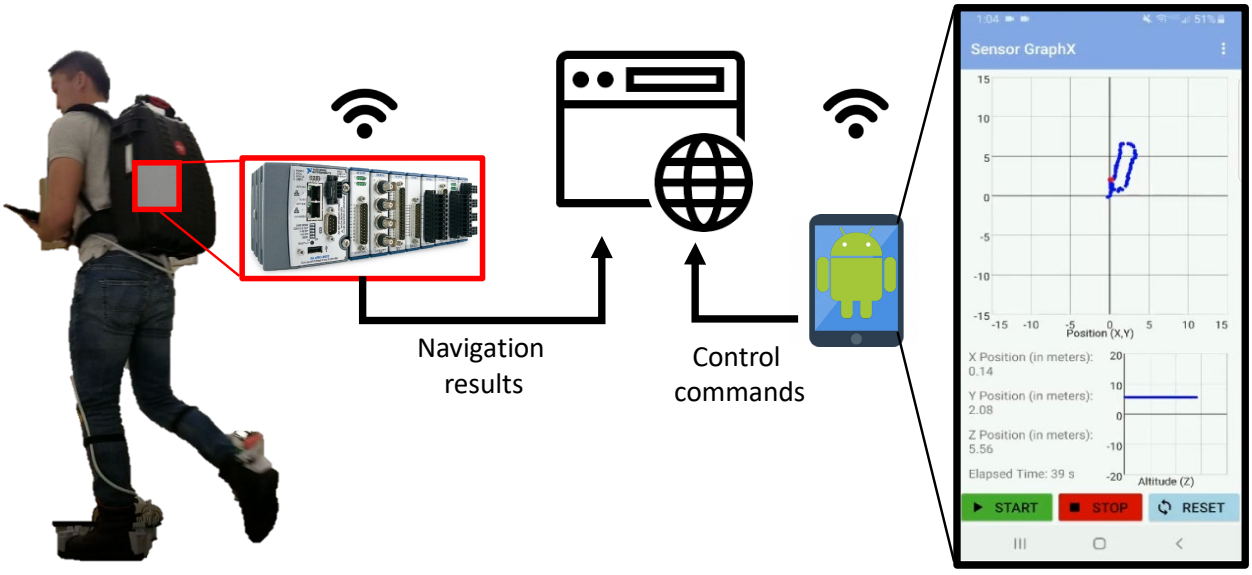


Figure A.22: The Android mobile App, Sensor GraphX, for real-time data visualization and sending commands.

The phone updates the value of the Shared Variables for control, and the platform reads the Shared Variables into the system.

Lab-On-Shoe 1.4: Integrating With Mass Flow Sensors

The Lab-On-Shoe 1.4 platform was upgraded based on the Lab-On-Shoe 1.3 platform with an additional sensing modality of mass flow sensors. Table A.4 lists COTS components integrated on the Lab-On-Shoe 1.4 platform. A mass flow sensor measures the flow rate using the thermo-transfer principle. Its output data is flow rate [liter/min], which is proportional to the velocity along the axis parallel to its tube. This section first discusses the integration of a mass flow sensor with the Lab-on-Shoe 1.4 platform and then analyzes the performance of the mass flow sensor in assisting the ZUPT algorithm.

Sensor Selection Mass flow sensor Renesas Electronics FS1012 was selected. The sensor is shown in figure A.23. This sensor measures the flow rate using the thermo-transfer principle.

Table A.4: COTS components used in Lab-On-Shoe 1.4 platform.

Component	Manufacturer	Model	Quantity	Purpose
FPGA	National Instruments	CompactRIO−9039	1	Processing Unit
Microcontroller	Arduino	Uno	1	Processing Unit
IMU	Analog Device	ADIS16497−3	2	Acceleration and angular rate
SONAR	Devantech	SRF08	4	Ranging
Altimeter	TE Connectivity	MS5803−01BA	2	Height
Camera	Basler	acA800−200gc	2	Image
Mass Flow Sensor	Renesas Electronics	FS1012	2	Velocity
Battery	Powerizer	LiFePO4	1	Power source
Voltage Regulator	Texas Instruments	LM3100EVAL	2	Power management
Voltage Regulator	Texas Instruments	TPS7A1601EVM−046	2	Power management

The output data is velocity along the axis parallel to the tube. It uses I2C as the communication protocol and requires a supply voltage 5 [V]. The sensor's measurements are available in both analog and digital formats. The Lab-On-Shoe 1.4 platform uses digital signals.

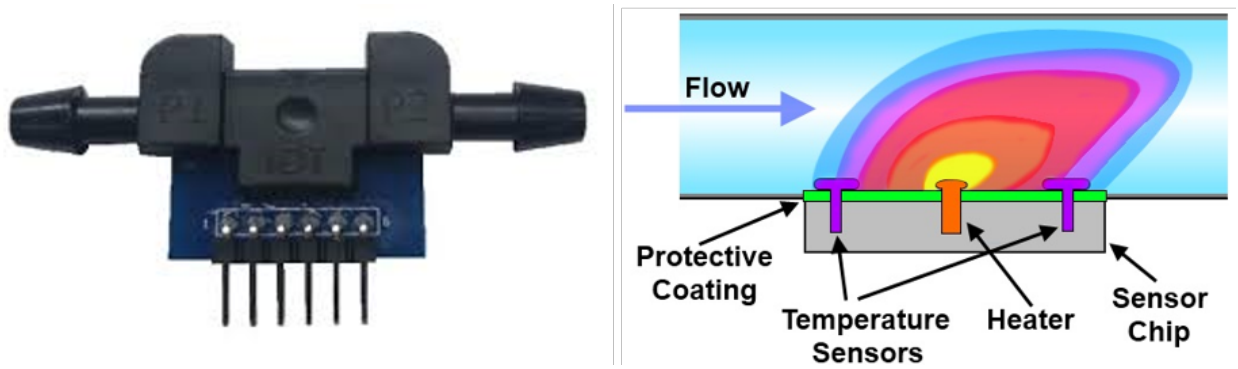


Figure A.23: FS1012 mass flow sensor and its operation principle [227]. The physics behind this sensing element is that changes in air/liquid/gas flow disturb the temperature gradient.

Integration of a Mass Flow Sensor with Lab-On-Shoe The selected mass flow sensor was connected to the I2C pins on an Arduino Uno board. The Arduino Uno Board was connected to the CompactRIO processor, and the measurements acquired from the mass flow sensor were transmitted from the Arduino board to the Compact-Rio. The sampling rate was set to 15 [Hz].

A customized 3D fixture for housing the mass flow sensor was designed and 3D-printed by UCI undergraduate student Manuel Aaron under the supervision of the author of this thesis, Chi-Shih Jao. The fixture was mounted on the heel side of the shoe, under an IMU. The configuration is shown in Figure A.24. Air flow data measured the sensor was collected by the Arduino board placed at the toe side of the same shoe. This configuration allows the mass flow sensor to detect the air flow rate along the direction parallel to the heading of the shoe.

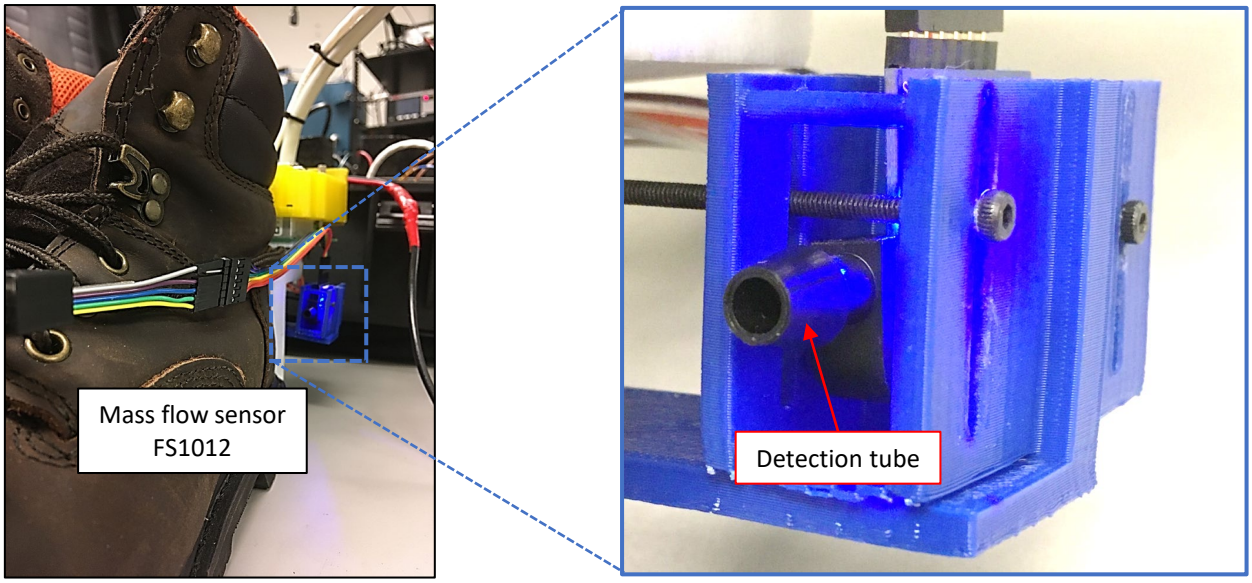


Figure A.24: Mass flow sensor integrated on the heel side of the Lab-On-Shoe 1.4 platform.

Performance of the Mass Flow Sensor When Used as a Velocity Estimator The mass flow sensor readouts are proportional to the velocity along the axis parallel to its tube. To test velocities estimated by the sensor, a series of experiments of walking straight toward the north for 53 [m] in 70 [s] were conducted. Here, only the data collected from one of the experiments is presented, but similar effects were observed in other runs of the same experiments. The readouts of the mass flow sensor mounted on the left foot are shown in Figure A.25. The velocity profiles along the north direction estimated from the left IMU measurements are presented in Figure A.26. Figure A.27 compares a normalized version of

the readouts of the mass flow sensor shown in Figure A.25 and a normalized version of the velocity profiles shown in Figure A.26. Comparing the two datasets, three remarks could be made. First, it could be observed that each peak in Figure A.25 corresponded to a peak velocity in a step in Figure A.26. Second, it could be observed that the mass flow sensor failed to detect some of the steps. Third, even for the steps the sensor successfully detected, their peak values were inconsistent. Therefore, it was concluded that using the mass flow sensor as a velocity-aiding approach would not improve navigation solutions.

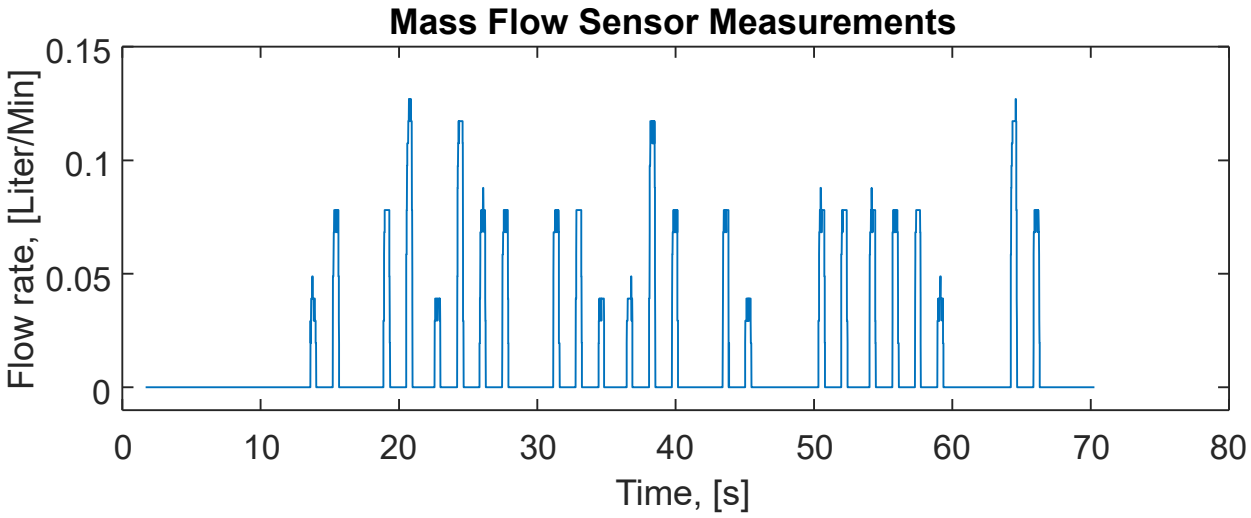


Figure A.25: The readouts from the mass flow sensor.

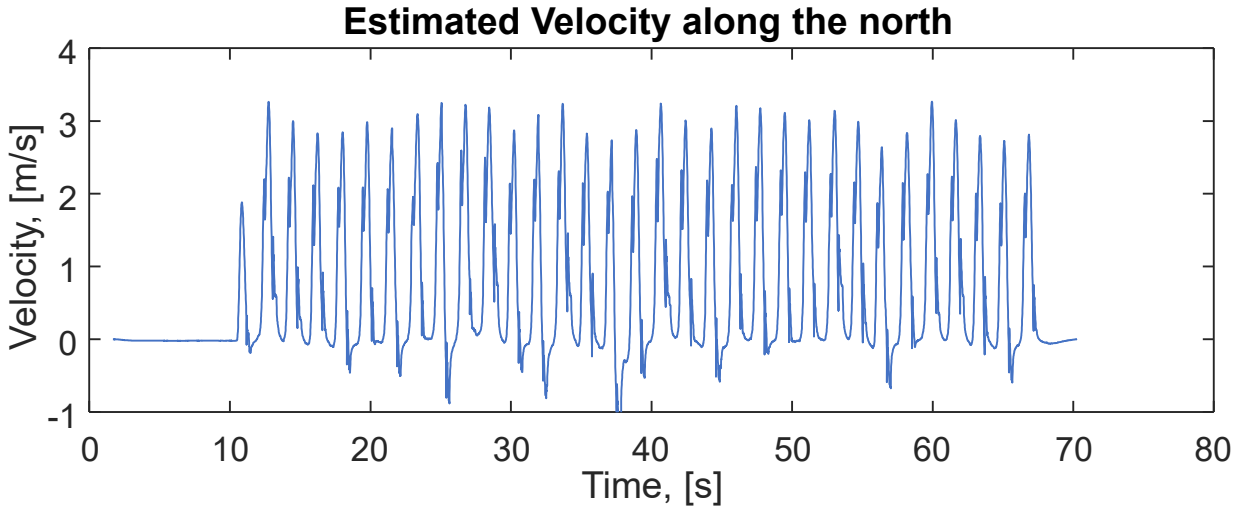


Figure A.26: The velocity along the north estimated by a ZUPT-aided INS algorithm.

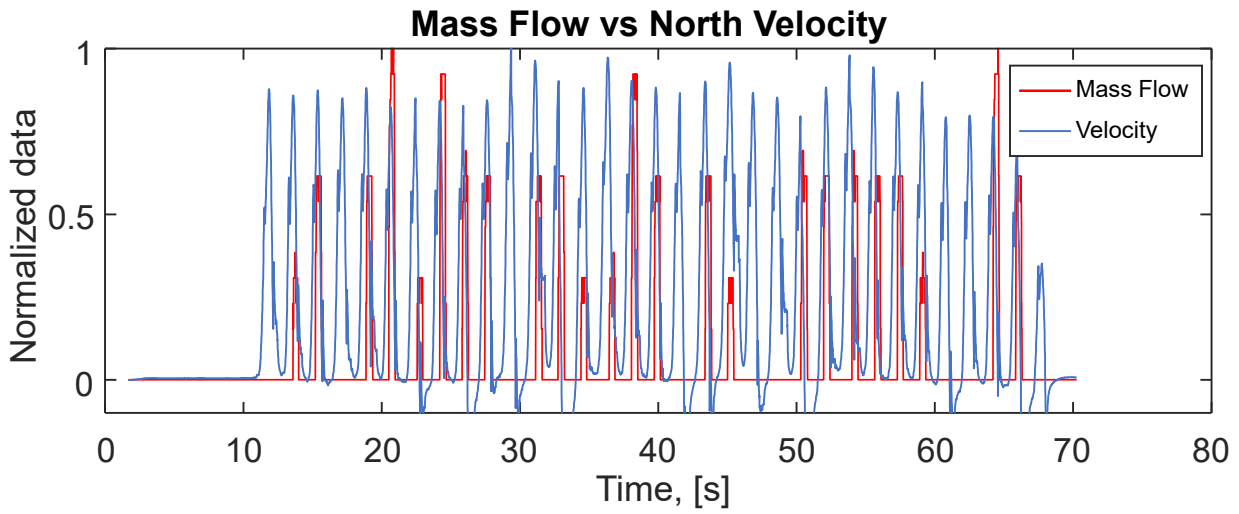


Figure A.27: The joint plot of the normalized version of the readouts from the mass flow sensor and estimation of velocity along the north based on IMU readouts.

stance phase detector Aided by Mass Flow Sensor This section explores the usage of a mass flow sensor as an aiding technique to improve stance phase detection used in a ZUPT-aided INS algorithm. In FigureA.27, an interesting phenomenon could be noticed: the mass flow sensor had measurements only when the shoe was moving. his phenomenon could be used to reduce the mis-detection rate of a stance phase detector. In a conventional IMU-based stance phase detector, a pre-defined fixed threshold was used to determine the stationary status of the shoe. The conventional stance phase detector can be described as follows:

$$\text{ZUPT status} = \text{H}(\frac{\sigma_a}{\bar{\sigma}_a} + \frac{\sigma_g}{\bar{\sigma}_g} - \gamma), \tag{A.1}$$

where σ_g and σ_a are the summation of variances of gyroscope readout and that of accelerometer readout, H() is a Heaviside function, $\bar{\sigma}_g$ and $\bar{\sigma}_a$ are normalized amplitudes of VRW and ARW. In the optimal implementation of the conventional ZUPT algorithm, different thresholds should be applied to different walking patterns. Mis-detection could happen when the ZUPT threshold is set to a value optimal for a walking speed higher than the actual walking speed. An example of such a mis-detection event is shown in Figure A.28. In terms of accumulated navigation errors, the mis-detection can lead to an estimated trajectory shorter

than the nominal one.

Based on the observation of the mis-detection, the stance phase detector expressed in (A.1) was modified by combining IMU measurements and the normalized mass flow sensor readouts with an logic OR gate. The mathematical expression of the modified stance phase detector is described as follows.

$$\text{ZUPT status} = \text{H}(\frac{\sigma_a}{\bar{\sigma}_a} + \frac{\sigma_g}{\bar{\sigma}_g} - \gamma) + \text{H}(s_i - \alpha) - \text{H}(\frac{\sigma_a}{\bar{\sigma}_a} + \frac{\sigma_g}{\bar{\sigma}_g} - \gamma) \times \text{H}(s_i - \alpha),$$

where s_i is a mass flow readout at sample i and α is the minimum resolution of the sensor. The modified stance phase detector is expected to reduce the mis-detection rate, as compared to as conventional IMU-based stance phase detector.

A experiment was conducted to simulate a situation where the actual walking speed during a pedestrian navigation task is lower than the nominal speed. This was done by adopting a ZUPT threshold e^8 that was higher than the threshold corresponding to normal walking speed (usually e^6). Accumulated navigation errors of ZUPT-aided INS that used the conventional stance phase detector were compared to the errors in the case of using the modified stance phase detector. The navigation results are shown in Figure A.29. The red curves in both plots are the estimated trajectories of the left shoe. The nominal final position of the left foot was $[0, 53]$ [m]. With the conventional stance phase detector, the estimated final position of the left foot trajectory was $(0.084191, 45.0314)$ $[m]$. With the stance phase detector augmented by the mass flow sensor, the estimated final position of the left foot trajectory was $[-0.07563, 48.4417]$ [m], showing an improvement of more than 40% in the accumulated distance error.

This section explored the possibility of integrating mass flow sensors with the Lab-on-Shoe. Based on the sensor readouts obtained in the walking experiments, it was concluded that it would be unsuitable to use mass flow sensors as a velocity estimator in pedestrian navigation.

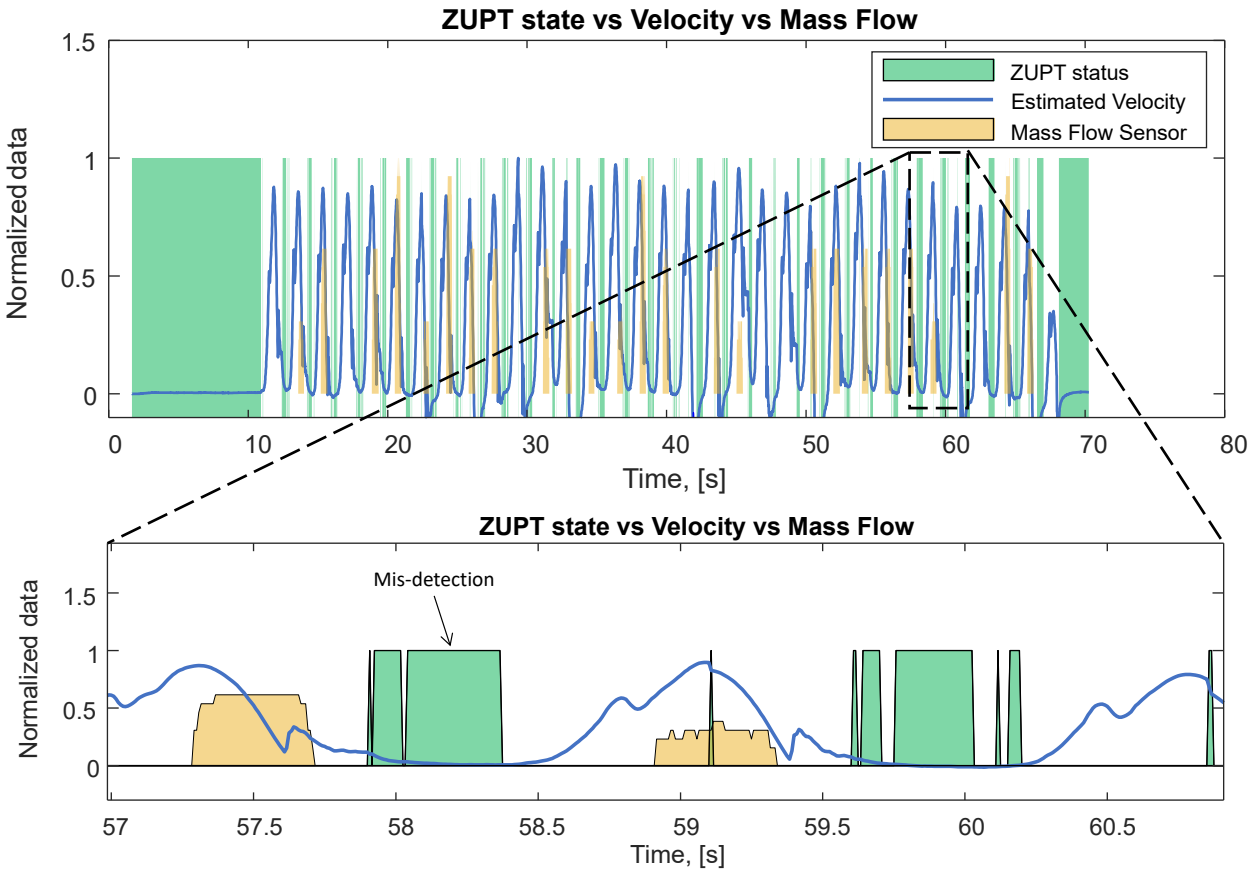


Figure A.28: The upper plot is a joint plot of ZUPT measurements, the normalized versions of the estimation of velocity along the north, and the mass flow readouts. The lower plot shows a zoomed-in view of the region where a mis-detection had happened. The blue area in the plot indicates that, during the period, the stance phase detector determined a stance phase. In the optimal implementation of the ZUPT algorithm, the blue area and the yellow area should not overlap. Since the value of the ZUPT threshold was deliberately set to too high in our experiment, we observed overlaps between the yellow area and the blue area, which were the mis-detection.

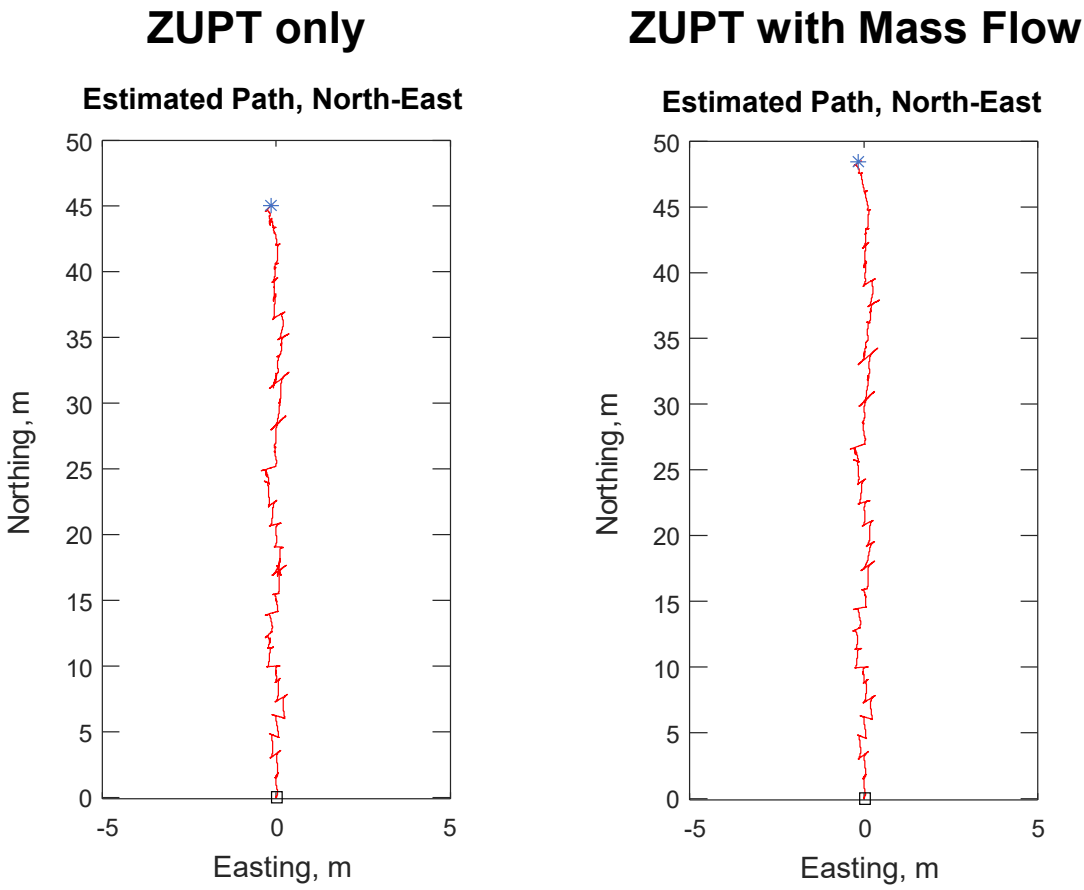


Figure A.29: The left plot shows the navigation results using the conventional stance phase detector. The right plot shows navigation results using the modified stance phase detector.

But the sensor could be used to assist the stance phase detection in the ZUPT algorithm in terms of reducing the the detector's mis-detection rate. The accumulated navigation errors using the modified stance phase detector showed an improvement of more than 40%, compared to the conventional stance phase detector.

Lab-On-Shoe 1.5: Integrating With Dynamic Vision Sensor

In this section, we explored the performance of a DVS, also known as, an Event Camera, and its communication protocol. The sensor is a new generation camera, with advantages of 1) fast sampling rate, 2) efficient data format, and 3) no issue with motion blur. Because of these advantages, it is potentially beneficial for implementing real-time vision-aided Inertial

Navigation Systems. The DVS sensor was integrated on the Lab-On-Shoe platform, and the integrated system is referred to as the Lab-On-Shoe 1.5 platform. Table A.5 lists COTS components integrated on the the Lab-On-Shoe 1.5 platform.

Table A.5: COTS components used in Lab-On-Shoe 1.5 platform.

Component	**Manufacturer**	**Model**	**Quantity**	**Purpose**
FPGA	National Instruments	CompactRIO−9039	1	Processing Unit
Microcontroller	Arduino	Uno	1	Processing Unit
IMU	Analog Device	ADIS16497−3	2	Acceleration and angular rate
SONAR	Devantech	SRF08	4	Ranging
Altimeter	TE Connectivity	MS5803−01BA	2	Height
Camera	Basler	acA800−200gc	2	Image
Mass Flow Sensor	Renesas Electronics	FS1012	2	Velocity
Dynamic Vision Sensor	IniLabs	DVS128	1	Light Intensity Change
Battery	Powerizer	LiFePO4	1	Power source
Voltage Regulator	Texas Instruments	LM3100EVAL	2	Power management
Voltage Regulator	Texas Instruments	TPS7A1601EVM−046	2	Power management

Background of DVS A Dynamic Vision Sensor (DVS), also known as an event camera or event-based camera, was designed based on the operation mechanism of human retina and is a new generation of cameras that detect changes in light intensity on each pixel asynchronously. The first commercial product was introduced in 2008. Due to its asynchronous detection mechanism, it is capable of achieving an equivalent frame per second of 10 [kHz] and minimizing the motion blur issue encountered in traditional CMOS cameras. Moreover, the output data is more concise than conventional cameras because it only gives a measurement of polarity (polarity +1 means light intensity increases, and −1 means the intensity decreases) when a change of light intensity is detected by a pixel. Data produced by event cameras is called an ”event.”

The DVS128 and its Data Structure ".aedat" The event camera DVS128 is one of the first-generation event-based cameras. It has a resolution of 128×128, with a Field Of View of 60^o. There is open-source firmware for this sensor. jAER is the most popular one. With jAER, not only can we visualize and log data but also apply real-time filters, such as noise compensation or edge detection. The output of the logged files is saved in the standard ".aedat" format.

Each event contains four pillars of information: 1) polarity (p), 2) pixel location in the x-axis (x), 3) pixel location in the y-axis (y), and 4) timestamps of the local processor on the DVS 128 (t). The unit of the timestamps is μs. This information is embedded in binary format with 6 bytes. Among the 6 bytes, polarity takes 1 bit, each of the pixel locations takes 7 bits, and the timestamp takes 32 bits. The four pieces of information are embedded with the following configuration:

1yyyyyyy.pxxxxxxx.tttttttt.tttttttt.tttttttt.tttttttt.

The UART Communication Protocol The DVS128 can also be accessed with the UART communication protocol. A USB-to-UART adapter module is used to achieve UART communication from the USB port. The serial communication should be set up as follows: 1) 12 [MHz] for baud rate and 2) RTS/CTS for hardware control. Each UART command should be followed by the new feed string ($0xA$).

The data can be configured in two formats. The first format is in binary form, introduced in the last section. The second format is 21-bit data represented in ASCII code string. The information p, x, y, and t take 1, 3, 3, and 6 characters in the string, respectively. Between the four pieces of information, it is filled with the space string, and the new feed string follows the end of the entire string.

Custom Firmware for CompactRIO Platform The DVS128 was integrated into the Lab-On-Shoe 1.5 platform. A custom firmware was developed in LabVIEW for the camera. The built-in serial communication package called NI-VISA was used to send commands for multiple different serial communication protocols, such as I2C, SPI, and UART. In the case of the DVS128, UART is used, and the corresponding commands provided in the DVS128 User Manual by IniLabs, are listed in Table A.6. Basic event acquisition requires only four commands (presented in ASCII): 1) reset (R), 2) data format setup (!E6), 3) acquisition starts, and 4) acquisition stops.

Command name	function
E+/-	enable/disable event sending
!Ex	specify event data format, ??E to show options (see below for more details)
!ET=x	set current timestamp to x (default: 0)
!ETM[0,+]	synch timestamp, master mode, 0: output stopped; +: output active
!ETS	synch timestamp, slave mode
!ETI	single retina, no external synch mode
!B[0–11]=x	set bias register to value
!BF	send bias settings to DVS (flush)
?Bx	get bias register x current value
0,1,2	LED off/on/blinking
R	reset board
??	display help menu
??E	display Event data menu

Table A.6: UART commands for DVS128.

A custom user interface (UI) was developed in LabVIEW for acquisition and visualization of events produced by the DVS128, connected to the NI-CompactRIO. Figure A.30 shows a snapshot of the UI under operation during a rotating doted plate experiment. The smaller plot on the right-hand side showed one frame of the events with a frame rate of 60 [Hz]. The 3D plot on the left-hand side presents the same set of events in 3 dimensions. The third dimension is time. At the bottom of the 3D plot is a projection of the frame of events onto the x-y plane. The projected image is identical to the 2D plot on the right-hand side. The

inner circle observed in the frame was the trail of the moving dot, and the outer circle was caused by the paper plate not being a perfect circle. Figure A.31 displays an image taken by a conventional camera that included all the objects within the FOV of the DVS128. The acquisition was executed by a sequential read of 21-byte data at a rate of 1 [MHz]. For data visualization, we designed two modes, frame and scatter. In the frame mode, a period is selected, and all the data collected during the period are displayed. The amount of data being shown in one frame is dynamic in this mode. In the scatter mode, a fixed number of events are collected and then displayed so that the frame rate is dynamic. Note that the choice of modes does not affect data acquisition or data logging.

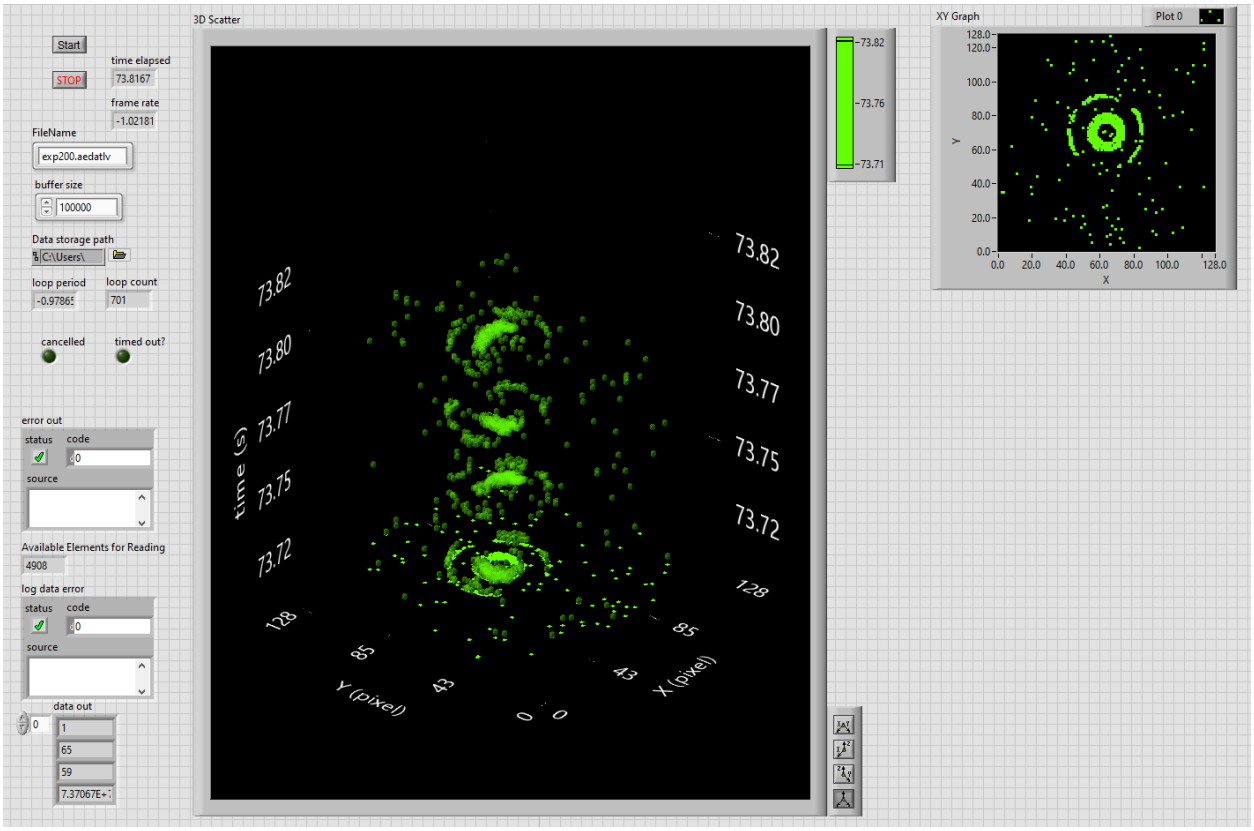


Figure A.30: Customized LabVIEW user interface for the DVS128.

Event Visualization To understand the properties of the data collected from DVS128, we first conducted an experiment, where the DVS128, mounted on one end of a small railway, recorded a cart moving on the railway away from the sensor. The experimental setup is

Figure A.31: The FOV of the DVS128 represented in a conventional camera in the rotated doted plate experiment.

shown in Figure A.32. The DVS128 was mounted on the red fixture, and the blue object was the cart moving away from the sensor. The entire experiment lasted around 6 seconds. The cart started to move after 2 seconds.

Figure A.33 shows the collected data. The sub-figure on the left-hand side is a 3D plot of the entire data set, where x and y axes are pixel locations in x and y respectively, and z is the time. Each event in the plot represents a change of light intensity at the corresponding pixel. The black arrow indicates the time instance when the car started to move. It is not hard to observe that when the cart started to move after 2 seconds, the number of events being generated increased dramatically. We could also observe that before 2 seconds, some events existed. These events are assumed to be the noise of the DVS128 sensor.

In Figure A.33, we could visualize the data by "slicing" a portion of the data set along the time axis. The three plots on the right hand side are the data sliced at $t = 0$ [s], $t = 2$ [s], and $t = 4$ [s]. We could see that at $t = 0$ [s], only noise data existed. At $t = 2$, the collected events depicted the shape of the cart. At $t = 4$ [s], the size of the shape became smaller due to that it was moving away from the sensor.

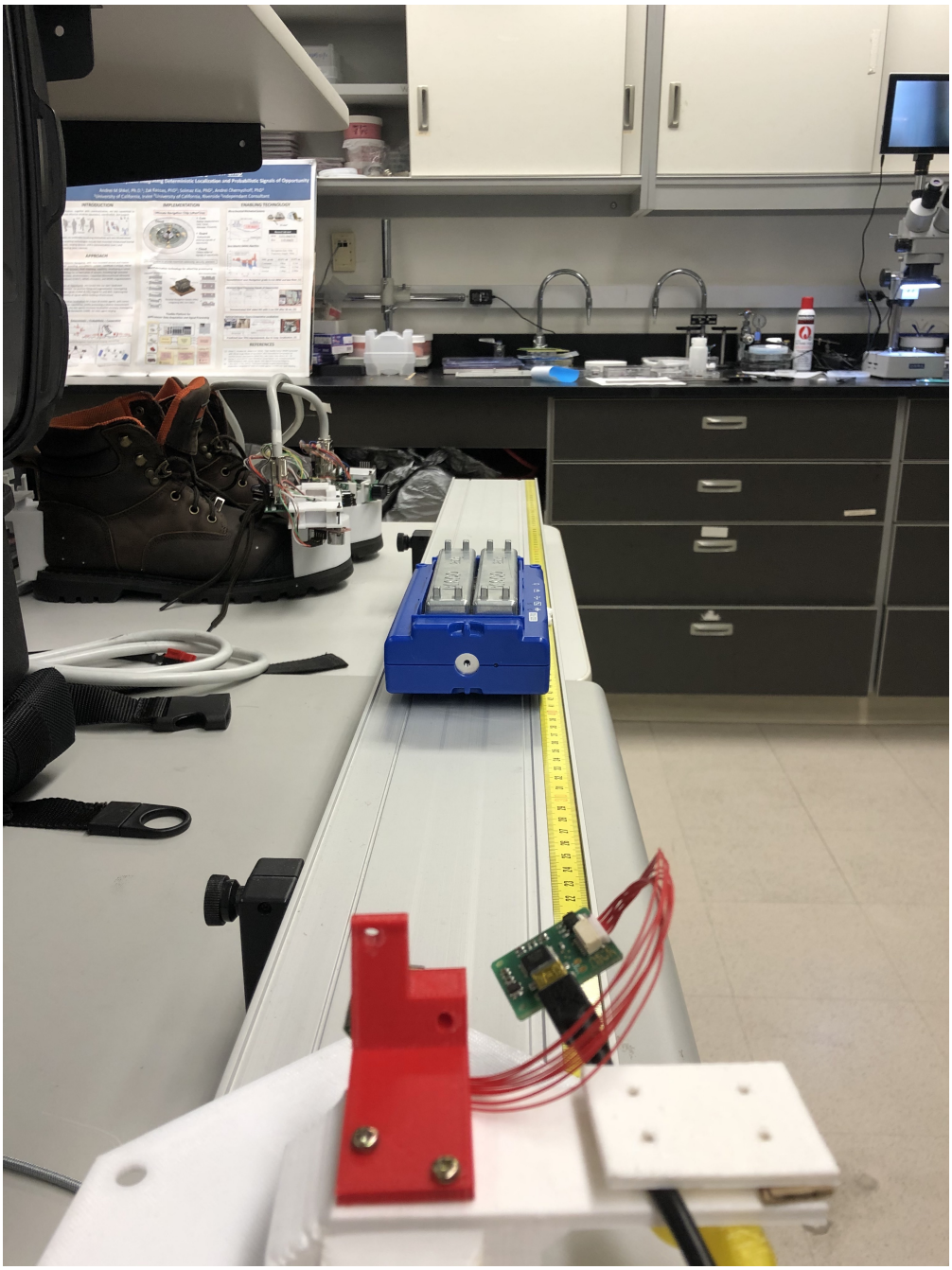


Figure A.32: Setup for the cart experiment.

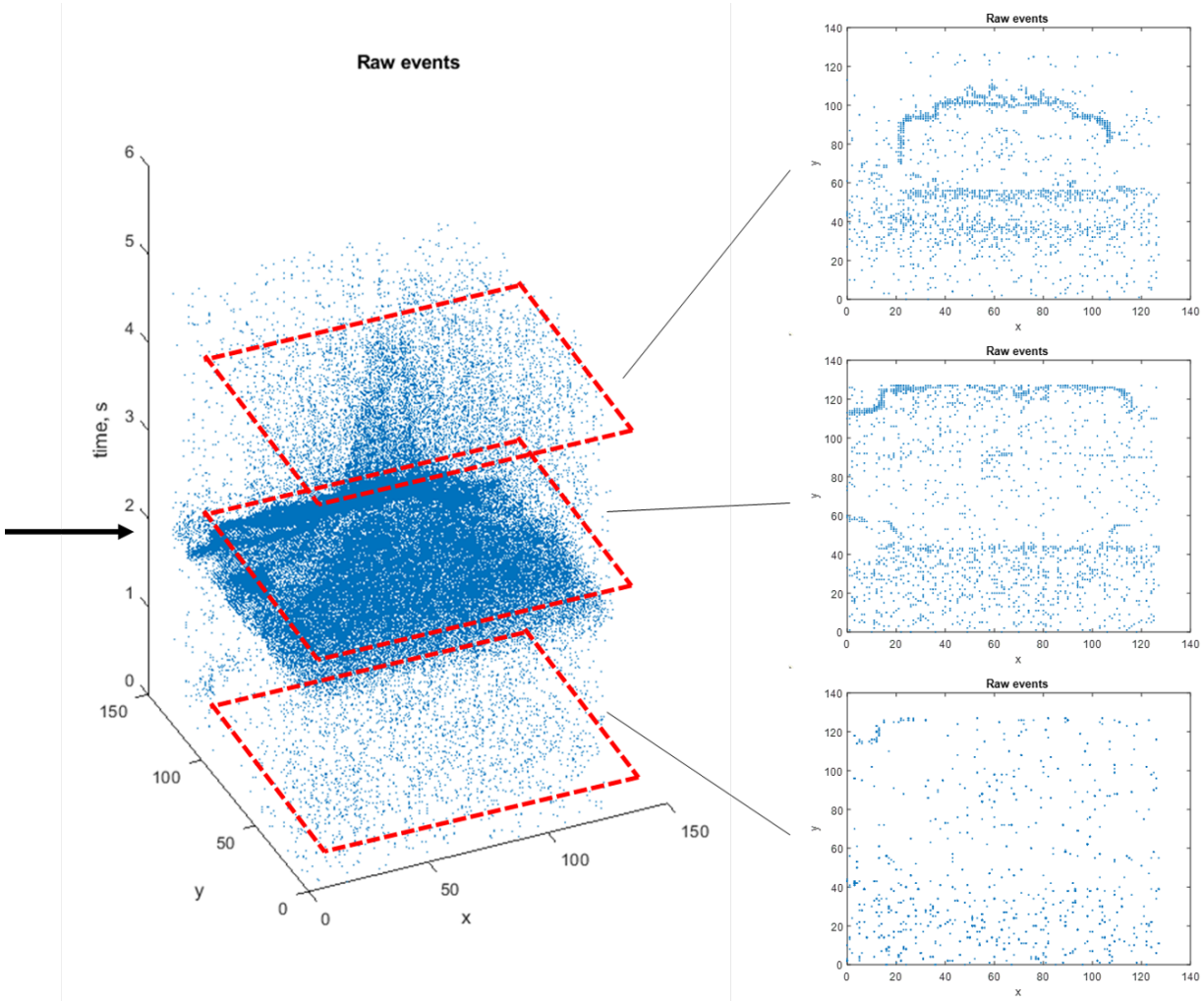


Figure A.33: DVS128 data from the cart experiment.

DVS128 Integrated With Lab-On-Shoe Platform One application of the DVS128 sensor is to improve the stance phase detector in the Zero Velocity Update algorithm. The improvement was envisioned because the generation of the events for event cameras is correlated with the relative velocity of the sensor to the surrounding environment. To investigate the validity of using the DVS128 to enhance traditional IMU-based stance phase detection, we first conducted experiments to understand the properties of a shoe-mounted event camera. Figure A.34 shows the integrated Lab-On-Shoe platform with the DVS128. The sensor was mounted on top of the IMU and faced forward.

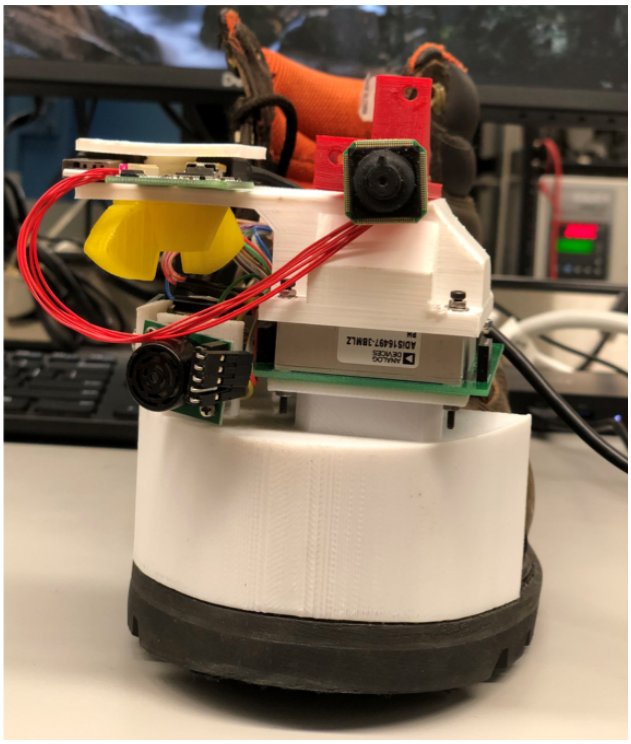


Figure A.34: Lab-On-Shoe 1.5 platform integrated with DVS128.

Figure A.35 presents the data collected with the integrated Lab-On-Shoe 1.5 platform in a walking experiment. In the experiment, the agent walked four steps on a flat plane in an indoor environment. The entire experiment lasted 16 seconds. In the first 6 seconds, the agent stood still. The agent walked four steps from 6 [s] to 12[s]. For the remaining time, the foot remained still. In Figure A.35, the periods with dense events corresponded to the swing phase of a gait cycle. The other periods mapped to the stance phase. A sliced data

corresponding to the swing phase showed that the events captured some parts of the internal structure of the surrounding. Notice that in this data, a group of events existed from the beginning to around 13 seconds. These events corresponded to the "EXIT" sign located in the surrounding. Because the "EXIT" sign was backlit with fluorescent light, which has a constant change of light intensity. Therefore, it also could generate events even if the event camera was not moving.

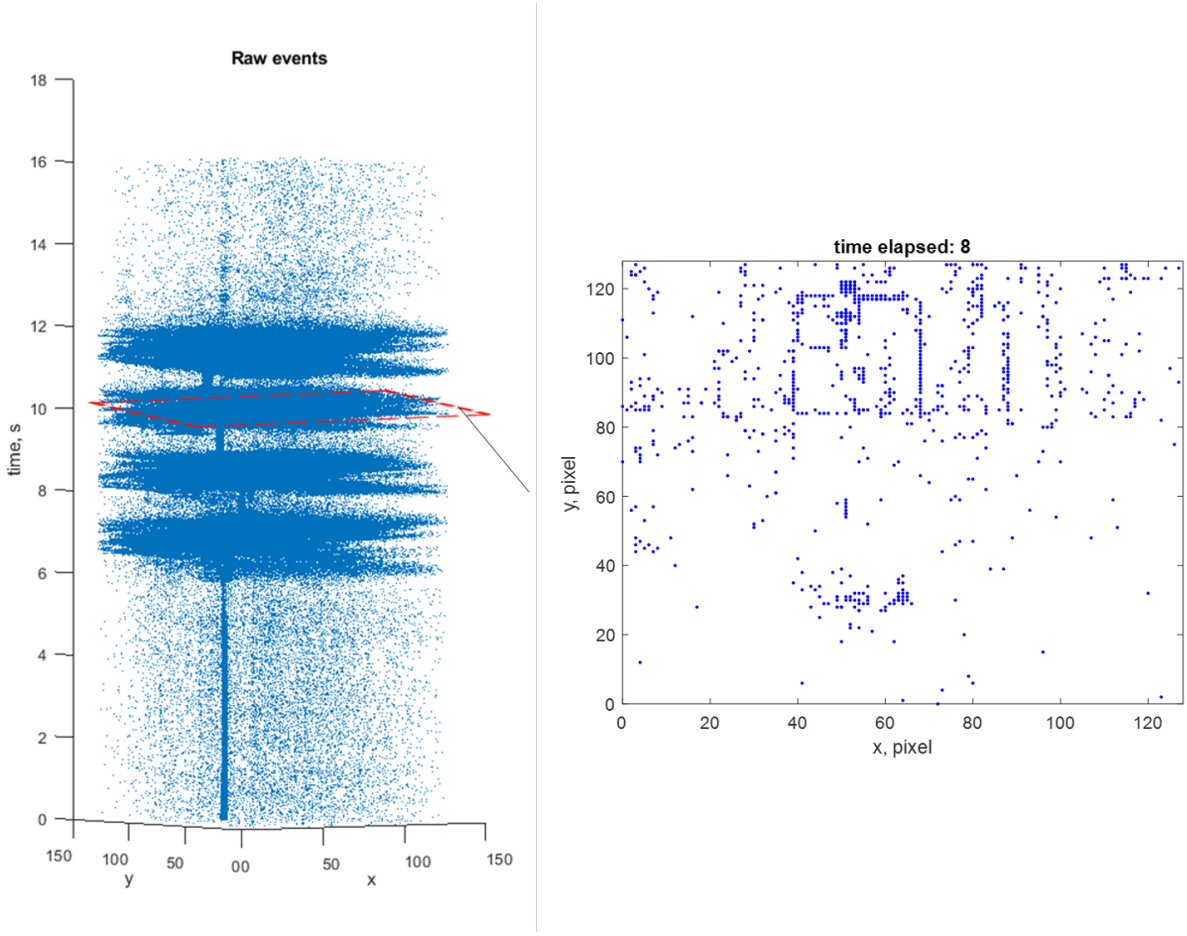


Figure A.35: DVS128 data from the walking experiment.

This section presents the Lab-On-Shoe 1.5 platform integrated with a DVS DVS128. We first developed a custom firmware for the DVS128, and the rotated doted plate experiment tested the firmware. To understand how to interpret the event data set, we experimented with the moving cart. The experiment showed that the sensor could capture the outline of the moving cart. We also noticed that when no object within the FOV was moving, the sensor

still produced events. These events were the noise data and needed to be minimized. Before integrating the DVS128 with the Lab-On-Shoe platform, we investigated the properties of the foot-mounted DVS128 with the indoor walking experiment. By visual observation, the collected data showed that the number of events being generated during the swing phase of a gait cycle was much higher than the amount during the stance phase. Moreover, the collected events presented the outlines of the internal stricture of the indoor environment. Based on these observations, the event camera is potentially a good candidate to be used along with the Lab-On-Shoe system for indoor pedestrian navigation.

Lab-On-Shoe 1.6: Integrating With Magnetometer

Table A.7: COTS components used in Lab-On-Shoe 1.6 platform.

Component	Manufacturer	Model	Quantity	Purpose
FPGA	National Instruments	CompactRIO−9039	1	Processing Unit
Microcontroller	Arduino	Uno	1	Processing Unit
IMU	Analog Device	ADIS16497−3	2	Acceleration and angular rate
SONAR	Devantech	SRF02	4	Ranging
SONAR	Devantech	SRF08	2	Ranging
Altimeter	TE Connectivity	MS5803−01BA	2	Height
Camera	Basler	acA800−200gc	2	Image
Mass Flow Sensor	Renesas Electronics	FS1012	2	Velocity
Dynamic Vision Sensor	IniLabs	DVS128	1	Light Intensity Change
Magnetometer	Triaxis	Melexis MLX90393	2	Magnetic fields
Battery	Powerizer	LiFePO4	1	Power source
Voltage Regulator	Texas Instruments	LM3100EVAL	2	Power management
Voltage Regulator	Texas Instruments	TPS7A1601EVM−046	2	Power management

Lab-On-Shoe 1.6 system was integrated with two IMUs, two barometric altimeters, four foot-to-foot ultrasonic transducers, two downward ultrasonic transducers, two CMOS cameras, and a DVS, and two magnetometers. Table A.7 lists COTS components integrated into the Lab-On-Shoe 1.6 platform. In previous sections of this appendix, it was experimentally

demonstrated that using measurements obtained from foot-to-foot ultrasonic sensors and foot-to-foot cameras to aid ZUPT-based INS could reduce the error of the estimated heading angles. However, the reduction was limited because the ultrasonic sensors and the cameras provided relative orientation between the two feet instead of absolute orientations. The error of the heading angle of the aided pedestrian INS still accumulated over time and exceeded an acceptable bound after navigating for several minutes. To further enhance the navigation performance of the pedestrian INS, magnetometers were integrated into the Lab-On-Shoe 1.6 platform for providing absolute heading measurements.

This section presents the development of the Lab-On-Shoe 1.6 platform. A calibration method for the magnetometer in indoor environments is presented in this section.

The Lab-On-Shoe platform Integrated With Magnetometer MLX90393 A COTS 3-axis magnetometer Melexis MLX90393 manufactured by Triaxis was chosen for the Lab-On-Shoe 1.6 platform. The low-cost magnetometer has a small size, low power consumption, high resolution (0.161 [μT]), and a large full-scale range (44000 [μT]). These properties are favorable for the magnetometer to be integrated with pedestrian navigation systems. The magnetometer was designed to be communicated via either SPI or I2C bus protocols. In our implementation, we selected I2C as the communication protocol between the NI Compact-Rio of the Lab-On-Shoe platform and the magnetometer. The firmware framework of the Lab-On-Shoe platform integrated with the magnetometer is shown in Figure A.36(a). The magnetometer is connected on the same I2C bus for the ultrasonic sensors SRF02 and SRF08 and the barometers MS5308−01BA. The maximum sampling rate of the magnetometer in this configuration is 50 [Hz].

One magnetometer was mounted on top of the IMU on each shoe. The Lab-On-Shoe 1.6 platform integrated with the magnetometer is shown in Figure A.36(b). The x-axis, y-axis, and z-axis of the magnetometer were visually aligned with the three axes of the IMU. Figure

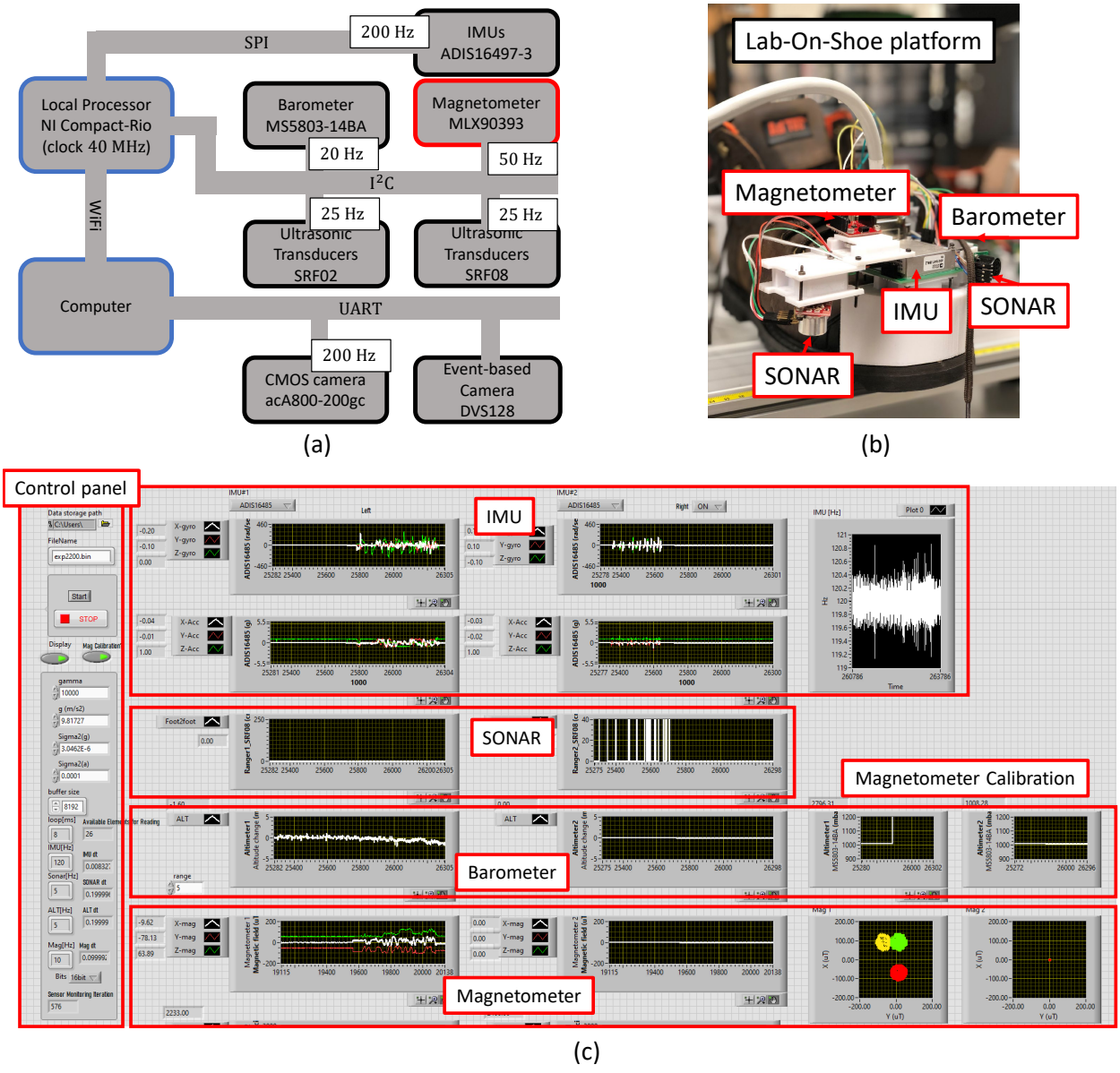


Figure A.36: (a) The communication diagram for the Lab-On-Shoe 1.6 platform. The blocks with the blue frame show computation units. The blocks with the black frame indicate sensors integrated into Lab-On-Shoe 1.6 platform. The red blocks present the magnetometer integrated into Lab-On-Shoe 1.6 platform. (b) The integrated Lab-On-Shoe platform. (c) LabVIEW user interface for the upgraded Lab-On-Shoe 1.6 platform.

A.36(c) presents the user interface for the upgraded Lab-On-Shoe platform. This firmware allows for simultaneously collecting measurements of acceleration and angular velocity with the IMUs, ambient air pressure with the barometers, foot-to-foot relative distance with the foot-to-foot ultrasonic sensors, height relative to the ground with the downward-facing ultrasonic sensors, and magnetic field with the magnetometers.

Calibration of Magnetometers in Indoor Environments

Magnetometer Measurement Model A magnetometer measures magnetic fields expressed in the sensor body frame. The measurement model of a magnetometer can be expressed as follows:

$$\mathbf{B}_k = (\mathbf{I} - \mathbf{D}_k)^{-1}(\mathbf{A}_k\mathbf{H}_k + \mathbf{B}_k + \boldsymbol{\epsilon}_k) = \boldsymbol{\Omega}^{-1}\mathbf{M}_k + \mathbf{B}_k^{'} + \boldsymbol{\epsilon}_k^{'}, \tag{A.2}$$

where $\mathbf{B}_k$ is magnetometer readings indicating the magnetic field in the sensor body frame, $\mathbf{D}_k$ is the scale factor errors, also known as the soft iron distortion, $\mathbf{A}_k$ is the DCM between the body frame and the Earth-fixed frame, $\mathbf{H}_k$ is the geomagnetic field w.r.t.the Earth-fixed frame, $\mathbf{B}_k$ is the magnetometer bias, also known as the hard iron distortion, $\boldsymbol{\epsilon}_k$ is the noise, modeled as a zero-mean Gaussian process. $\mathbf{M}_k = \mathbf{A}_k\mathbf{H}_k$ is the geomagnetic field observed in the body frame, $\boldsymbol{\Omega} = \mathbf{I} - \mathbf{D}_k$. $\mathbf{B}_k^{'} = \boldsymbol{\Omega}^{-1}\mathbf{B}_k$, and $\boldsymbol{\epsilon}_k^{'} = \boldsymbol{\Omega}^{-1}\boldsymbol{\epsilon}_k$. Note that in this measurement model, we assumed that the three axes of a magnetometer are perfectly aligned.

To use Earth's magnetic field to determine the orientation of a magnetometer, the measurement model, expressed in (A.2), is alternatively presented as follows:

$$\mathbf{M}_k = \mathbf{A}_k\mathbf{H}_k = \boldsymbol{\Omega}(\mathbf{B}_k - \mathbf{B}_k^{'}), \tag{A.3}$$

where $\mathbf{M}_k$ is the error-free or calibrated magnetometer readings that only measure geomag-

netic field. Since the geomagnetic field expressed in the Earth-fixed frame is $[0, \kappa, 0]^T$, where κ is the strength of the geomagnetic field, when there is no tilt for the magnetometer, the yaw (heading) angle α of the magnetometer can be estimated as follows:

$$\alpha = \tan^{-1}(\frac{\mathrm{M}_{k,y}}{\mathrm{M}_{k,x}}) \tag{A.4}$$

$$\theta = \tan^{-1}(\frac{-\mathrm{M}_{k,z}}{\sqrt{\mathrm{M}_{k,x}^2 + \mathrm{M}_{k,y}^2}}) \tag{A.5}$$

where $\mathrm{M}_{k,x}, \mathrm{M}_{k,y}$, and $\mathrm{M}_{k,z}$ are the calibrated magnetometer measurements along the x-axis, y-axis, and z-axis, respectively.

In practical implementation, we do not have direct access to $\mathbf{M}_k$ because magnetic field interference contributed from the hard iron distortion and soft iron distortion effects contaminates the magnetometer measurements. The hard iron distortion is generated by magnetic sources other than the geomagnetic field. The sources could include permanent magnets and a power supply. The soft iron distortion changes the magnitude and direction of the earth's magnetic field experienced by the magnetometer. This effect is enhanced when the magnetometer is close to ferromagnetic objects, such as steel. To effectively use a magnetometer to determine the orientation of a navigation system, calibration of the sensor is required. Based on the magnetometer measurement model, the calibration process is equivalent to determining the matrix $\mathbf{\Omega}$ and the vector $\mathbf{B}_k^{'}$ in (A.3).

In-situ Calibration Magnetometers can be effectively calibrated by the TWOSTEP method [12] in situations where orientation at a fixed location or in a small area is of interest. The idea of the TWOSTEP method is that when an ideal (error-free) magnetometer is rotating 360° along the three axes, then the corresponding measurements will form a perfect sphere whose center is at the origin, whereas a magnetometer suffering from hard and soft iron distortions has measurements forming an ellipsoid. The calibration process uses a curve-

fitting technique to estimate a translation vector and an orientation matrix between the sphere and the ellipsoid. The translation vector and the orientation matrix are the $\mathbf{B}_k^{'}$ and the $\mathbf{\Omega}$ in (A.3).

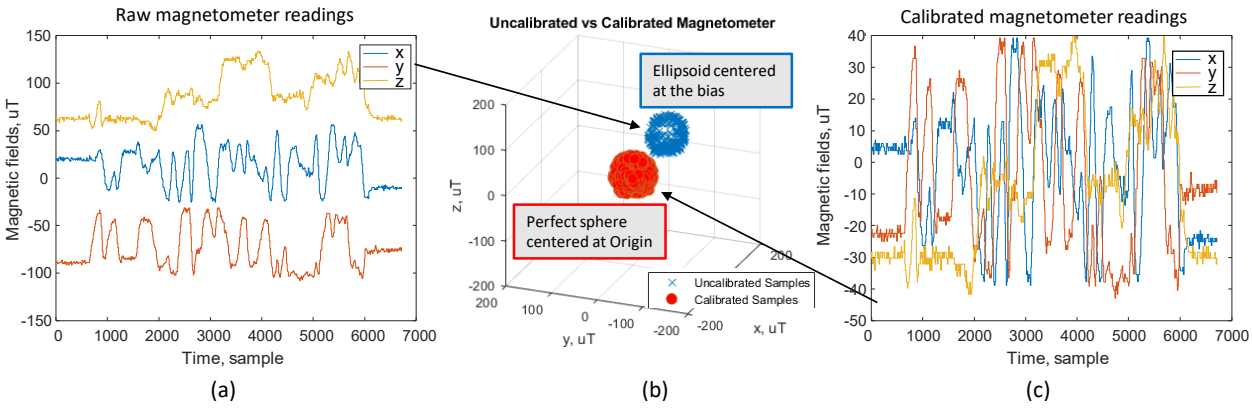


Figure A.37: (a) Uncalibrated magnetometer readings. (b) Calibrated magnetometer readings. (c) Comparison of the uncalibrated and calibrated measurements. Each data point represents a measurement vector of the magnetometer collected at a specific time.

To verify the validity of the TWOSTEP method in our application, we performed an experiment with the magnetometer MLX90393 mounted on the Lab-On-Shoe platform on the second floor of the Engineering Gateway Building at the University of California, Irvine. In the experiment, we rotated the magnetometer by hand for 360° on each of the three axes. Figure A.37(a) shows the corresponding raw readouts of the magnetometer. Each of the blue crosses in Figure A.37(b) represents a raw measurement vector plotted in 3D, which formed an ellipsoid with a non-zero center. We applied the TWOSTEP calibration process to the raw measurements to estimate hard and soft iron distortions. The results are summarized in Table A.8. Figure A.37(c) presents calibrated magnetometer measurements. The calibrated magnetometer measurements shown in Figure A.37(b) formed an ideal sphere centered at the Origin.

Experimental Results To investigate the accuracy of the heading angles estimated based on the calibrated untilted magnetometer measurements, we performed an indoor experiment

Axis	x	y	z
Hard iron distortion	15.3794	−68.0464	92.9233
Soft iron distortion	0.95591	1.0703	0.97843

Table A.8: Estimated hard and soft iron distortions in the in-situ calibration experiment.

with the Lab-On-Shoe platform on the second floor of the Engineering Gateway Building at the University of California, Irvine. At the beginning of the experiment, the Lab-On-Shoe platform was placed on the ground, and the z-axes of the IMU and the magnetometer mounted on the shoe were perpendicular to the ground. The starting position of this experiment was the same as the last experiment conducted for magnetometer calibration. Thus, we used the values for hard iron distortion and soft iron distortion specified in Table A.8 to calibrate the magnetometer measurements collected in this experiment. The initial heading angle was 134° w.r.t the North. Then, we moved to the shoe by hand for 2 [m] and rotated the shoe for approximately 90° so that the heading direction became −136° w.r.t. the North. Next, we moved the shoe for 1 [m] and rotated for 90° again. After the rotation, the heading angle became −46° w.r.t. the North, and we moved the shoe for 2 [m] along this direction. Next, the shoe was turned another 90°, facing a direction of 44° w.r.t. the North, traveling for 1 [m]. Lastly, the shoe returned to the original position and faced the original direction. The duration of the experiment was 73 s. The reference trajectory of this experiment is shown in Figure A.38(a).

We compared the yaw angles estimated by the magnetometer and the ZUPT-aided INS in this experiment. Figure A.38(b) presents the results of the two methods. The yellow curve in FigureA.38(b) indicates the nominal heading angles. We evaluate the accuracy of the yaw angle estimations based on the Root Mean Square Error (RMSE) w.r.t. the nominal heading angles. The RMSE of the heading angle estimated by the ZUPT-aided INS was 23°, and the RMSE for the magnetometer was 36°. This experiment showed that the accuracy of heading angles estimated by magnetometer measurements calibrated with the TWOSTEP method

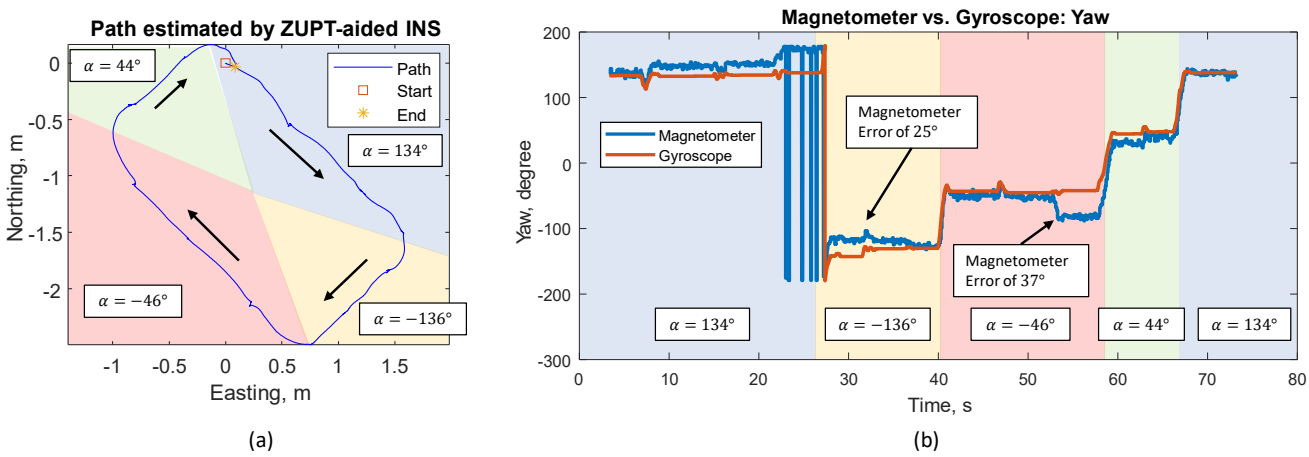
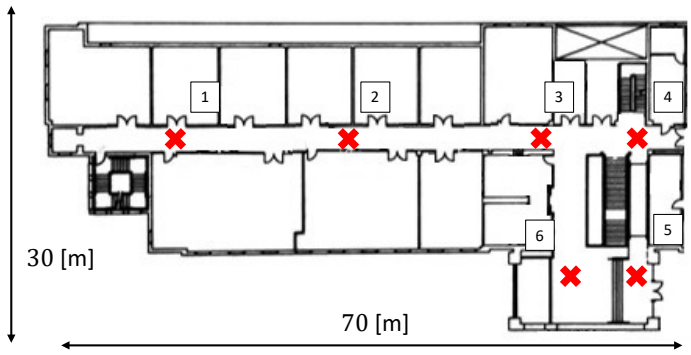


Figure A.38: (a) The reference trajectory estimated by ZUPT-aided INS for the indoor experiments. The different colors indicate different heading directions. (b) Comparison of yaw angles estimated by the magnetometer and the gyroscope. The different colors indicate different heading directions.

was worse than the estimation based on ZUPT-aided INS.

Three things can be noted from Figure A.38(b). First, the yaw angle estimated by the ZUPT-aided INS does not have a large error because the IMU on the Lab-On-Shoe platform was tactical-grade, and the experiment duration was only around one minute. Second, the yaw angle estimation based on magnetometer measurements collected near the starting points had higher accuracy. This implies that the TWOSTEP calibration method was valid in the small area around the starting position. Third, at timestamps of 30 s and 55 s, the yaw angles estimated by the magnetometer had noticeable errors of 25° and 37°. These errors come from the observation that hard iron distortion and soft iron distortion vary at different indoor locations. We could observe the variation by repeating the experiment described in Figure A.37 at other different locations, indicated in Figure A.39(a), in the same building. The results collected at the six different places are presented in the table in Figure A.39(b). Therefore, in the experiment shown in Figure A.38(a), the values specified in Table A.8 were no longer valid to calibrate the magnetometer when operating at different locations. Based on these experiments, we concluded that the TWOSTEP calibration method was insufficient for the magnetometer to effectively provide the heading angle estimation for aiding pedestrian

navigation in indoor environments, and calibration methods that adapt to different locations are required for the magnetometer in our application.



(a)

Location	Distortion type	X-axis	Y-axis	Z-axis
1	Hard	11.89	-65.98	94.83
	Soft	0.89	1.38	0.81
2	Hard	19.56	-64.2	87.37
	Soft	0.99	1.06	0.95
3	Hard	16.74	-63.57	91.41
	Soft	0.95	1.14	0.91
4	Hard	12.49	-63.58	93.16
	Soft	0.96	1.06	0.97
5	Hard	15.01	-57.68	97
	Soft	0.96	1.1	0.99
6	Hard	18.98	-54.61	112.0
	Soft	0.98	1.40	0.84

(b)

Figure A.39: (a) The floor map of the second floor at the Engineering Gateway Building at the University of California, Irvine, and the six locations where the experiments were conducted. (b) Table summarizing the magnetic interference estimated at the six locations.

A magnetometer MLX90393 was integrated with the Lab-On-Shoe platform. The purpose of integrating with the magnetometer is to provide compensation for the yaw angle for pedestrian INS. To minimize the effects of magnetic interference presented in indoor environments, we applied the TWOSTEP algorithm to calibrate the magnetometer measurements. We experimentally tested the accuracy of heading angles estimated by the calibrated magnetometer in an indoor environment. The experimental results showed that the heading angle estimated by the magnetometer had a larger error than the ZUPT-aided INS. The larger errors were caused by the fact that magnetic interference varies at different locations in indoor environments. Therefore, the TWOSTEP algorithm is insufficient for the magnetometer to be used for indoor navigation, and calibration methods are required to adapt to different locations.

Generation of Reference Trajectory with Foot-mounted IMUs In order to obtain a reliable magnetic interference map, a practical approach is to collect magnetometer measurements at different locations and then estimate the magnetometer measurements at unexplored

locations. Since the earth's magnetic field is uniform over a small area, we can deduce that any magnitude of the measured magnetic field that is not equal to the local magnetic field is generated by surrounding magnetic interference. In the first step of this approach, it is necessary to determine the accurate locations where the magnetometer measurements are collected. Since we use foot-mounted INS, it is important to develop an algorithm for the system that achieves high-accuracy localization results.

Since we attempt to obtain a magnetic field map in an off-line manner, we can utilize known foot motion to enhance the performance of the ZUPT-aided INS. In the data collection process, we restricted the motion of a user of the Lab-On-Shoe platform so that the user can only walk straight or turn 90°. Since the motions are restricted, we can assume that the heading angle during the stance phase in a step is either the same as the previous step or has a 90° difference. Then, in the EKF, we can utilize this information to update the yaw angle state. The novelty of the developed approach is in the EKF measurement model.

The measurement model that uses the ZUPT algorithm and the information on the known foot motion is described as follows:

$$z_{\text{feet}} = \gamma_{\text{previous step}}, z_{\text{ZUPT}} = \mathbf{v}_n^b,$$

and the corresponding measurement matrices are

$$\mathbf{H}_{\text{feet}} = \begin{bmatrix} 0 & 0 & 1 & \mathbf{0}_{1\times 12} \end{bmatrix},$$
$$\mathbf{H}_{\text{ZUPT}} = \begin{bmatrix} \mathbf{0}_{3\times 3} & I_{3\times 3} & \mathbf{0}_{3\times 9} \end{bmatrix}.$$

The overall measurement model is

$$\mathbf{z} = \begin{bmatrix} \gamma_{\text{previous step}} \\ \mathbf{v}_n^b \end{bmatrix}, \mathbf{H} = \begin{bmatrix} \mathbf{H}_{\text{feet}} \\ \mathbf{H}_{\text{ZUPT}} \end{bmatrix}$$

where $\gamma_{\text{previous step}}$ is the yaw angle estimates computed in the previous detected stance phase.

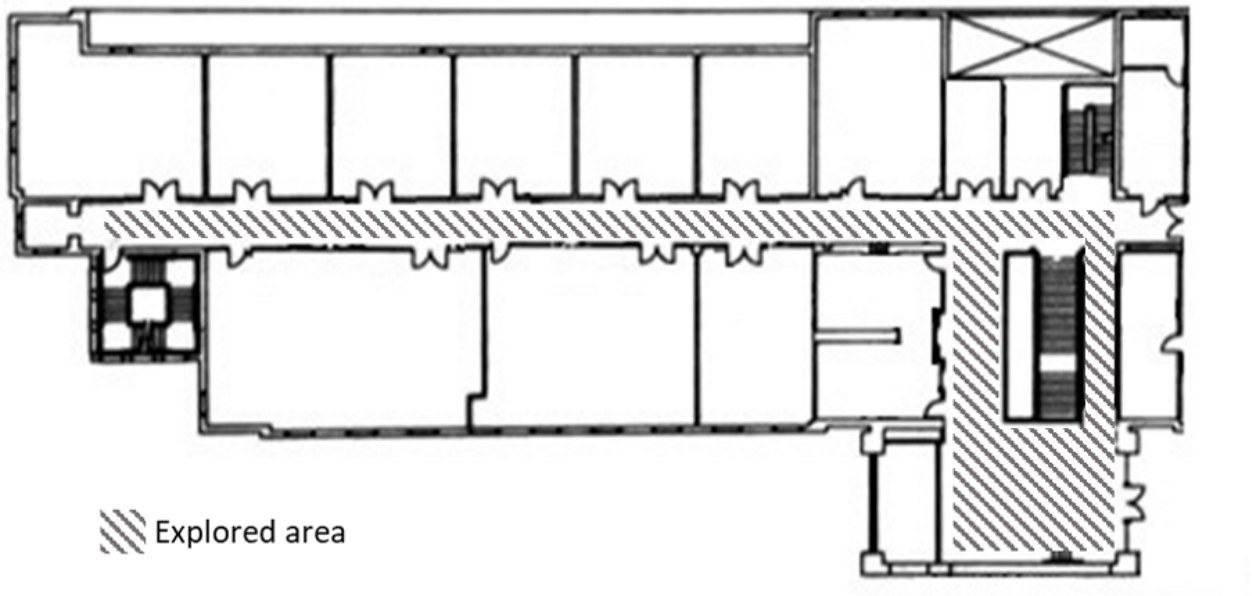


Figure A.40: A floor plane of the experiment field.

To test the localization performance of the proposed approach, we conducted an experiment with the Lab-On-Shoe platform where a pedestrian passed by different locations in an indoor environment at the Engineering Gateway building at the University Of California, Irvine. A floor plan of the experiment field is shown in Figure A.40. In the experiment, the pedestrian walked straight or turned 90°. The walking speed was 60 [step/min], and the trajectory was a closed loop. The duration of the experiment was 28 minutes.

Figure A.41 shows a comparison of trajectories estimated by the standalone ZUPT-aided INS and the ZUPT-aided INS using known foot motion. In this experiment, the ZUPT-aided INS had an accumulated displacement error of 92.8 [m], while the proposed approach had

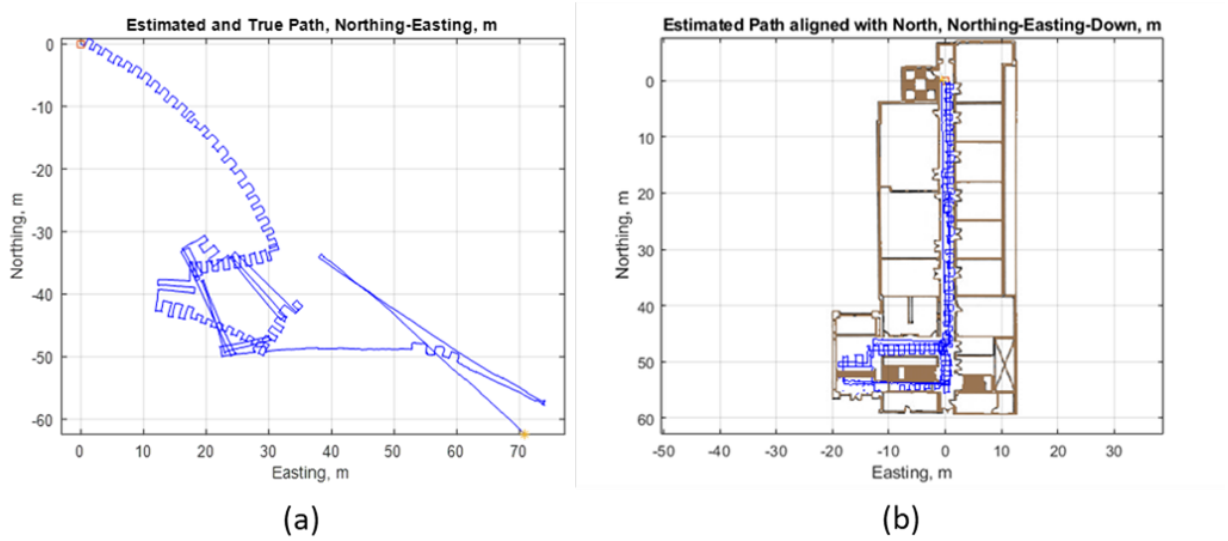


Figure A.41: (a) Trajectory estimated by ZUPT-aided INS. (b) Trajectory estimated by ZUPT-aided INS using known foot motions. The accumulated displacement error was reduced from 92.8 [m] to 1.2 [m]

an error of 1.2 [m]. The improvement in displacement error was a result of yaw angles being corrected in every step.

In this section, an enhancement approach was presented to extend the usage of foot-mounted IMU to generate a reliable and accurate reference trajectory. The concept of the enhancement method was to increase the observability of the yaw angle. The generated reference trajectories can be used to survey the magnetometer interference in indoor environments. We demonstrated the enhanced ZUPT-aided INS could achieve a displacement error of 1.2 [m] after navigating for 28 minutes.

Lab-On-Shoe 1.7: Integrating With LTE Receiver

The Lab-On-Shoe 1.7 platform was upgraded based on the Lab-On-Shoe 1.6 platform and was integrated with an additional software-defined LTE receiver. This platform was a collaborative effort. The Lab-On-Shoe platform was developed by the author of this thesis, Chi-Shih Jao, under the supervision of UCI professor Andrei Shkel, and the LTE receiver

was developed by UCI graduate student Ali Abdallah, under the supervision of OSU professor (former UCI professor) Zak Kassas.

Table A.9: COTS components used in Lab-On-Shoe 1.7 platform.

Component	Manufacturer	Model	Quantity	Purpose
FPGA	National Instruments	CompactRIO−9039	1	Processing Unit
Microcontroller	Arduino	Uno	1	Processing Unit
DSP	National Instruments	USRP−2955	1	Signal Processing
IMU	Analog Device	ADIS16497−3	2	Acceleration and angular rate
SONAR	Devantech	SRF02	4	Ranging
SONAR	Devantech	SRF08	2	Ranging
Altimeter	TE Connectivity	MS5803−01BA	2	Height
Camera	Basler	acA800−200gc	2	Image
Mass Flow Sensor	Renesas Electronics	FS1012	2	Velocity
Dynamic Vision Sensor	IniLabs	DVS128	1	Light Intensity Change
Magnetometer	Triaxis	Melexis MLX90393	2	Magnetic fields
LTE Antenna	Laird	Omni-direction	4	LTE receiver
Battery	Powerizer	LiFePO4	1	Power source
Voltage Regulator	Texas Instruments	LM3100EVAL	2	Power management
Voltage Regulator	Texas Instruments	TPS7A1601EVM−046	2	Power management

Hardware Description

Figure A.42(c) shows the Lab-On-Shoe 1.7 platform The receiver was equipped with two consumer-grade cellular omni-directional Laird antennas to collect data from the same carrier frequency, which we connected to a dual-channel National Instruments (NI) universal software radio peripherals (USRPs)−2954R to simultaneously down-mix and synchronously sample LTE signals at 10 Msps. The sensors that were located on the Lab-On-Shoe platform and were used in the experiment included the Analog Device IMU ADIS16497−3, barometric altimeter MS5803−01BA, and ultrasonic transducer SRF08. The sampling rate of the IMU, the altimeter, and the ultrasonic sensor was 120 Hz, 20 Hz, and 25 Hz, respectively. Table A.9 lists COTS components included in the Lab-On-Shoe 1.7 platform. The Lab-On-Shoe

platform is referred to as the LONS in this section.

Experimental Layout and Experimental Setup

In this section, an experiment was conducted to evaluate the performance of the LONS-LTE integrated system for both integration architectures. The experiment was conducted at the Engineering Gateway building at the University of California, Irvine, USA. The pedestrian-mounted receiver receives signals from three U.S. cellular providers: T-Mobile, Verizon, and AT&T, transmitting at four different frequencies, as summarized in Table A.10. Locations of the LTE eNodeBs are shown in Figure A.42(a). In this experiment, a pedestrian who carried the Lab-On-Shoe platform moved the cart where the LTE receivers were mounted. The experimental setup is shown in Figure A.42(c). During the experiment, the Lab-On-Shoe platform and the LTE receivers simultaneously collected measurements. The pedestrian walked in a straight line for 45 meters, turned 180 deg, and walked back to the starting point. The ground true trajectory is illustrated with the red lines in Figure A.42(b). The experimental results are presented in the next section.

Table A.10: LTE ENodeBs' Characteristics

eNodeB	**Carrier frequency [MHz]**	N_{ID}^{Cell}	**Bandwidth [MHz]**	**Cellular provider**
1	1955	93	20	AT&T
2	2125	223	20	Verizon
3	1955	11	20	AT&T
4	1955	198	20	AT&T
5	2145	441	20	T-Mobile
6	2112.5	401	20	AT&T

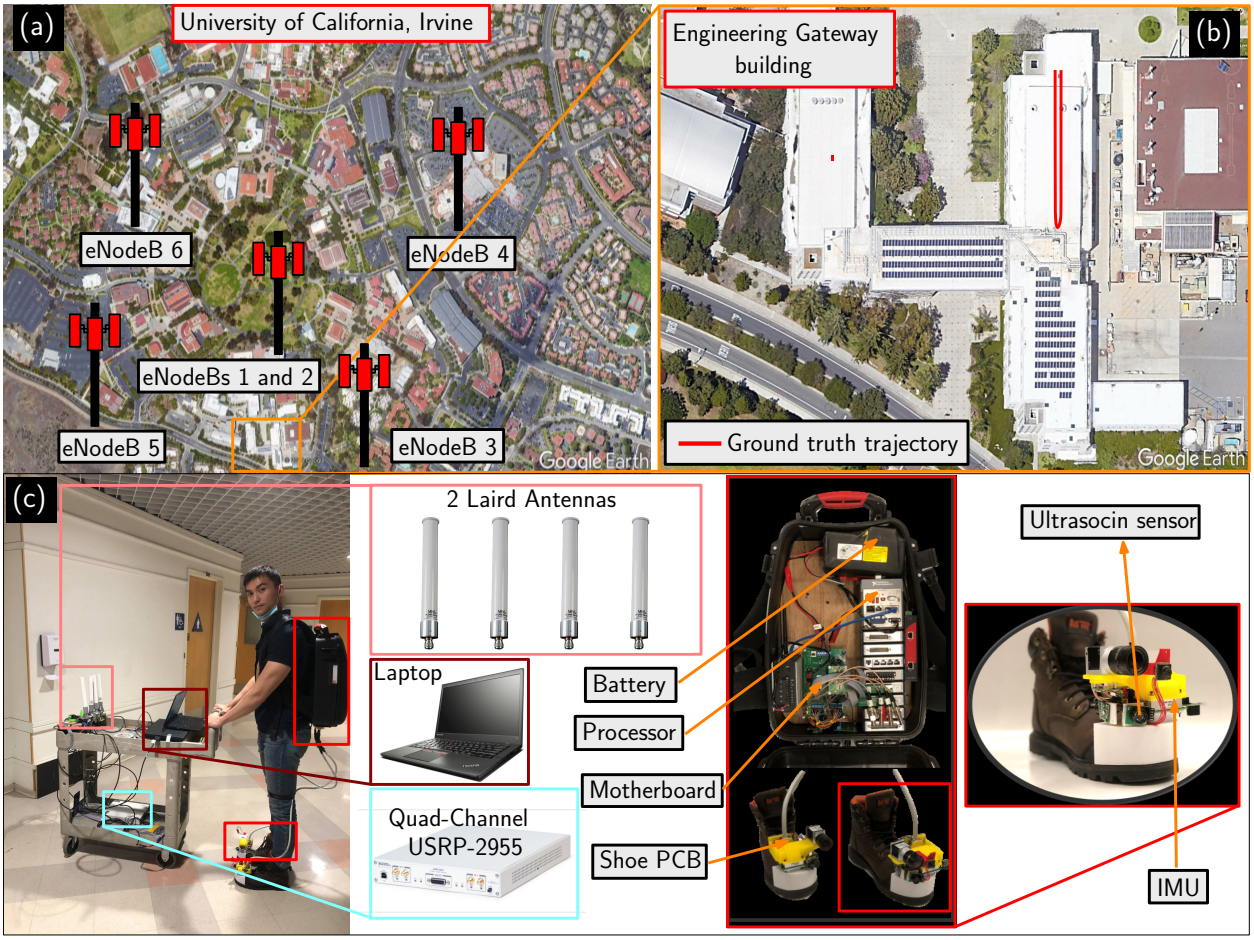


Figure A.42: Experimental layout and experimental setup for LONS-LTE experiment: (a) LTE eNodeBs' positions, (b) Engineering Gateway building where the experiment was performed and the ground truth trajectory, and (c) experimental setup.

Experimental Results

The LTE data was processed in a post-processing manner, and the resulting pseudoranges, which are plotted in Figure A.43, were extracted after removing the initial bias. The resulting pseudoranges were fed to the EKF to generate the standalone LTE solution. Moreover, the LONS sensors data was fed into the navigation framework. The standalone navigation solutions are shown in Figure A.44. The results are summarized in Table A.11.

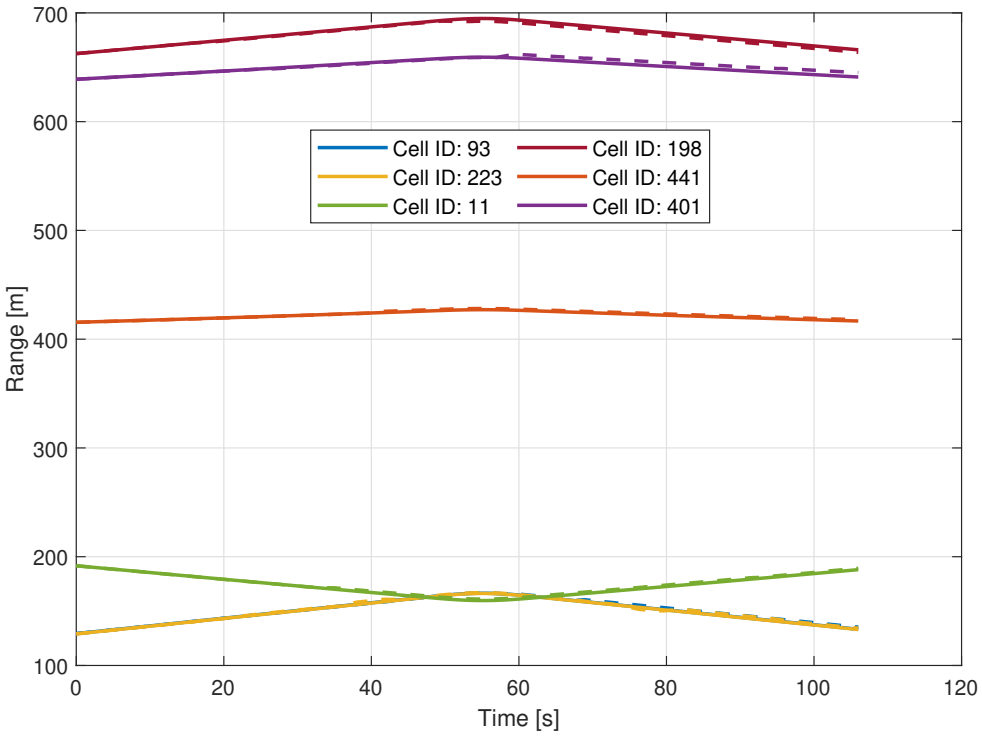


Figure A.43: LTE-SAN-DNN pseudoranges after removing the initial bias using the ground truth.

We would like to point out that, based on the estimated trajectories shown in Figure A.44, the solutions produced by the Lab-On-Shoe platform seemed to have a higher accuracy during the entire experiment than the standalone LTE system. However, the duration of the experiments was around 3 minutes, and the pedestrian walked at a stable pace. Thus, the amount of position drift existing in the ZUPT-aided INS was still at a low level. For navigation tasks with longer duration and trajectories, we would expect that the displacement

error of the ZUPT-aided INS accumulates over time while the error of the LTE system is bounded.

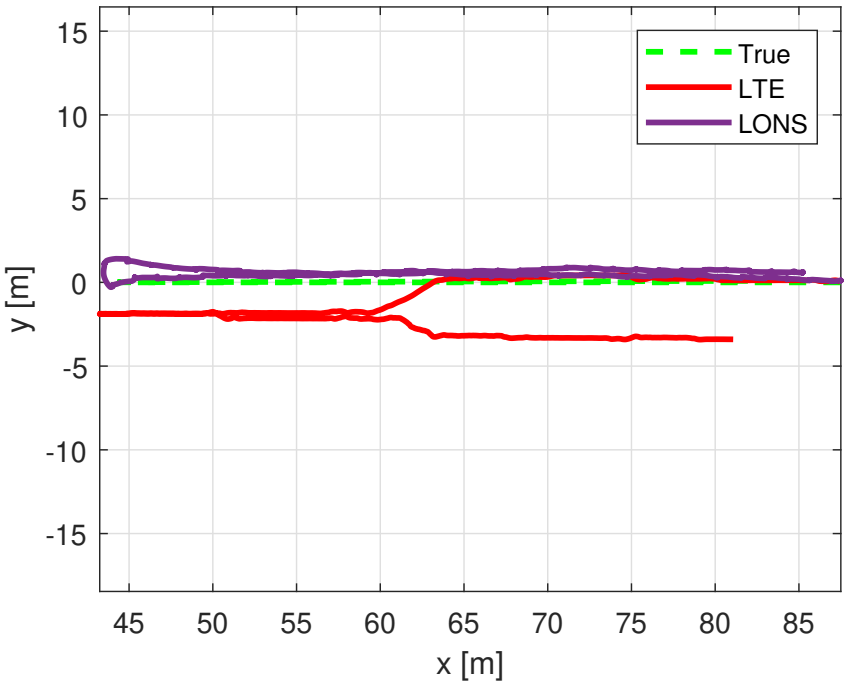


Figure A.44: The LTE and the LONS standalone navigation solution.

Table A.11: Indoor Positioning Performance

Framework	**RMSE [m]**	**Standard deviation [m]**	**Final error [m]**
LONS	1.20	0.60	1.51
LTE	2.36	1.26	3.87

Lab-On-Shoe 1.8: Integrating With UWB Transreceivers

The Lab-On-Shoe 1.8 platform was upgraded based on the Lab-On-Shoe 1.7 platform and was integrated with additional UWB modules. The development of this platform was a collaborative effort. The Lab-On-Shoe platform was developed by the author of this thesis, Chi-Shih Jao, under the supervision of UCI professor Andrei Shkel, and the UWB modules

were developed by UCI graduate students Changwei Chen and Min-Won Seo, under the supervision of UCI professor Solmaz Kia.

Table A.12: COTS components used in Lab-On-Shoe 1.8 platform.

Component	Manufacturer	Model	Quantity	Purpose
FPGA	National Instruments	CompactRIO−9039	1	Processing Unit
Microcontroller	Arduino	Uno	1	Processing Unit
DSP	National Instruments	USRP−2955	1	Signal Processing
IMU	Analog Device	ADIS16497−3	2	Acceleration and angular rate
SONAR	Devantech	SRF02	4	Ranging
SONAR	Devantech	SRF08	2	Ranging
Altimeter	TE Connectivity	MS5803−01BA	2	Height
Camera	Basler	acA800−200gc	2	Image
Mass Flow Sensor	Renesas Electronics	FS1012	2	Velocity
Dynamic Vision Sensor	IniLabs	DVS128	1	Light Intensity Change
Magnetometer	Triaxis	Melexis MLX90393	2	Magnetic fields
LTE Antenna	Laird	Omni-direction	4	Signals of opportunity
UWB	DecaWave	DWM1000	2	Cooperative localization
Battery	Powerizer	LiFePO4	1	Power source
Voltage Regulator	Texas Instruments	LM3100EVAL	2	Power management
Voltage Regulator	Texas Instruments	TPS7A1601EVM−046	2	Power management

Hardware Description Table A.12 lists all COTS components integrated into the Lab-On-Shoe 1.8 platform. To test the performance of cooperative localization with the Lab-On-Shoe platform, we added a DWM1000 UWB module to the platform. The firmware level design for the integrated Lab-On-Shoe platform is illustrated with the block diagram in Figure A.45(a). When acquiring measurements from the UWB DWM1000, the sensor is first communicated with an Arduino, which is an external micro-controller, via the communication protocol I^2C, and then the Arduino sends the measurements to a remote laptop to synchronize with sensor measurements collected from the Lab-On-Shoe platform. A screenshot of the LabVIEW user interface for the integrated Lab-On-Shoe platform is shown in Figure A.45(b). When the system is running, measurements of IMUs, inter-foot ranging, barometer, magnetometer,

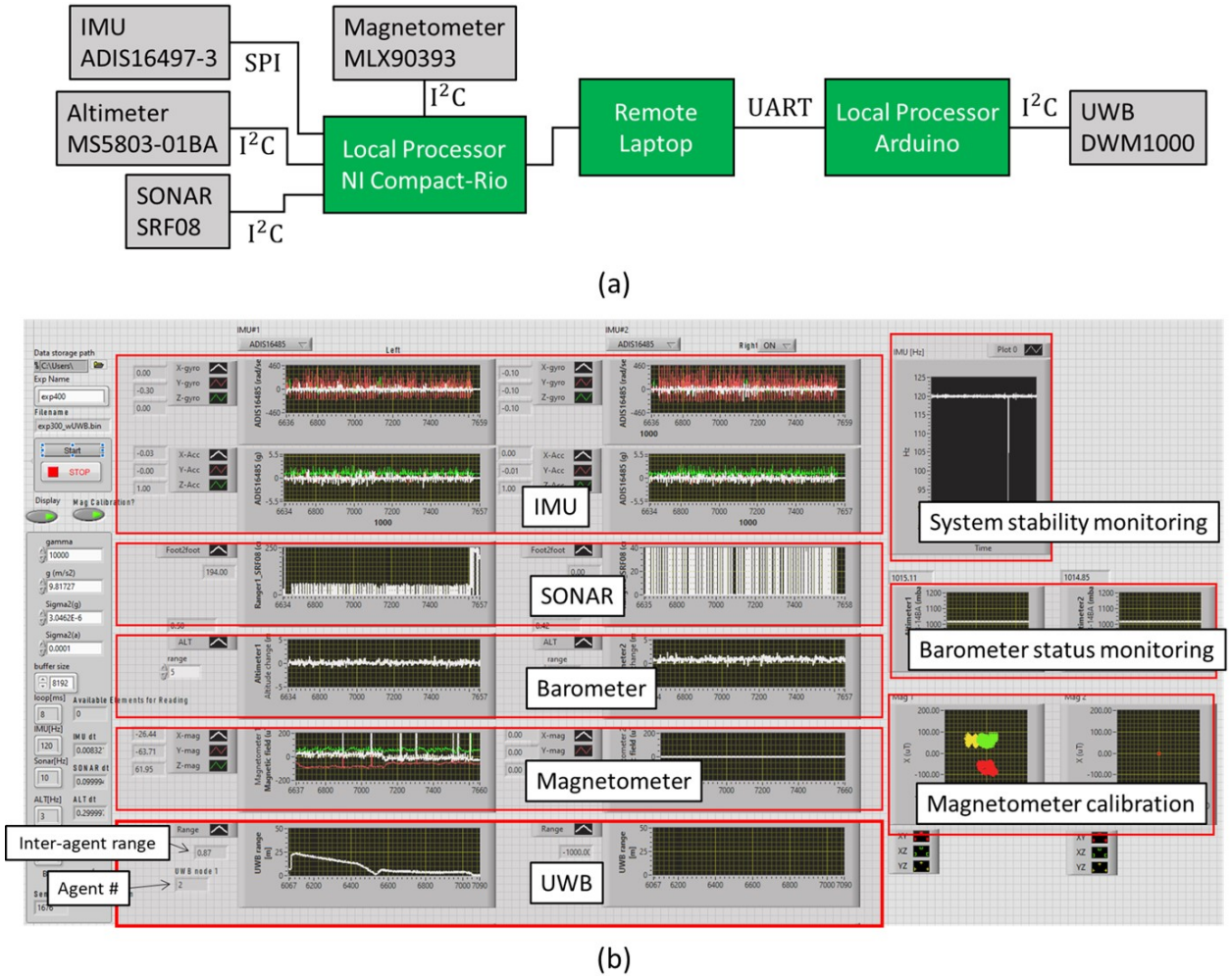


Figure A.45: (a) Hardware schematic for integration of the Lab-On-Shoe platform and a UWB. (b) The user interface for the Lab-On-Shoe platform integrated with a DWM1000 UWB sensor.

and UWBs are simultaneously collected, saved, and displayed. The UWB measurements include information about inter-agent ranges, signal power, and agent identification. In our implementation, the obtained UWB signal power is used in the process of NLOS detection.

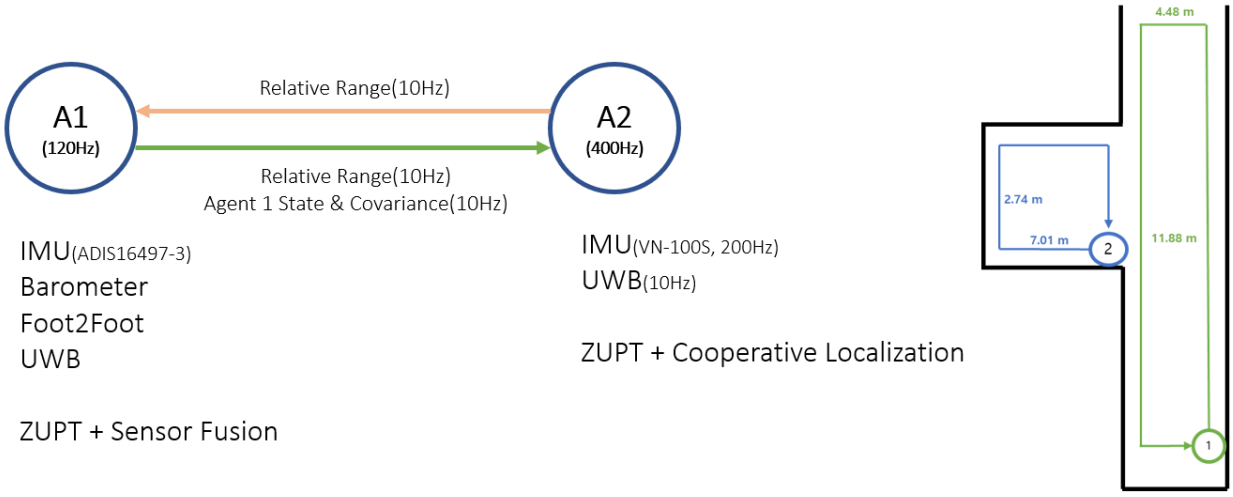


Figure A.46: The configuration of the integration experiment and scenario

Experimental Results We performed several integration experiments for validating hardware/software integration of CL and Lab-On-Shoe by studying the cooperative localization performance. One example scenario is shown in Figure A.46 (Left). In this experiment, two agents in the Engineering Gateway Building at UCI walked along a reference trajectory simultaneously according to the schematic shown in Figure A.46 (Right). Here, agent 1 (A1) is equipped with an IMU (ADIS 16497−3), a barometer (MS5803−01BA), two pairs of foot-to-foot ultrasonic transducers (SRF02), and a UWB transceiver (DecaWave DWM1000), while agent 2 (A2) is only equipped with an IMU (VectorNav VN−100) and a UWB transceiver (DecaWave DWM1000). In this experiment, agent 1 sends its states and covariance to agent 2 using UWB at a rate of 10Hz. Agent 1 walked along a reference trajectory (green) as shown in Figure A.46. At the same time, agent 2 walked along a reference trajectory (blue), as shown in Figure A.46. During the test, inter-agent measurements between the two agents were taken under Line-of-Sight (LoS) as well as Non-Line-of-Sight (NLoS) conditions, as

shown in Figure A.47. While both agents implemented the ZUPT-aided INS, the estimation accuracy of agent 1 was higher than agent 2 because agent 1 used a better IMU and had many complementary sensors. Therefore, we focused on agent 2's localization performance to verify the system and the performance of the cooperative localization.

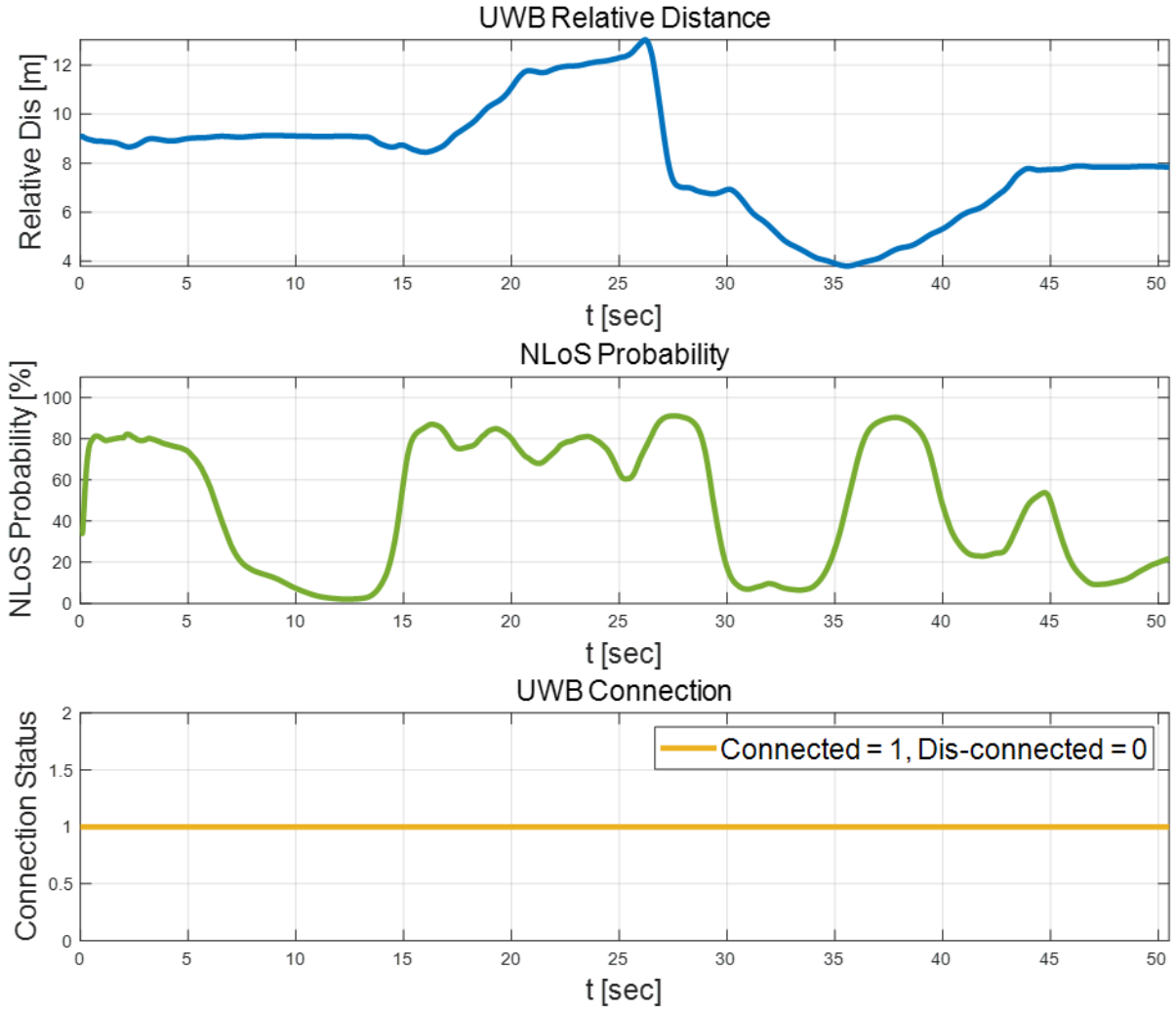


Figure A.47: UWB ranging measurement, NLoS probability based on power identification, and connection status during integration experiment.

To assess the performance of the cooperative localization performance of the heterogeneous agents, we compared localization performance for agent 2 in two cases (executed in parallel): (1) IMU/ZUPT and (2) IMU/ZUPT + DMV CL (with our bias compensation) as shown in Figure A.48. In case (1), even though agent 2 walked slowly, ZUPT was not enough

to maintain the drift in the estimates. Thus the location estimate started to deviate from the true trajectory (see case (1) in Figure A.48). In case (2), we used our proposed CL, which includes a bias compensation measure for NLoS rangings, to improve the localization accuracy of agent 2 by processing the inter-agent UWB range measurements. As we can see CL leads clearly to considerable improvement in agent 2's localization (see case (2) in Figure A.48).

These experiments brought to our attention that ZUPTing feedback in its current form leads to inconsistent reductions in the filter uncertainty (this can be observed by small size of the uncertainty ellipsoids (3σ) shown on the trajectory that do not contain the ground truth). As future work, we will revisit the ZUPTing feedback modeling and make appropriate changes so that the final estimates are consistent.

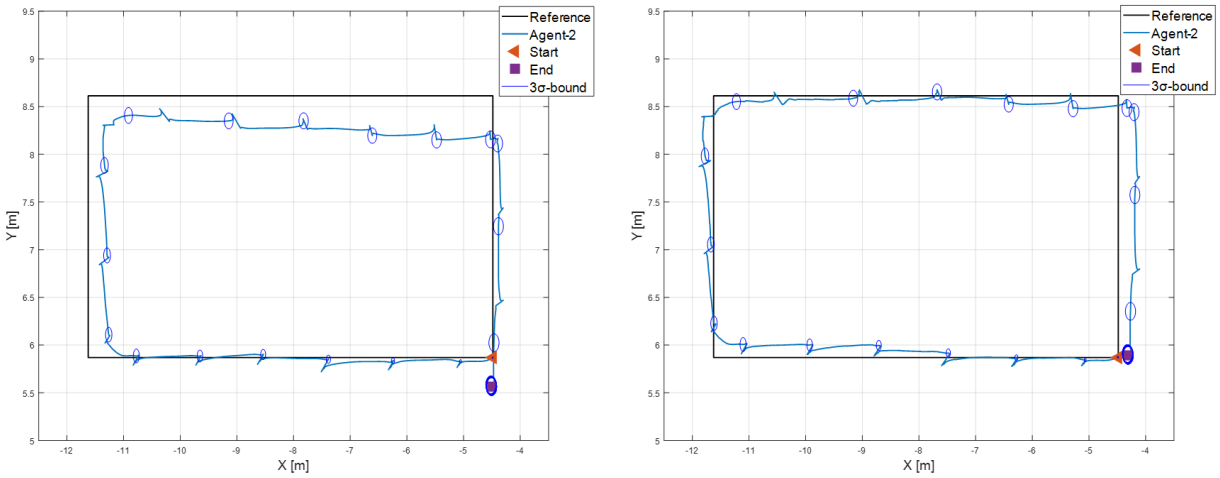


Figure A.48: Agent 2's localization results for 2 cases: (1) IMU/ZUPT, (2) IMU/ZUPT + DMV CL (with our bias compensation)

In this section, we focused on the integration of the Lab-On-Shoe platform and cooperative localization solutions. We added UWBs to the Lab-On-Shoe platform to obtain inter-agent ranging measurements and used the integrated platform to conduct cooperative localization experiments of two agents. The experimental results showed that the localization accuracy of the agent equipped with a low-performance IMU was greatly improved when cooperative

localization was used.

A.1.2 Lab-On-Shoe 2: Everything on the Shoes

The Lab-On-Shoe 2 platforms are extended versions of the Lab-On-Shoe 1 platform with modifications: 1) onboard processor choices from FPGAs to microcontrollers and 2) hardware architecture from backpack-and-shoe implementation to only-shoe implementation. The Lab-On-Shoe 2 platform has two variations, including the Lab-On-Shoe 2.0 platform discussed in Section A.1.2 and the Lab-On-Shoe 2.1 platform discussed in Section A.1.2.

Lab-On-Shoe 2.0: Firmware on Teensy 4.0 Microcontroller

In this section, we discuss the development progress of the Lab-On-Shoe 2.0 platform. The discussion includes hardware, firmware, and testing of the Lab-On-Shoe 2.0.

Hardware and Firmware Compared to the earlier version of the Lab-On-Shoe system, the Lab-On-Shoe 2.0 system replaces the National Instruments CompactRIO controller with microcontroller Teensy 3.2 and equips with additional Ultra-Wideband (UWB) devices. A prototype of the Lab-On-Shoe 2.0 is shown in Figure A.49. Table A.13 lists all integrated components of the platform. In the current implementation of the Lab-On-Shoe 2.0 system, all the components are mounted on the shoes, and the backpack used for the previous version is no longer needed. Each shoe is equipped with a deterministic module and a cooperative module. The deterministic module includes a microcontroller teensy 3.2, a Bluetooth module, an IMU, three ultrasonic transducers, and an altimeter. The cooperative module contains a microcontroller teensy 3.2, a Bluetooth module, and a UWB.

A block diagram of firmware for Lab-On-Shoe 2.0 is presented in Figure A.50. The micro-

Table A.13: COTS components used in Lab-On-Shoe 2.0 platform.

Component	**Manufacturer**	**Model**	**Quantity**	**Purpose**
Microcontroller	Teensy	Teensy 3.2	2	Processing Unit
IMU	Analog Device	ADIS16497−3	2	Acceleration and angular rate
SONAR	Devantech	SRF02	4	Ranging
SONAR	Devantech	SRF08	2	Ranging
Altimeter	TE Connectivity	MS5803−01BA	2	Height
Magnetometer	Triaxis	Melexis MLX90393	2	Magnetic fields
UWB	DecaWave	DWM1000	2	Cooperative localization
Bluetooth	HiLetgo	HC−05	2	Data Transmission
Battery	SparkFun	Lithium-ion	2	Power source

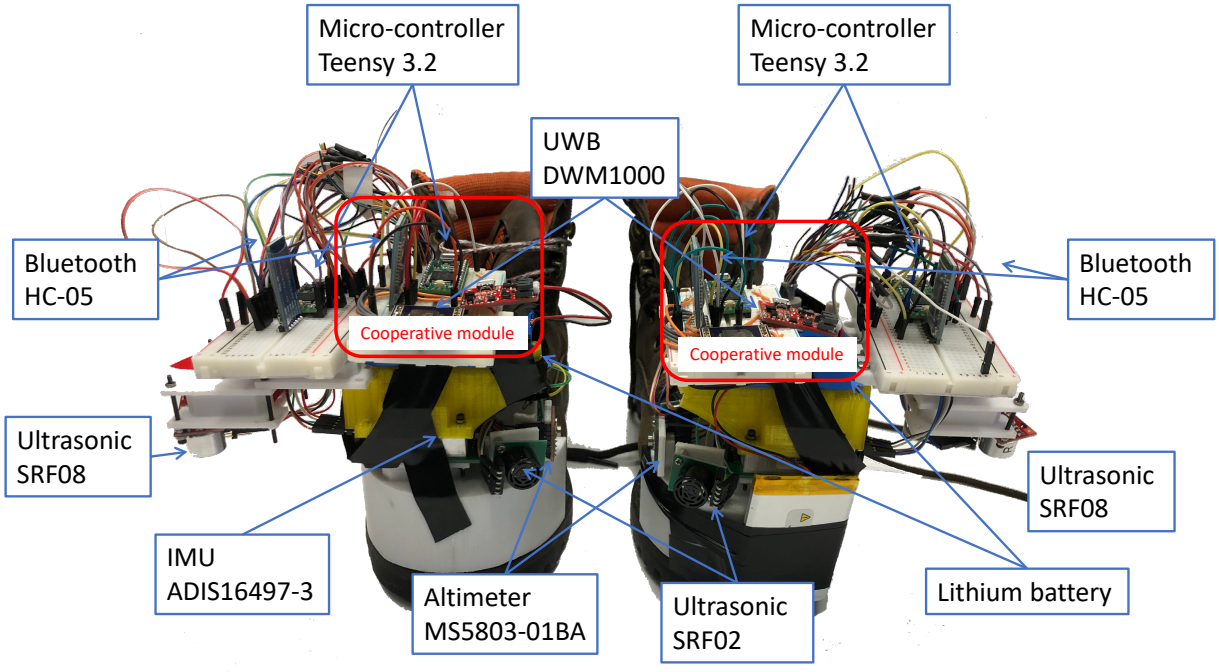


Figure A.49: A prototype of the Lab-On-Shoe 2.0 platform.

controller in the cooperative module is used to acquire range measurements from the UWB via SPI protocol at a rate of 10 [Hz]. The acquired UWB measurements are saved in a local variable on the microcontroller. The value of the variable is sent via Bluetooth to the microcontroller located in the deterministic module at a rate of 20 [Hz]. The microcontroller in the deterministic module is responsible for collecting measurements from IMUs, ultrasonic sensors, altimeters, and UWBs and transmitting the measurements via UART protocol on a USB cable to a laptop at a rate of 1000 [Hz]. The sampling rates of the IMU, the ultrasonic sensors, and the altimeters were 1000 [Hz], 100 [Hz], and 25 [Hz], respectively. In the current implementation, the laptop is used for saving and visualizing acquired sensor measurements.

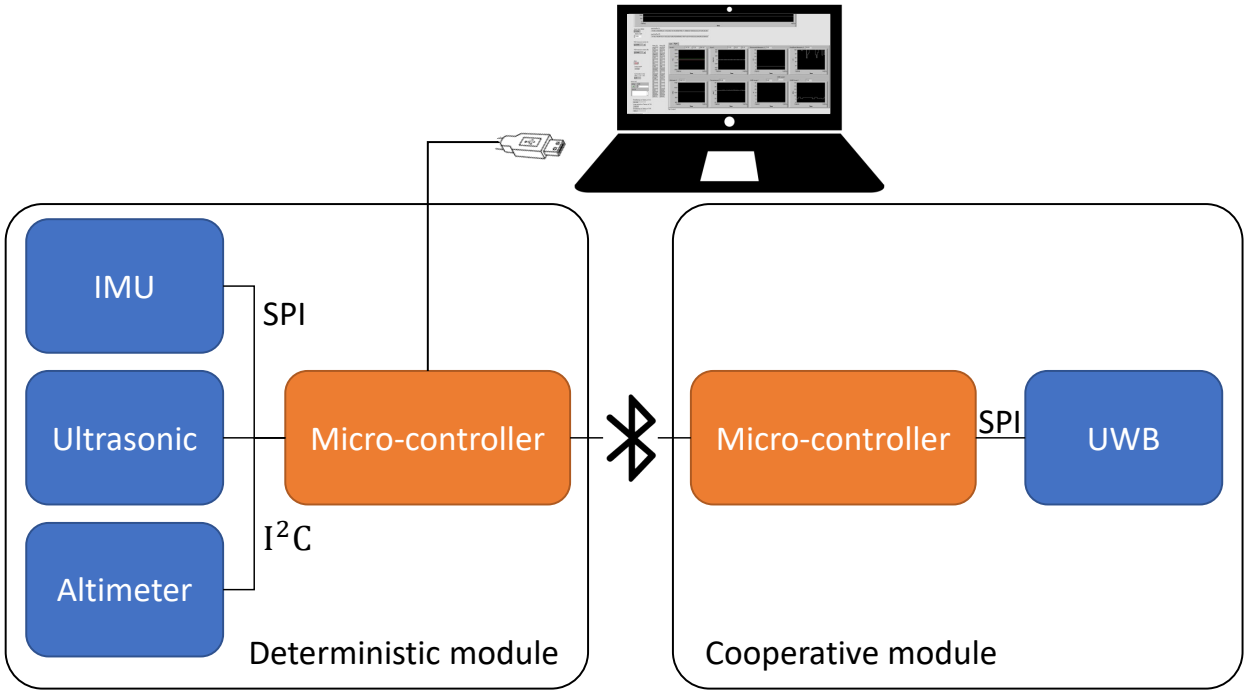


Figure A.50: A block diagram describing the firmware deployed on the Lab-On-Shoe 2.0 platform.

Indoor Navigation Testing To test performance of the Lab-On-Shoe 2.0 platform, we conducted the following two tests. The first test was two indoor pedestrian navigation experiments that were conducted at the Engineering Gateway Building at the University of California, Irvine. The two experiments, carried out by two different people, had similar tra-

jectory length, which was around 600 [m] and lasted about 700 [s]. Figure A.51(a) presents the experimental setup, which included LTE and GPS receivers. It its worth mentioning that the LTE and GPS measurements were not used in the results discussed in the next paragraph. Figure A.51(b) illustrates activities and terrains the pedestrians experienced during the experiments.

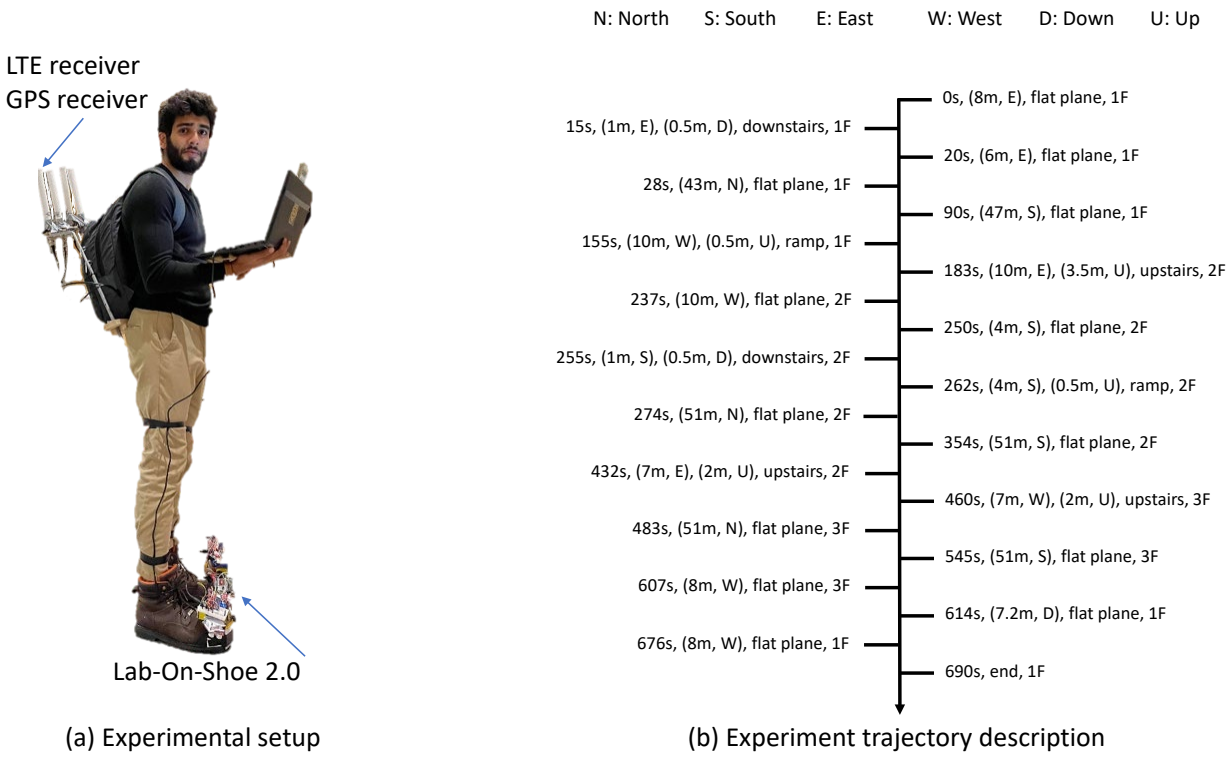
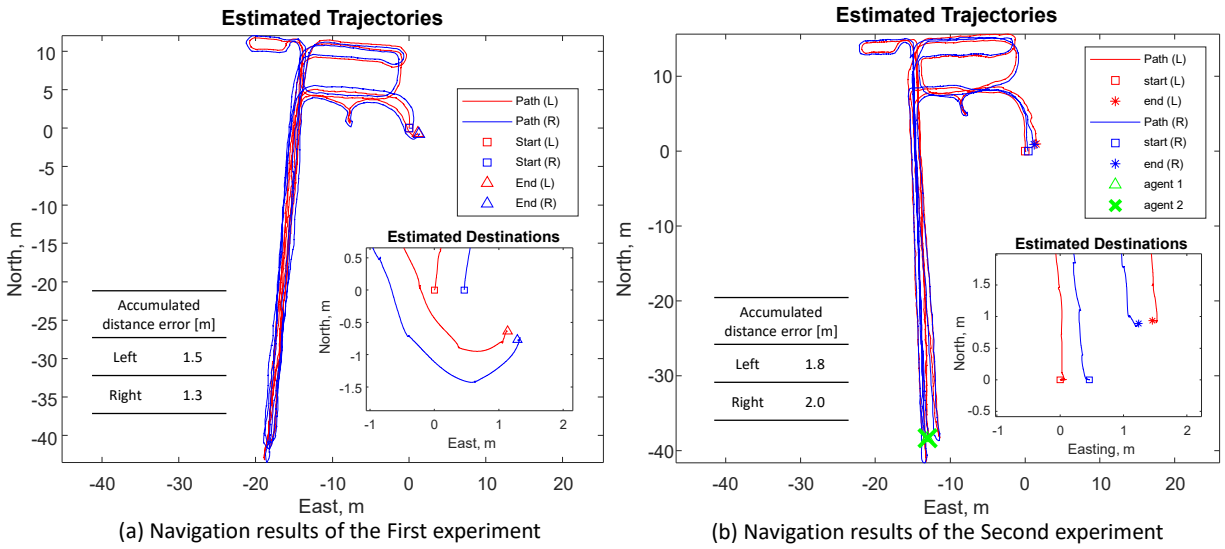


Figure A.51

Figure A.52 demonstrates the navigation solutions estimated by a ZUPT-aided INS enhanced by an altimeter and foot-to-foot ranging measurements. In the first experiment, the accumulated position error was 1.5 [m] for the left foot and 1.3 [m] for the right foot. In the second experiment, the accumulated position error was 1.8 [m] for the left foot and 2.0 [m] for the right foot.

Notice that in the experiment shown in Figure A.52(b), two UWB-based beacons were placed at locations indicated by a green triangle and a green cross. The one marked with the green triangle was located on the second floor of the building, and the one marked with the green

cross was placed on the third floor. UWBs located on the beacons were paired with the UWBs mounted on the Lab-On-Shoe 2.0 system, and connections between the devices were established once the agent walked within the operation range of the UWBs. In the results presented in this section, the range measurements between the pedestrian and the beacons were not used to enhanced the navigation solutions.



(a) Navigation results of the First experiment

(b) Navigation results of the Second experiment

Figure A.52

Reliability Test In order to investigate the reliability of the microcontroller Teensy 3.2 and the firmware on the Lab-On-Shoe 2.0 system, we tested the system with a long navigation experiment. The path of the navigation experiment included indoor environments and outdoor environments, and the total duration was around one hour. The trajectory length was around 2 [km].

Figure A.53 present the navigation solutions estimated by the ZUPT-aided INS using the SHOE detector with a fixed threshold. The upper plot shown in Figure A.53 presents the estimated trajectory expressed in the global coordinate, and the bottom plot presents the navigation solution in the local NED frame. The accumulated position error at the end of the experiments was around 50 [m]. It is worth pointing out that, by visual inspection, the

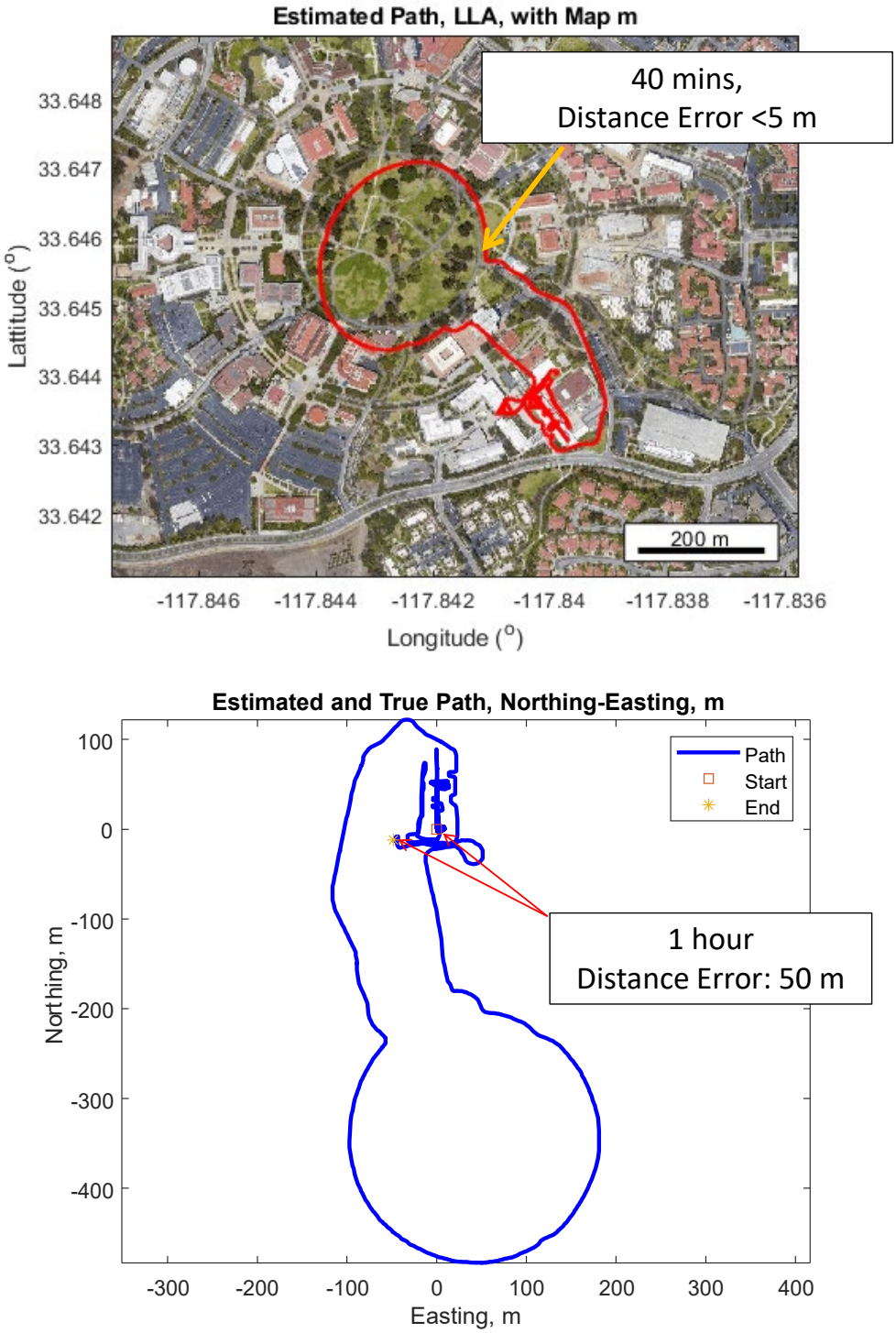


Figure A.53: Estimated trajectory of in reliability test for the Lab-On-Shoe 2.0 platform. The trajectory included indoor environments and outdoor environments. The total duration was around one hour, and the total path length was about 2 [km]. Loop-closure displacement error was 50 [m].

position error at time elapsing 40 minutes was around 5 [m]. This experiment demonstrated that the firmware and the Teensy 3.2 microcontroller could handle navigation tasks for more than one hour.

The hardware and firmware for the Lab-On-Shoe 2.0 navigation platform were developed. The Lab-On-Shoe 2.0 platform uses microcontrollers to communicate with IMUs, ultrasonic sensors, altimeters, and UWBs. Two tests were conducted to evaluate the accuracy and reliability of the Lab-On-Shoe 2.0 platform. The first test showed that, based on the data collected by the platform, the ZUPT-aided INS had an accumulated displacement error of 2 [m] after navigating in an indoor environment for 600 [m]. The second test demonstrated that the firmware and the microcontroller on the Lab-On-Shoe 2.0 system could handle navigation tasks for one hour.

Lab-On-Shoe 2.1: Compact PCB

In this section, we have upgraded the Lab-On-Shoe 2.0 system in two aspects. First, we redesigned the hardware implementation of the system to improve its robustness and compactness. Second, we modified the firmware of the system to add measurements of Ultra-WideBand (UWB) receiver power and transmitter power. The upgraded system is referred to as Lab-On-Shoe 2.1.

Hardware Architecture and Sensor Measurements The Lab-On-Shoe 2.1 system is shown in Figure A.54. The system includes two different pieces of hardware: a foot-mounted system and UWB beacon modules. In current implementation, the foot-mounted module is installed on the toe side of each foot, and the module includes an Inertial Measurement Unit (Analog Device ADIS16497−3), a barometric altimeter (MS5803−01BA), a UWB (Decawave DWM1000), multiple ultrasonic sensors (SRF02), and a microcontroller (Teensy 4.0). All the

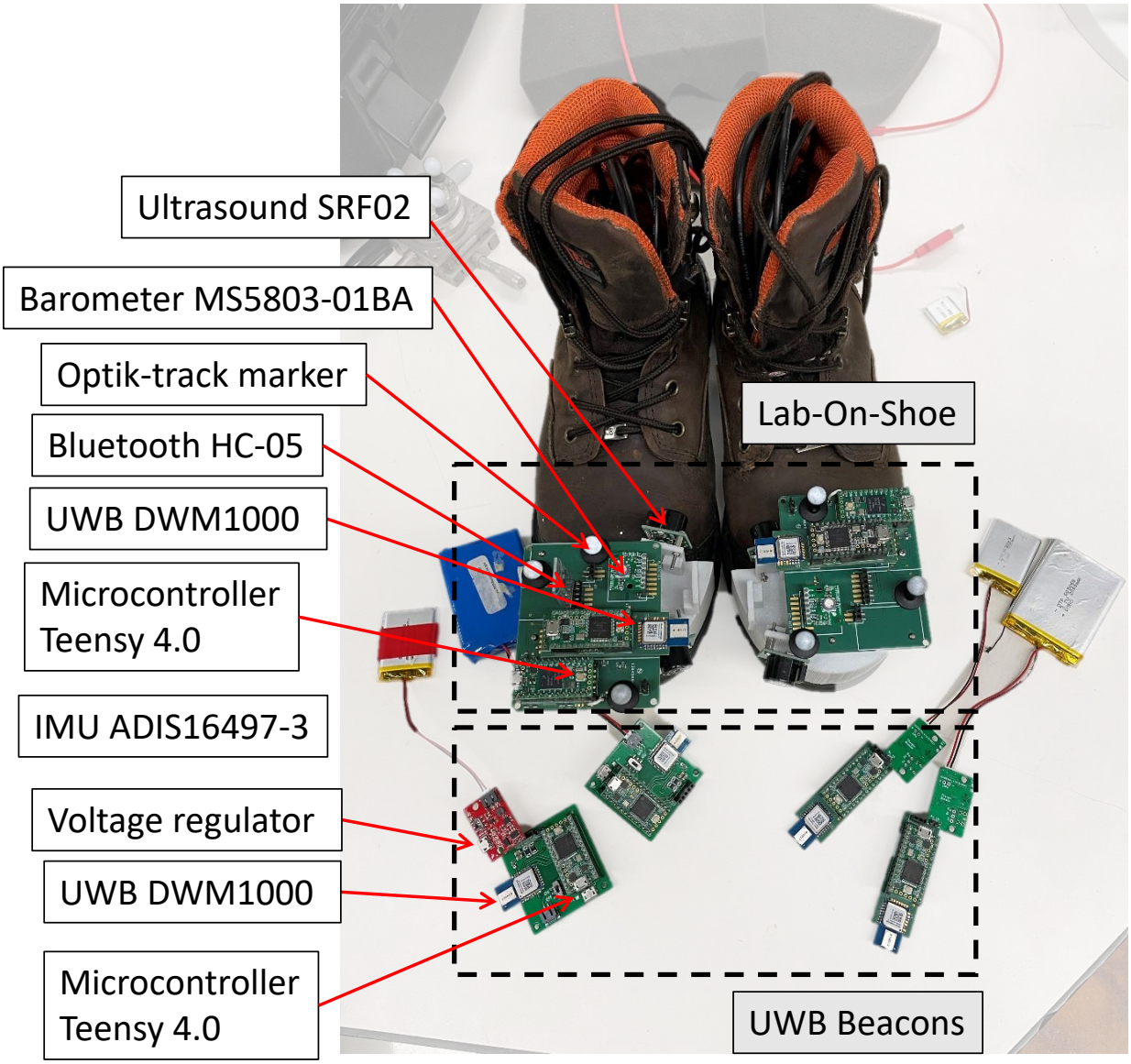


Figure A.54: The Lab-On-Shoe 2.1 system.

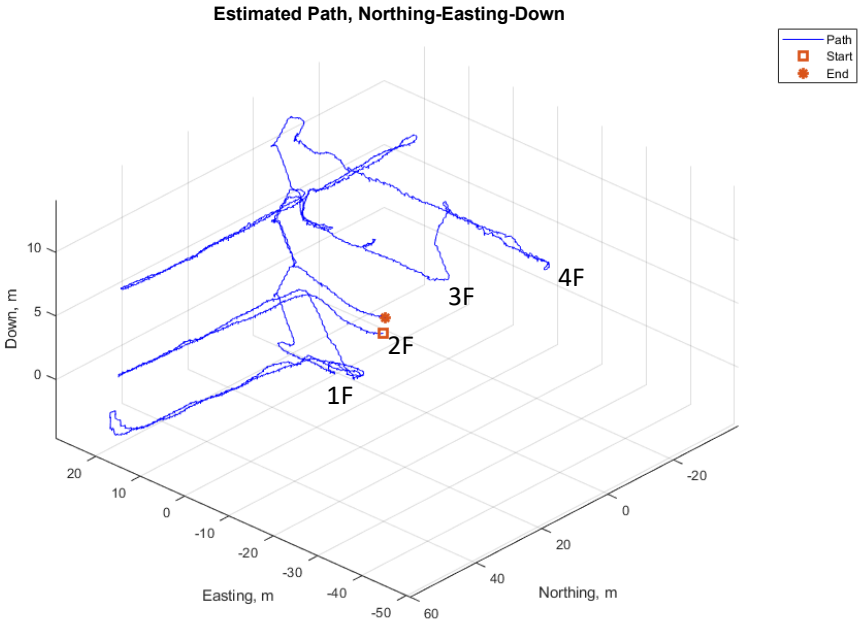
electronic devices, except for the ultrasonic sensors, of the foot-mounted modules are directly installed on a Printed Circuit Board (PCB). The sampling rate of each sensor is 1000 Hz for the IMU, 25 Hz for the barometer, 10 Hz for the UWB, and 50 Hz for the ultrasonic sensor. The UWB beacon module includes a UWB (DWM1000) and a microcontroller (Teensy 3.2). The UWB beacon modules are programmed to have two different types: tag and anchor. Distance measurements between a pair of a tag and an anchor are determined based on the two-way ranging approach. In the current setup, a tag can be connected to a maximum number of four anchors. All the UWB beacon modules are programmed as anchors, and the UWBs on the foot-mounted modules are programmed as tags.

Each foot-mounted module of the Lab-On-Shoe 2.1 provides 14 different measurements, including accelerometers along the three axes (g), gyroscopes along the three axes (degree per second), air pressure (mbar), temperature ($^{\circ}C$), ultrasonic range (cm), UWB range (m), UWB power metric (dBm), UWB receiver power (dBm), and UWB transmitter power (dBm).

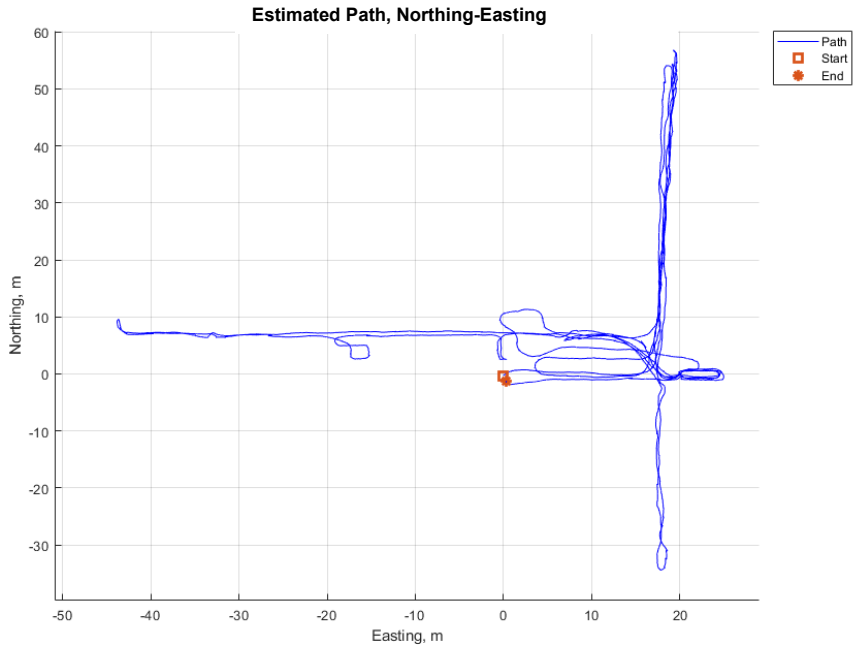
Testing To test the performance of the Lab-On-Shoe 2.1 system, we conducted two indoor navigation experiments.

The first experiment was conducted at the Engineering Gateway Building at the University Of California, Irvine. In this experiment, an agent equipped with the Lab-On-Shoe 2.1 system started on the second floor of the building. The agent then proceeded to walk for 912 [m] in 13.1 minutes in both indoor and outdoor environments and between four different floors. At the end of the experiment, the agent returned to the starting point. The trajectory included terrains of flat planes, stairs, ramps, and elevators. Sensors measurements of the Lab-On-Shoe 2.1 system were collected and saved on a laptop.

We used the ZUPT-aided INS augmented by altimeter measurements and foot-to-foot UWB



(a)



(b)

Figure A.55: Navigation results in the experiment conducted at the Engineering Gateway Building at the University of California, Irvine. The solution estimated by the ZUPT-aided INS augmented by altimeter measurements and foot-to-foot range measurements.

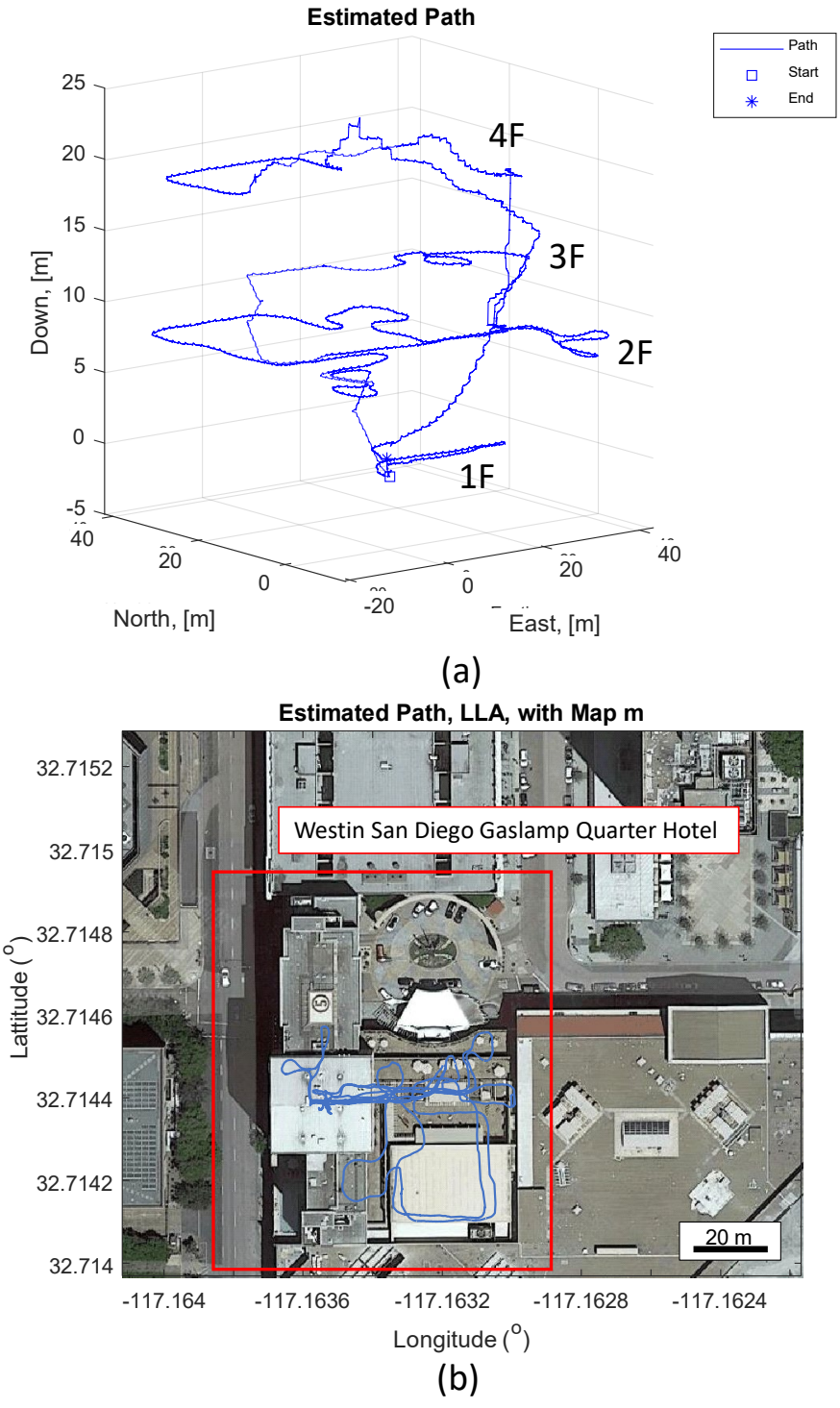


Figure A.56: Navigation results in the experiment conducted at the Westin San Diego Gaslamp Quarter Hotel. Navigation solution estimated by the ZUPT-aided INS augmented by altimeter measurements and foot-to-foot range measurements.

ranging measurements. Figure A.55(a) and (b) shows the 3D and horizontal navigation solution of the experiment. The loop-closure error of this experiment was 1.43 [m].

The second experiment was conducted at the Westin San Diego Gaslamp Quarter Hotel. In this experiment, the agent was equipped with the Lab-On-Shoe 2.1 system and started at the lobby on the 1F. Then, the agent walked inside the hotel for 1.2 [km] in 17 minutes between 1F to 4F. At the end of the experiment, the agent walked back to the original location. The trajectory included flat planes constructed with carpet and concrete, ramps, stairs, and elevators.

Navigation results of this experiment were produced by the ZUPT-aided INS augmented by altimeter measurements and foot-to-foot UWB ranging. Figure A.56(a) and (b) present the 3D and horizontal trajectory. The loop-closure error of this experiment was 1.3 meters.

Lab-On-Shoe 2.0 platform was upgraded to Lab-On-Shoe 2.1 platform. The upgraded Lab-On-Shoe system had a smaller physical size and a more reliable wire connection. We tested the system with two indoor navigation experiments at two different locations. The first experiment had a loop-closure error of 1.43 [m] after walking for 912 [m] in 13.1 minutes. The second experiment had a loop-closure error of 0.5 meters after navigating for 1.1 [km] in 17 minutes.

Lab-On-Shoe: C/C++ Implementation

```
1  ////////////////////////////////////////////////////////////////////////////////
2  //  April 2023
3  //  Author: Chi-Shih Jao <chishihj@uci.edu>
4  ////////////////////////////////////////////////////////////////////////////////
5  //  Lab_On_Shoe_2_0.ino
6  ////////////////////////////////////////////////////////////////////////////////
7  //
8  //  Pinout for a Teensy 3.2 Development Board
9  //  RST = D6
10 //  SCK = D13/SCK
11 //  CS = D10/CS
```

```
12 //  DOUT(MISO) = D12/MISO
13 //  DIN(MOSI) = D11/MOSI
14 //  DR = D2
15 //
16 ////////////////////////////////////////////////////////////////////////////////
17 #include <ADIS16490.h>
18 #include <SPI.h>
19 #include <Wire.h>
20 #include <SRF08.h>
21
22 // Temporary Data Arrays
23 int16_t* sensorData;
24 float scaledData[7];
25
26 // Control registers
27 int MSC = 0;
28 int DECR = 0;
29
30 unsigned long StartTime = millis();
31 unsigned long PrevTime = StartTime;
32 unsigned long SONARPrevTime = StartTime;
33 unsigned long CurrentTime = StartTime;
34 double Period, fs, ElapsedTime = CurrentTime - StartTime, SONAR_fs = 200,
35 SONARElapsedTime = CurrentTime - StartTime;
36 double SONAR_Period = 1/SONAR_fs*1000;
37
38 // Delay counter variable
39 int printCounter = 0;
40
41 // Call ADIS16490 Class
42 ADIS16490 IMU(10,2,6); // Chip Select, Data Ready, Reset Pin Assignments
43 SRF08 SONAR(117); // Chip Select, Data Ready, Reset Pin Assignments
44 int range_reading = 0;
45
46 void setup()
47 {
48     Serial.begin(115200); // Initialize serial output via USB
49     IMU.configSPI(); // Configure SPI communication
50     delay(1000); // Give the part time to start up
51     IMU.regWrite(FNCTIO_CTRL, 0x0C); // Enable Data Ready, set polarity
52     delay(20);
53     IMU.regWrite(DEC_RATE, 0x00), // Disable decimation
54     delay(20);
55
56     // Read the control registers once to print to screen
57     MSC = IMU.regRead(FNCTIO_CTRL);
58     DECR = IMU.regRead(DEC_RATE);
```

```
59
60      // Configure SPI settings for IMU
61      IMU.configSPI();
62
63      // Attach interrupt to pin 2. Trigger on the rising edge
64      attachInterrupt(2, grabIMUData, RISING);
65      SONAR.configSensor();
66  }
67
68  // Function used to read register values when an ISR is triggered using the IMU's
69  DataReady output
70  void grabIMUData()
71  {
72      sensorData = IMU.sensorRead();
73  }
74
75  // Function used to scale all acquired data (scaling functions are included in
76  ADIS16490.cpp)
77  void scaleIMUData()
78  {
79      scaledData[0] = IMU.gyroScale(sensorData[2]); // XGYRO
80      scaledData[1] = IMU.gyroScale(sensorData[3]); // YGYRO
81      scaledData[2] = IMU.gyroScale(sensorData[4]); // ZGYRO
82      scaledData[3] = IMU.accelScale(sensorData[5]); // XACCL
83      scaledData[4] = IMU.accelScale(sensorData[6]); // YACCL
84      scaledData[5] = IMU.accelScale(sensorData[7]); // ZACCL
85      scaledData[6] = IMU.tempScale(sensorData[8]); // TEMP
86  }
87
88  // Main loop. Print data to the serial port. Sensor sampling is performed in the
        ISR
89  void loop()
90  {
91      printCounter ++;
92      if (printCounter >= 1) // Delay for writing data to the serial port
93      {
94          PrevTime = CurrentTime;
95          CurrentTime = millis();
96          ElapsedTime = ((float) CurrentTime - (float) StartTime)/1000;
97          Period = ((float) CurrentTime - (float) PrevTime);
98          fs = 1/Period*1000;
99          SONARElapsedTime = CurrentTime - SONARPrevTime;
100
101         scaleIMUData(); // Scale data acquired from the IMU
102
103         // Print Status Registers
104         Serial.print("DIAG_STS: ");
```

```
105         Serial.println(sensorData[0]);
106         Serial.print("DATA_CNT: ");
107         Serial.println(sensorData[1]);
108 
109         // Print scaled temp data
110         Serial.print("TEMP: ");
111         Serial.println(scaledData[6]);
112 
113         // Print scaled gyro data
114         Serial.print("XGYRO: ");
115         Serial.println(scaledData[0]);
116         Serial.print("YGYRO: ");
117         Serial.println(scaledData[1]);
118         Serial.print("ZGYRO: ");
119         Serial.println(scaledData[2]);
120 
121         // Print scaled accel data
122         Serial.print("XACCL: ");
123         Serial.println(scaledData[3]);
124         Serial.print("YACCL: ");
125         Serial.println(scaledData[4]);
126         Serial.print("ZACCL: ");
127         Serial.println(scaledData[5]);
128         //Serial.println(" ");
129 
130         Serial.print("Range: ");
131         Serial.println(range_reading);
132 
133         Serial.print("Sampling Rate: ");
134         Serial.println(fs);
135 
136         Serial.print("Period: ");
137         Serial.println(Period);
138 
139         Serial.print("ElapsedTime: ");
140         Serial.println(ElapsedTime);
141 
142         Serial.print("\n");
143         //Serial.println("Status Registers");
144 
145         if (SONARElapsedTime > SONAR_Period) {
146           range_reading = SONAR.regRead();
147           SONAR.regWrite();
148           SONARPrevTime = CurrentTime;
149           Serial.print("SONAR Sampling Rate: ");
150           Serial.println((float) 1/SONARElapsedTime*1000);
151         }
```

```
152
153 #ifdef DEBUG
154         // Print unscaled gyro data
155         Serial.print("XGYRO: ");
156         Serial.println(sensorData[2]);
157         Serial.print("YGYRO: ");
158         Serial.println(sensorData[3]);
159         Serial.print("ZGYRO: ");
160         Serial.println(sensorData[4]);
161
162         // Print unscaled accel data
163         Serial.print("XACCL: ");
164         Serial.println(sensorData[5]);
165         Serial.print("YACCL: ");
166         Serial.println(sensorData[6]);
167         Serial.print("ZACCL: ");
168         Serial.println(sensorData[7]);
169         Serial.println(" ");
170
171         // Print unscaled temp data
172         Serial.print("TEMP: ");
173         Serial.println(sensorData[8]);
174 #endif
175         printCounter = 0;
176         grabIMUData();
177         //attachInterrupt(2, grabIMUData, RISING);
178     }
179 }
```

A.2 Sugar-Cube Platform

This section of the appendix presents the development of the Sugar-Cube platforms. The platforms were designed to be miniaturized versions of pedestrian navigation testbeds presented in Section A.1 with selected electronic components and real-time navigation capabilities. The rest of this section is organized as follows. Section A.2.1 presents the Sugar-Cube 1.0 platform, Section A.2.2 discusses the Sugar-Cube 2.0 platform, Section A.2.3 describes a developed Android App used to visualized real-time localization solutions of the Sugar-Cube 2.0 platform, Section A.2.4 presents a demonstration procedure and a demonstration inter-

face for the platform, and Section A.2.5 provides C/C++ source codes implemented on the microcontroller of the Sugar-Cube platforms.

A.2.1 Sugar-Cube 1.0: Real-Time Navigation Platform

This section presents the development of the Sugar-Cube navigation platform in firmware and hardware levels. The developed Sugar-Cube navigation platform was used to conduct a real-time localization task, and the experimental results are included in this section.

Hardware of Sugar-Cube Navigation Platform

The current version of the Sugar-Cube platform uses a micro-controller, Teensy 4.0, a 9-axis IMU Invensense ICM−20948, a Bluetooth module HC−05, a voltage booster module, and a lithium battery. Figure A.57(b) shows a prototype of the Sugar-Cube platform under development, and Figure A.57(a) presents a Printed Circuit Board (PCB) realization of the Sugar-Cube platform mounted on a shoe. The PCB shown in Figure A.57(a) was used for reliable wire connection and circuit rerouting. The Teensy 4.0 has a clock rate of 600 MHz. The ICM−20948 is a consumer-grade Inertial Measurement Unit (IMU) that contains a 3-axis accelerometer, a 3-axis gyroscope, and a 3-axis magnetometer. The lithium battery shown in both Figure A.57(a) and Figure A.57(b) can support the Sugar-Cube platform to run more than one hour when fully charged.

Firmware development

Current firmware implementation of the Sugar-Cube platform enables real-time localization. The firmware includes two parts. The first part performs sensor measurement acquisitions, and the second part estimates real-time navigation states based on the collected sensor mea-

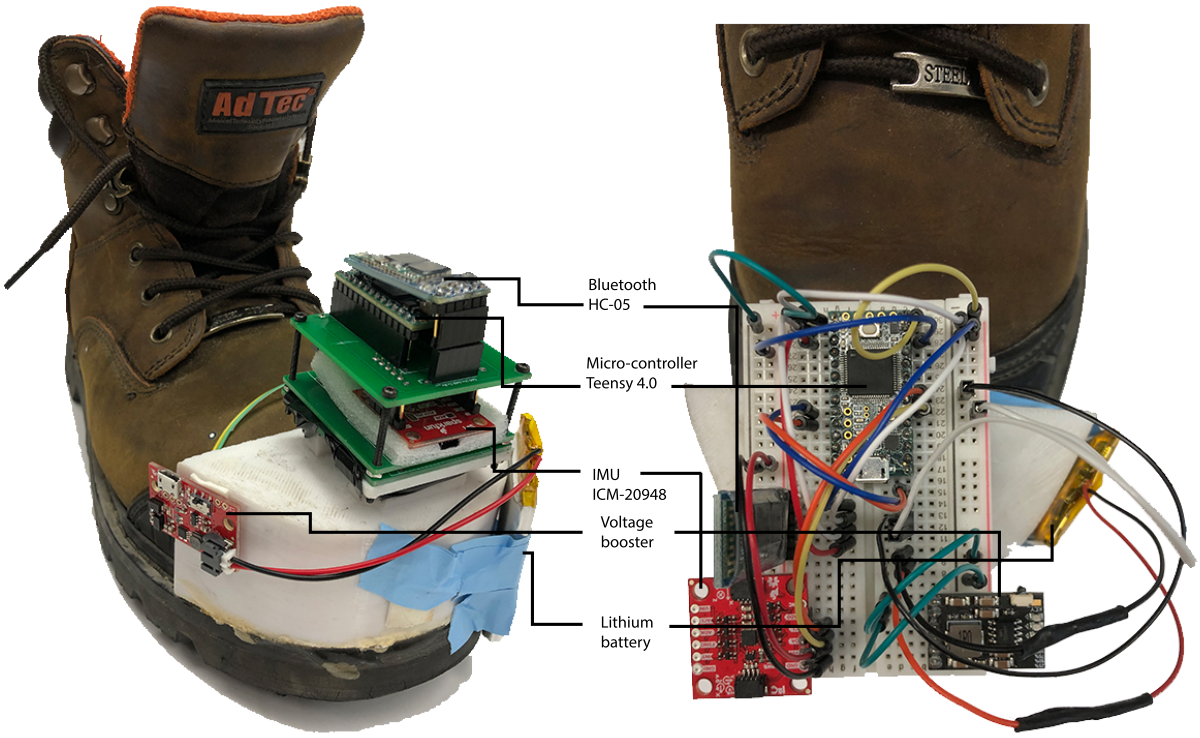


Figure A.57

surements. Figure A.58 presents a block diagram of the firmware. In the sensor acquisition module, the Teensy 4.0 communicates with the IMU and the magnetometer via I^2C protocol. The sampling rates of the IMU and the magnetometer are set at 350 Hz and 50 Hz, respectively.

After the system boots up, the sensor acquisition module starts to collect sensor measurements and send the measurements to the navigation solutions module in the firmware. In the first ten seconds of operation, the navigation solutions module is in an initialization stage, where the algorithm assumes that the Sugar-Cube platform is stationary. In this stage, accelerometer measurements are used to initialize roll and pitch angles, and magnetometers are utilized for yaw angles. After initiation, the navigation solutions module calculates current navigation states using a Zero velocity UPdaTe (ZUPT)-aided INS algorithm with the Stance Hypothesis Optimal dEtection (SHOE) detector and produces the information at a rate of

350 Hz once sensor measurements are received. Then, the current navigation solution is sent wireless to a remote display via Bluetooth using the UART communication protocol. The Baud rate for the Bluetooth UART is set to 115200 Hz. In a single transmission, navigation information is stacked and sent in a vector $\mathbf{x}$, expressed as follows:

$$\mathbf{x} = [\mathrm{t}, \mathbf{a}, \mathbf{g}, \mathbf{q}, \mathbf{v}, \mathbf{p}, \mathrm{T}, \Phi],$$

where t is current timestamp on a micro-controller, $\mathbf{a}$ is 3-axis accelerometer's measurements, $\mathbf{g}$ is 3-axis gyroscope's measurements, $\mathbf{q}$ is orientation vector, containing roll, pitch, and yaw, $\mathbf{v}$ is velocity vector, $\mathbf{p}$ is position vector, T is stance phase status, and Φ is stance phase detection statistics. It is worth mentioning that the navigation state information is sent at a rate of 10 Hz to a remote display in order to achieve reliable transmission via Bluetooth.

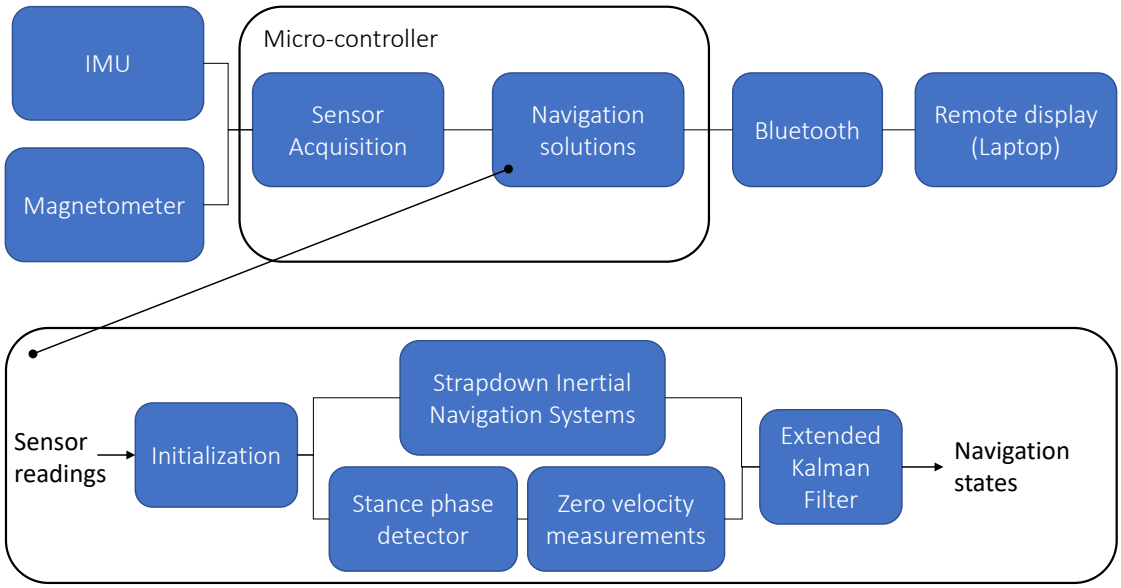
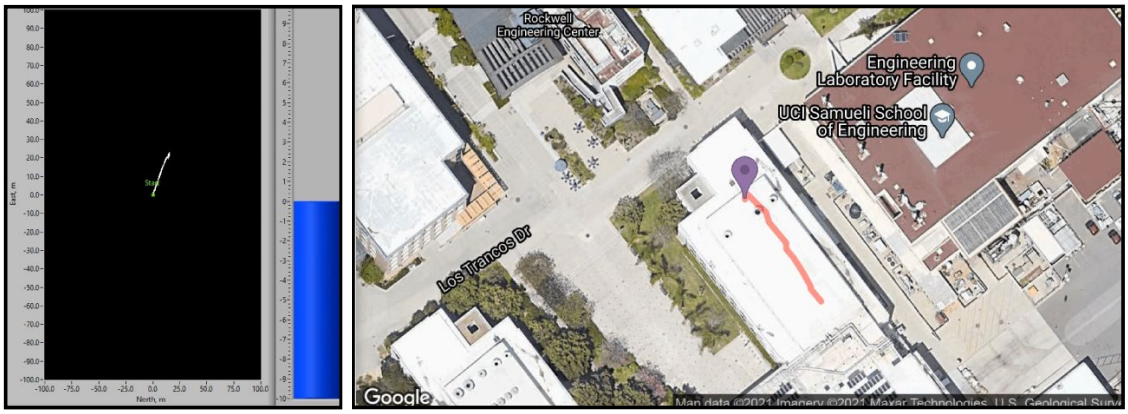


Figure A.58: A block diagram describing the firmware of the Sugar-Cube navigation platform.

Remote display Interface

Real-time localization solution estimated by the Sugar-Cube navigation platform is sent to a remote device via Bluetooth for display. A device, such as a desktop, a laptop, a tablet, or a smartphone, can be used to pair with the Sugar-Cube platform as long as it has a Bluetooth module. In our current implementation, a laptop is used for this purpose, and a LabVIEW User Interface (UI) was designed to facilitate data visualization.

Figure A.59 demonstrates the LabVIEW UI. The UI provide real-time positions in two different coordinates. The first is local positions along the north, the east, and the down directions. The second is the global coordinate in Longitude, Latitude, and Altitude (LLA). The global positions of the Sugar-Cube platform are initialized at N33.6436^o, W117.8402^o, and 41 [m] above sea level, which is the coordinate of the second floor of Engineering Gateway Building at the University of California, Irvine. The global positions are displayed on a real-time satellite map downloaded from Google Static Map API. The center of the map is the current LLA provided by the Sugar-Cube platform. The map is updated every two seconds. The red curve in Figure A.59(b) indicates the trajectory of the user.

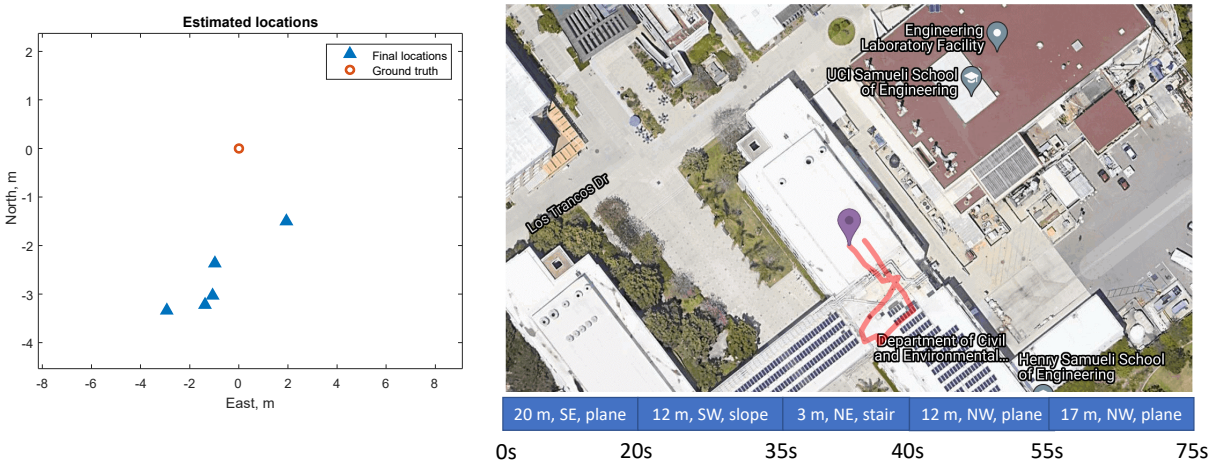


(a) Local position (b) Global position

Figure A.59: An UI developed in LabVIEW for visualizing navigation solutions estimated by the Sugar-Cube navigation platform.

Demonstration of Real-time Localization Task

To test the localization accuracy of the developed Sugar-Cube navigation platform, we conducted five indoor walking experiments. In each experiment, the Sugar-Cube platform with a PCB was mounted on the shoe of a pedestrian, and the pedestrian walked a closed-loop trajectory. An example of the trajectory is shown and described in Figure A.60(b). Figure A.60(a) demonstrates estimated destinations of the five experiments. The position Mean Absolute Error (MAE) of the experiments was 3.2 meters. The experimental results showed that the Sugar-Cube navigation platform provided real-time solutions with the same level of accuracy as post-processing.



(a) Estimated final locations

(b) An example of the experiment trajectory

Figure A.60

This section presents documentation of the hardware and firmware of the Sugar-Cube navigation platform. The implementation of the Sugar-Cube platform was achieved in real-time, allowing for estimation of the position of a user using the ZUPT-aided INS in local NED frame and global LLA frame with an assumption that initial global coordinate is available. A series of indoor navigation experiments were conducted to test the Sugar-Cube platform. The position MAE in the 75-second-long experiments was 3.2 meters.

A.2.2 Sugar-Cube 2.0: Real-Time Sensor-Fusion-aided Navigation

This section discusses the documentation of Sugar-Cube 2.0 navigation platform. Sugar-Cube 2.0 platform included a micro-controller, Teensy 4.0, an Inertial Measurement Units (IMU), a barometer, and an ultrasonic sensor. The platform produced localization of a user in real-time based on the altimeter-enhanced ZUPT-aided INS using the UA-SHOE detector. This version of the platform provides a flexible plug-and-play architecture for hardware and software development.

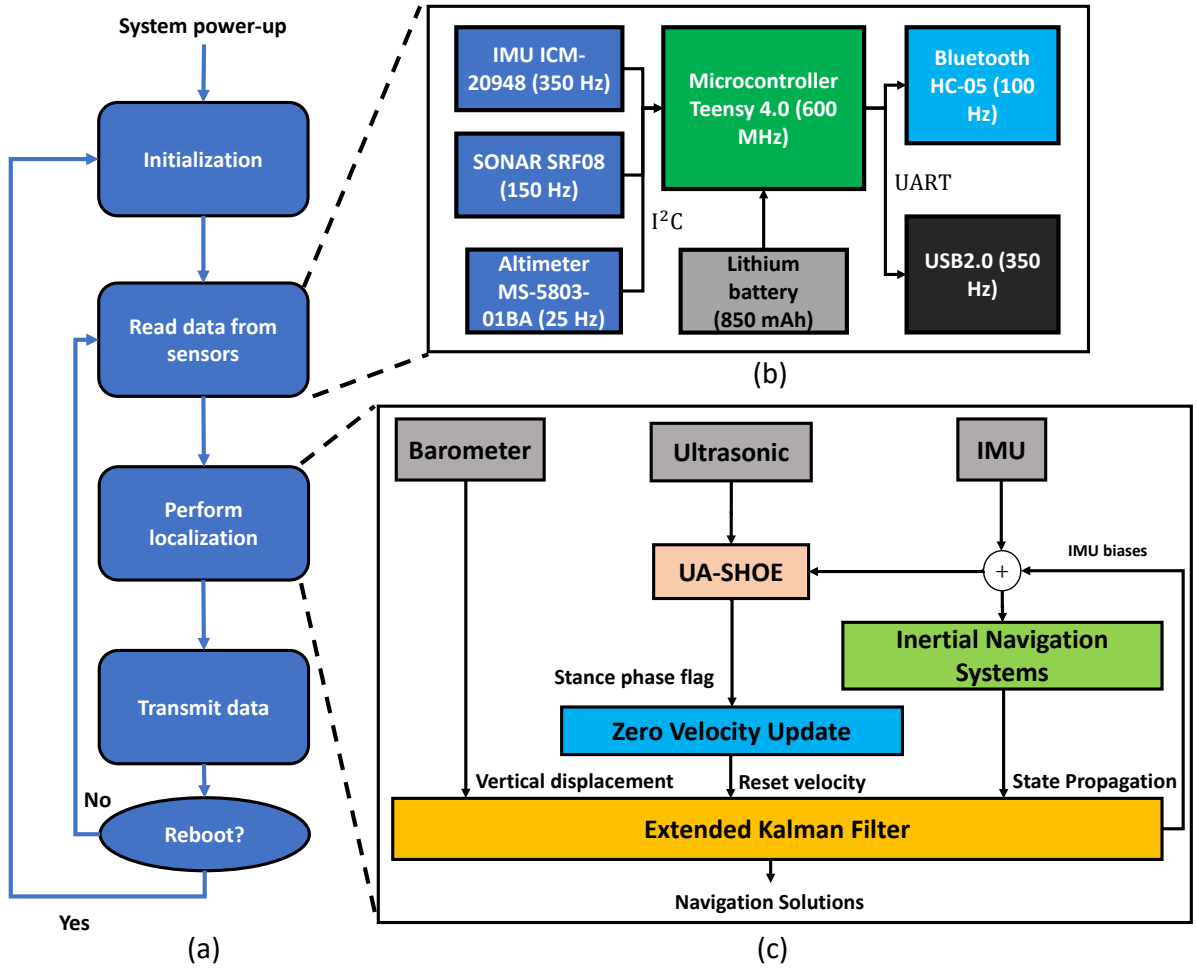


Figure A.61: (a) The runtime framework of the Sugar-Cube platform. (b) Sensor connection and communication mechanism on the Sugar-Cube platform. (c) Navigation algorithm implemented on the on-board micro-controller.

Navigation Algorithm

The real-time navigation solution implemented on the current Sugar-Cube platform is the ZUPT-aided INS enhanced by a barometric altimeter with the UA-SHOE detector. Figure A.61(c) exhibits a block diagram of the algorithm, which is realized in an Extended Kalman Filter (EKF) framework. The stance phase detector used in this system is the UA-SHOE detector. The stance phase detector used in the navigation algorithm has the following form.

$$T_h(z_n) = \frac{1}{N}\sum_{k\in\Omega_n}(\frac{1}{\sigma_\alpha^2}\parallel y_k^\alpha - g\frac{\bar{y}_k^\alpha}{\parallel \bar{y}_k^\alpha \parallel}\parallel^2 + \frac{1}{\sigma_\omega^2}\parallel y_k^\omega \parallel^2 + \frac{1}{\sigma_h^2}\parallel y_k^h - h \parallel^2) < \gamma_h,$$

where $z_n = \{y_k\}_{k=n}^{k=N-1}$, y_k^α is accelerometer measurements at time step k, y_k^ω is gyroscope measurements at time step k, y_k^h is ranging measurements provided by the downward-facing ultrasonic sensor at time step k, σ_a is Velocity Random Walk (VRW), σ_g is Angular Random Walk (ARW), σ_h is resolution of the ultrasonic sensor, h is the height of the ultrasonic sensors above the ground when the shoe is in contact, $\Omega_n = \{l \in \mathbb{N}, n \leq l < N-1\}$ is a collection of the sensor measurement indexes at time n with a window of length N, and γ_h are user-defined thresholds. Table A.14 lists values of EKF parameters, including ARW σ_g, VRW σ_a, Rate Random Walk (ARW) σ_r, Acceleration Random Walk (AcRW) σ_{Ac}, and threshold γ_h used for the UA-SHOE.

Table A.14: Parameters for the EKF

Hyper-parameter	**Value**
σ_g	2.7221×10^{-5}
σ_a	0.0017
σ_r	8.3174×10^{-7}
σ_{Ac}	6.63×10^{-6}
γ_h	e^{-12}

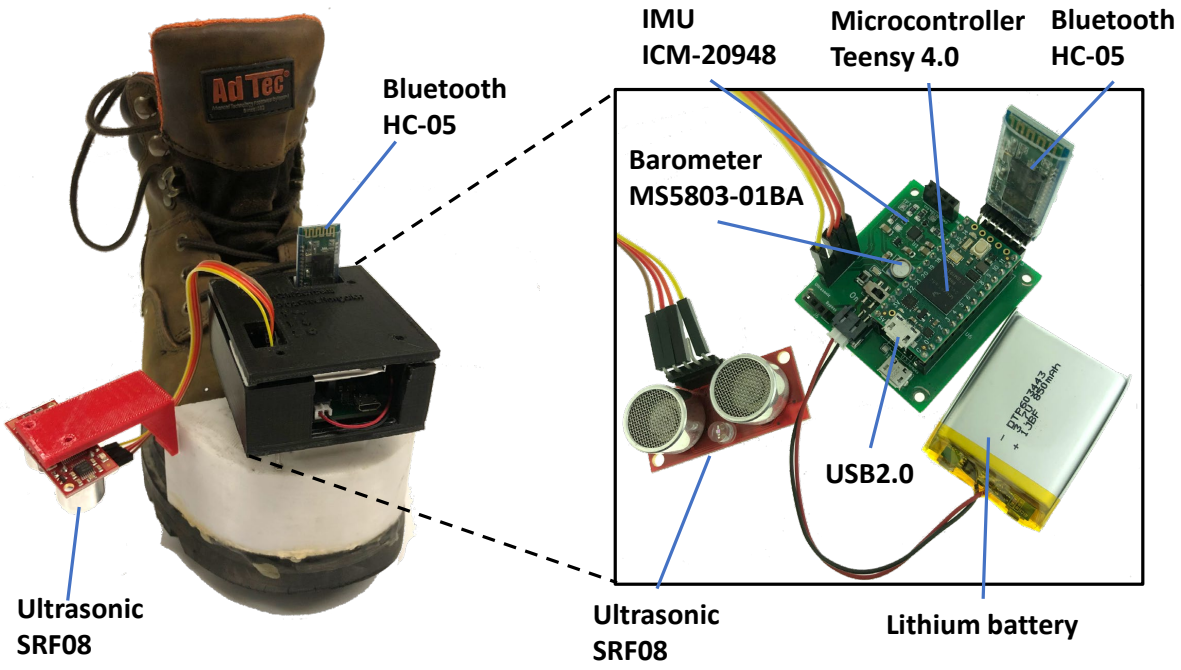


Figure A.62: Hardware of the Sugar-Cube navigation platform.

On-board sensors

A hardware prototype of the Sugar-Cube navigation platform is illustrated in Figure A.62. The black box in Figure A.62 is a fixture for placing customized Printed Circuit Boards (PCBs). A micro-controller Teensy 4.0, a consumer-grade 9-axis IMU ICM−20948, a barometric altimeter MS5803−01BA, and a Bluetooth module HC−05 were located on the PCB. A downward-facing ultrasonic sensor SRF08 was firmly attached at the backside of the red extended fixture arm shown in Figure A.62. The nominal distance between the ultrasonic sensor and the ground was 9 cm. In this configuration, the Sugar-Cube platform was mounted at the toe side, but it can be detached and moved to other locations, such as the heel side. The power source of the Sugar-Cube platform was an 850 mAh lithium battery with a 3.7 v output. The battery was connected to a voltage booster module located on the PCB to bring the voltage up to 5 v.

Firmware architecture

A block diagram illustrating sensor connection and communication protocols on the Sugar-Cube platform is shown in Figure A.61(b). The processing unit of the platform is the microcontroller Teensy 4.0, which has a nominal clock rate of 600 MHz and can be boosted up to 1.008 GHz. In the current implementation, the Sugar-Cube platform was programmed with language C/C++ through the Teensyduino library in the Arduino Integrated Development Environment (IDE). The on-board IMU, ultrasonic sensor, and barometer were communicated in I^2C and had a sampling rate of 350 Hz, 150 Hz, and 25 Hz, respectively. Information, including orientation, velocity, position, zero velocity states, and sensor readings, can be selectively transmitted to a remote device, such as a smartphone or a computer, via Bluetooth and USB2.0 port with up to 100 Hz and 350 Hz transmission frequency, respectively.

A flow chart describing a runtime framework of the Sugar-Cube platform is presented in Figure A.61(a). In the initialization process, the Sugar-Cube platform is assumed completely stationary for around 10 s. During this process, all communication protocols are initiated, accelerometer biases are estimated by implementing the ZUPT algorithm, gyroscope biases are calculated by taking the average of the measurements, and initial heading angle of the system is calculated based on the on-board magnetometer. Then, the Sugar-Cube platform enters a sensor data acquisition process. The obtained sensor readings are accessed by the localization module, which produces locations of a user based on the ZUPT-aided INS enhanced by a barometer with the UA-SHOE detector at a rate of 350 Hz. At the end of each iteration, the system waits for user's command to determine whether to continue the next iteration or to re-initiate the entire system.

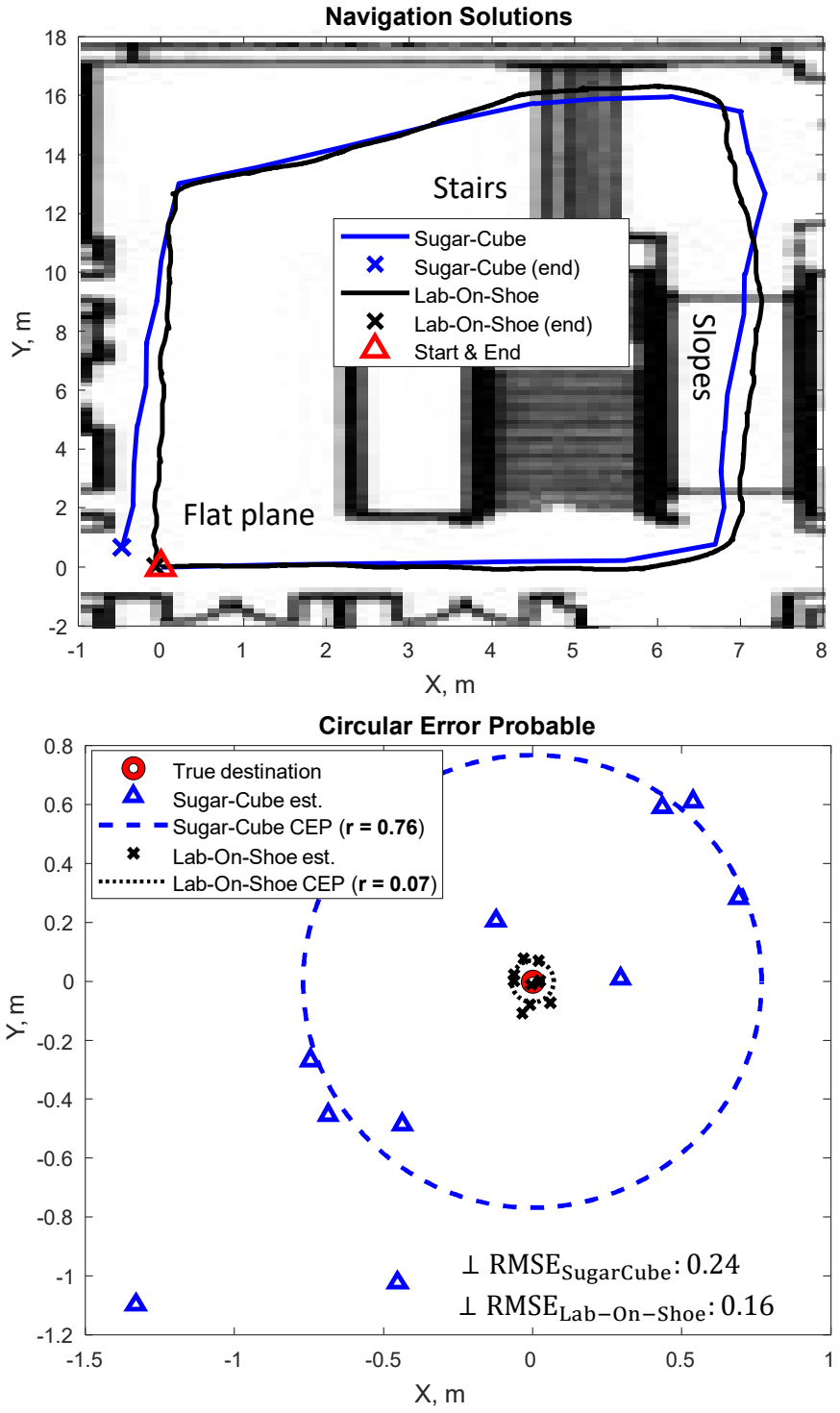


Figure A.63: The upper plot shows an example of the trajectories estimated by the Sugar-Cube platform and Lab-On-Shoe platform, respectively. The bottom plot shows estimated destination, CEPs, and RMSEs in the ten experiments discussed in Section A.2.2.

Performance Evaluation

To evaluate the navigation performance of the developed Sugar-Cube platform, we conducted ten sets of indoor navigation experiments in the Engineering Gateway Building at the University of California, Irvine. In the experiments, the Sugar-Cube navigation platform was mounted on a pedestrian's right foot, and a Lab-On-Shoe platform was installed on the pedestrian's left shoe. In this section, the Lab-On-Shoe 2.0 system was used as a benchmark provided by the navigation accuracy. In each experiment, the pedestrian started from a marker placed on the ground, walked a closed-loop trajectory at a speed of 60 step/min on terrains of flat planes, slopes, and stairs, and returned to the marker at the end of the experiment. The duration of each of the experiments was 1 min, and the length of the trajectories was 50 [m].

Figure A.63 presents the experimental results. The Sugar-Cube platform estimated navigation solutions in real-time, and the solutions provided by the Lab-On-Shoe were calculated in a post-processing manner using the same algorithm implemented on the Sugar-Cube platform. The upper plot in Figure A.63 displays an example of the trajectories estimated from the two systems overlapped on a floor plan of the building. We could see that the real-time solution obtained from the Sugar-Cube platform was able to depict the movement of the pedestrian. The bottom plot presents CEPs, which are circles enclosing 50% of the estimated horizontal destinations in the ten experiments, and RMSEs of the estimated vertical displacements. The CEP's Radius was 0.76 [m] in the case of the Sugar-Cube platform and was 0.07 [m] in the case of the Lab-On-Shoe platform. The vertical RMSEs of the Sugar-Cube and the Lab-On-Shoe platform were 0.24 [m] and 0.16 [m], respectively. In our opinion, the Lab-On-Shoe platform had a higher accuracy mainly because the current configuration of the Sugar-Cube platform used a consumer-grade IMU while the Lab-On-Shoe platform adopted a near-tactical-grade IMU. Based on these experiments, we concluded that the Sugar-Cube platform could collect measurements from an IMU, a barometer, and a downward-facing

ultrasonic sensor and perform real-time localization of a pedestrian with respect to an initial location.

This section presents the Sugar-Cube navigation platform, which was capable of performing real-time localization of a pedestrian based on a ZUPT-aided INS enhanced by an altimeter with the UA-SHOE detector. A series of indoor walking experiments of 60 [s] were conducted, and the experimental results demonstrated that the real-time navigation results of the Sugar-Cube platform had a horizontal CEP of 0.76 [m] and a vertical RMSE of 0.24 [m].

A.2.3 Sugar-Cube: Android User Interface

This section presents a developed Android application to display real-time location information produced by the Sugar-Cube platform. Workflow and User Interface (UI) of the application are discussed in this section..

The Android application, shown in Figure A.64, was developed in Android Studio Integrated Development Environment (IDE) and was designed to be paired to a Sugar-Cube platform via Bluetooth connection. The Android application has a UI shown in Figure A.66. The application and firmware implemented on a Sugar-Cube platform have integrated workflow presented in Figure A.65.

After a sugar-Cube platform is powered up and the Android application is implemented, the Sugar-Cube platform waits for an initial to be sent from the Android application. The user can click the start button on the UI to send the initial command to the module, and the module enters the initialization process. The initialization process takes five seconds. Then the module starts collecting sensor measurements from the onboard IMU and barometer. Then, the module calculates navigation solutions and transmits the solutions to the Android application. The solutions are then displayed on the UI. At this point, if the user does

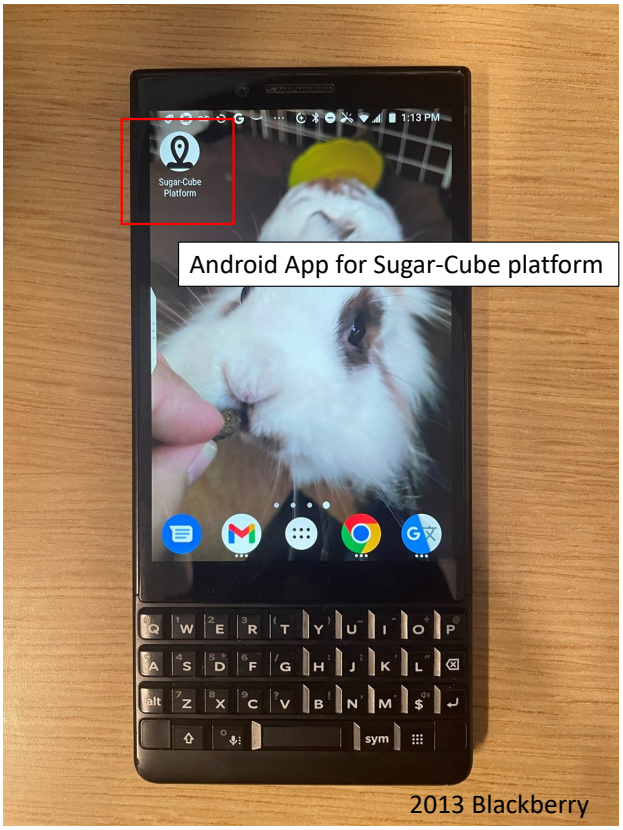


Figure A.64: Sugar Cube Android App Icon

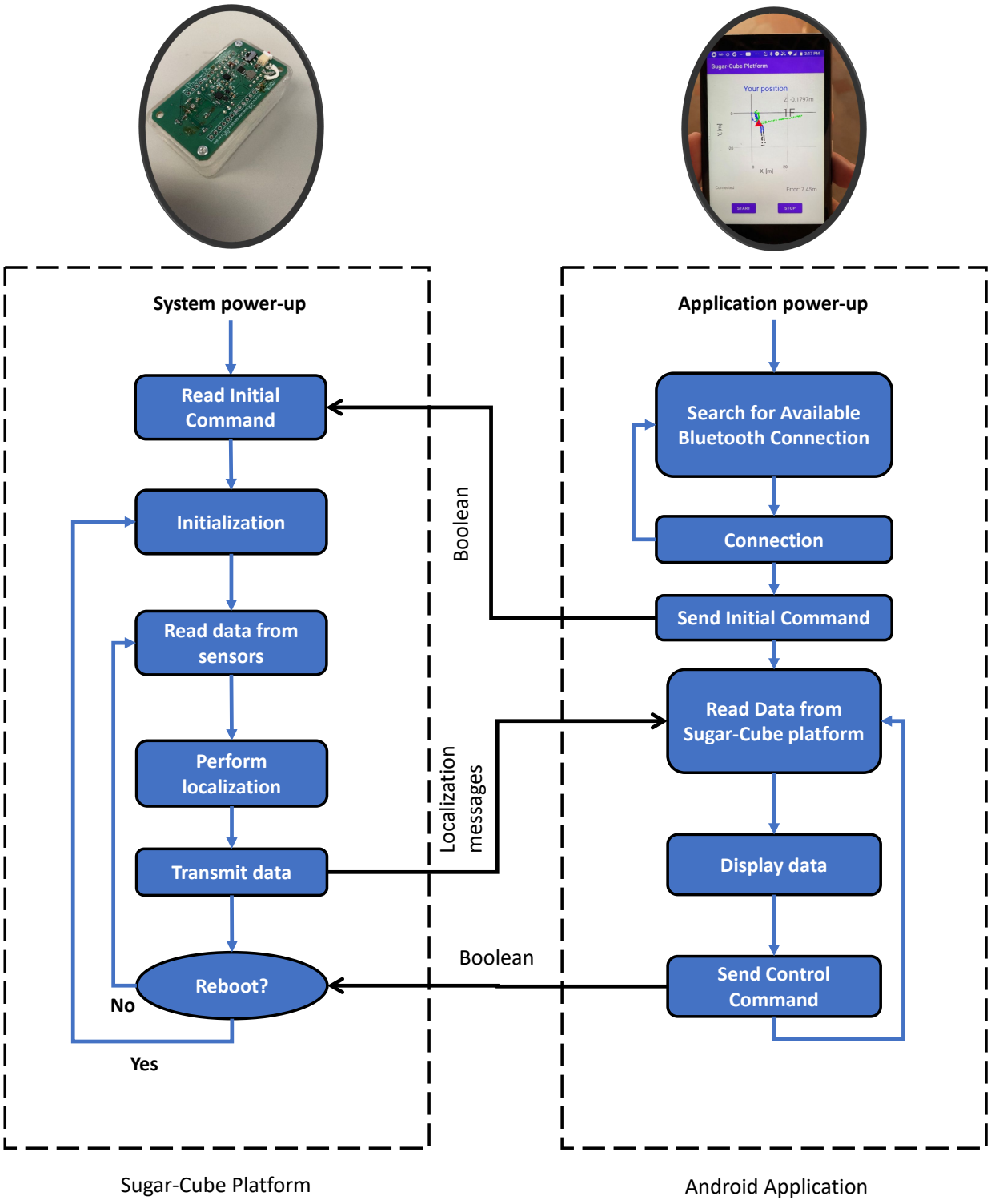


Figure A.65: Sugar-Cube Android application workflow.

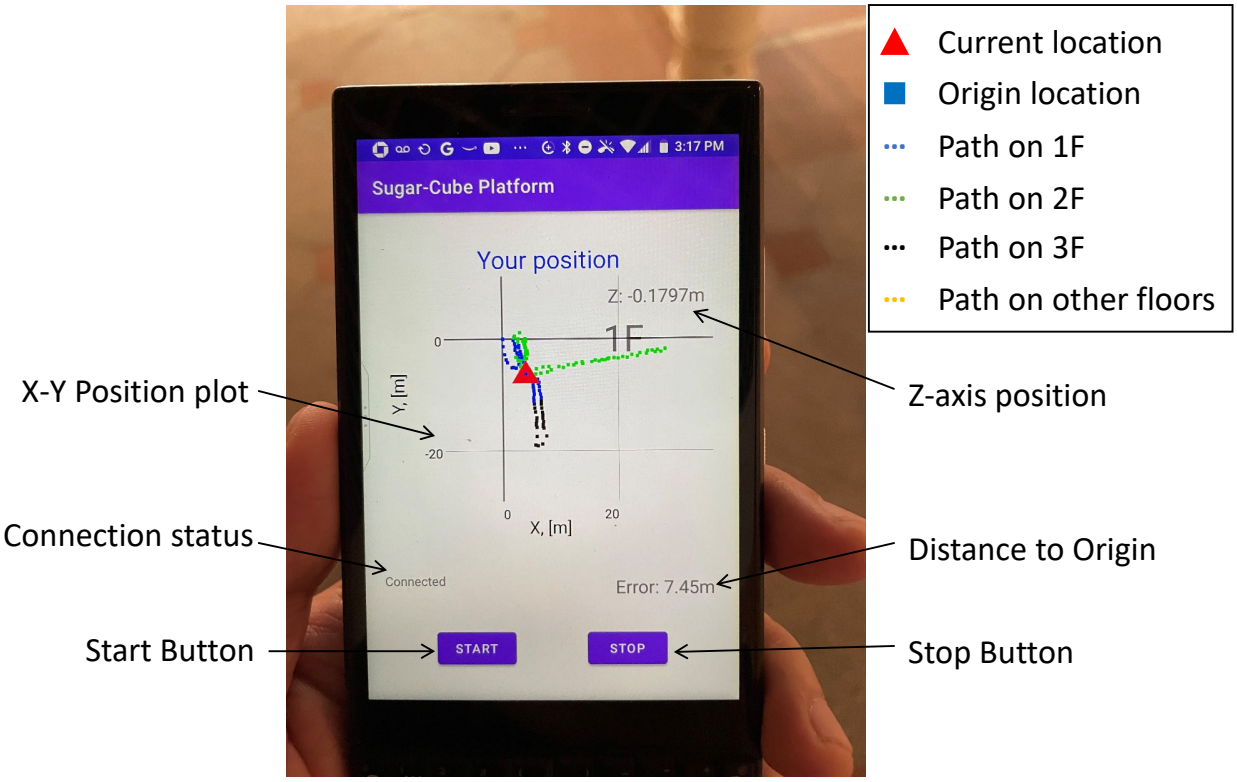


Figure A.66: Sugar-Cube Android application UI.

not do anything on the UI, the sugar-cube module would continue transmitting real-time localization messages to the Android application, and the display on the UI would be updated based on the received data. Alternatively, the user can click the stop button on the UI to stop the Bluetooth connection, and the sugar-cube platform would be re-initialized, waiting for the next navigation task.

A.2.4 Sugar-Cube: A Demonstration Process

A real-time demonstration of the Sugar-Cube platform was conducted for the 2022 PSCR conference. This section discusses a demonstration process and the results collected during the three days of the conference.

Demonstration Process

Demonstration of the Sugar-Cube platform is based on Sugar-Cube V7, which integrated a micro-controller Teensy 4.0 clocked at 600 Mhz, an IMU ICM−42605, a barometer ICP-20100, a Bluetooth module HC−05, and a lithium-ion battery. On the micron-controller, we implemented a ZUPT-aided INS enhanced with barometer measurements. Estimated positioning solutions were transmitted to a laptop that ran a MATLAB program overlapping the positions with a LiDAR point cloud map of the conference room. The program saved all the received positions.

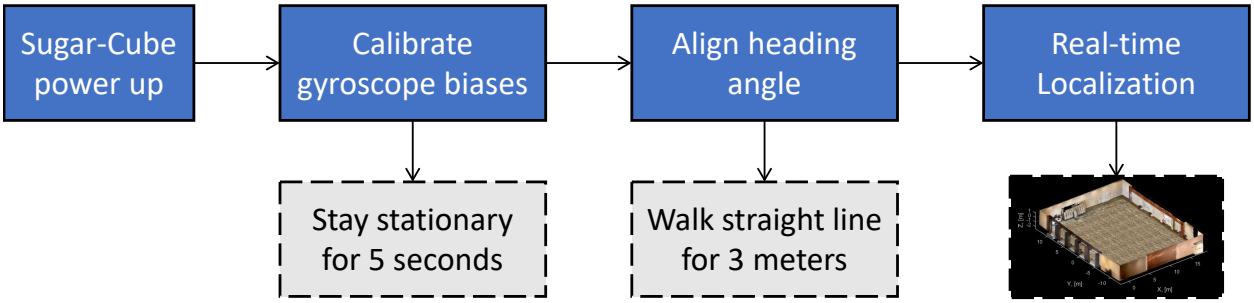


Figure A.67: Sugar-Cube platform demonstration process in the 2022 PSCR conference.

Figure A.67 presents a demonstration process of the Sugar-Cube platform at the 2022 PSCR conference. In each demon run, a Sugar-Cube platform was attached to the toe side of a demo participant with double-sided tape and powered up. The demo participant was asked to remain stationary for 5 seconds and walk a straight line for 3 meters. Gyroscope and accelerometer biases were in the stationary period, and we visually aligned the heading angles of the Sugar-Cube platform with the conference point cloud map. Once the heading angles were determined, all position solutions received on the laptop were multiplied with a corresponding rotation matrix. Then, the participant could casually walk around and eventually return to the starting point to evaluate position errors.

During the three days-session of the conference, 30 different demo trials were conducted. Figure A.68(a) shows the trajectories estimated by the Sugar-Cube platform. Figure A.68(b)

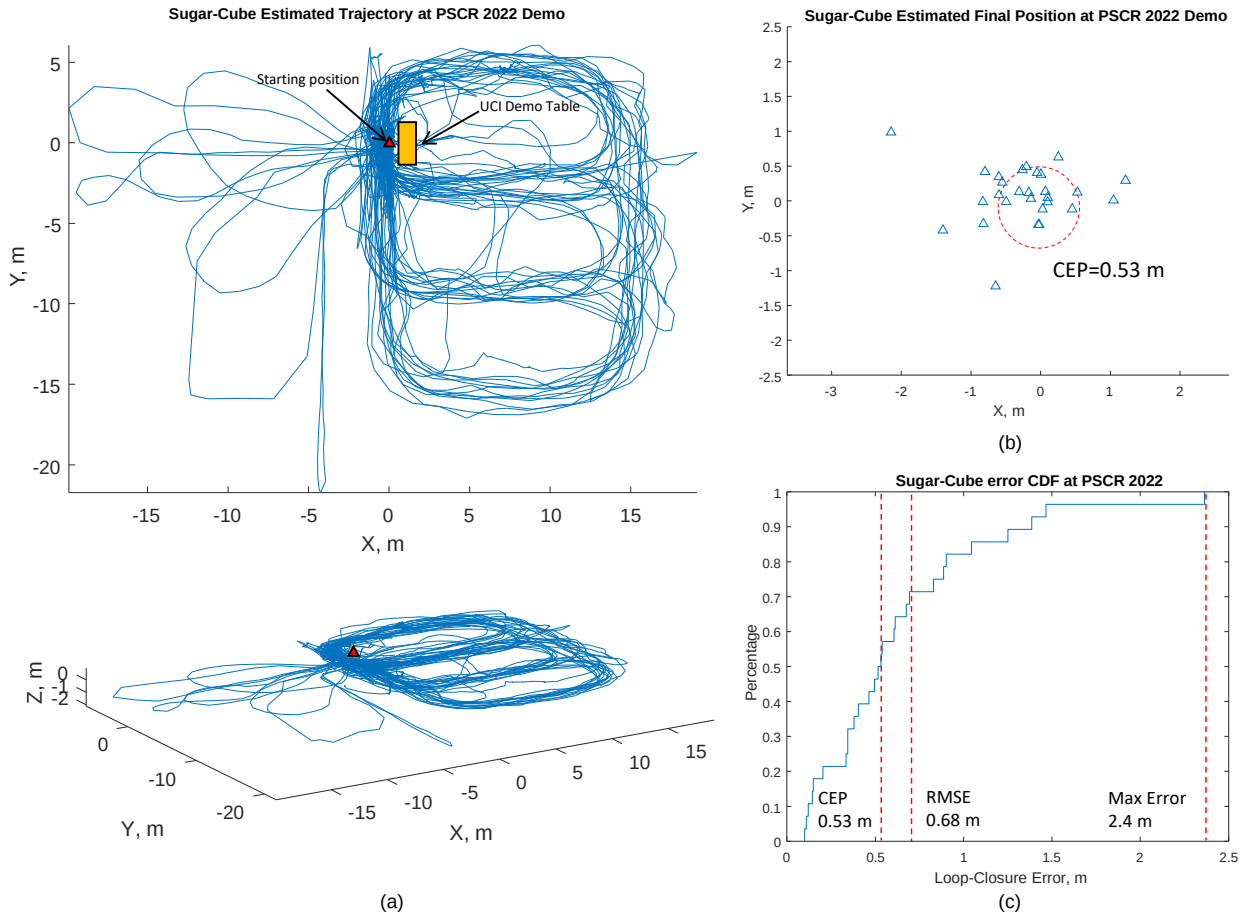


Figure A.68: (a) Trajectories produced by the Sugar-Cube platform in the 30 demo trials. (b) Estimated final destinations. (c) CDF of loop-closure errors.

presents the estimated final destinations of the 30 trials. The loop-closure errors had a Circular Error Probable (CEP) of 0.53 m. Figure A.68(c) shows a Cumulative Distribution Function (CDF) of the loop-closure errors. We could see that, in all the trials, the position Root Mean Squared Error (RMSE) was 0.68 m, and the maximum error was 2.4 m.

Demo Interface MATLAB Source Codes

```
% parsing sugar-cube platform
close all
clear all
clc

hWaitbar = waitbar(0, 'Iteration 1', 'Name', 'Solving
    problem','CreateCancelBtn','delete(gcbf)');
numData = 500000;

% Sugar Cube Labeled as 5 is COM6
% Sugar Cube Labeled as 2 is COM 18
% Unlabeled (good) Sugar Cube is COM25
device = serialport("COM25",115200);
flush(device)
position = nan(3,numData);
figure;
p_plot_ini = plot3(0,0,0,'rs','MarkerSize',5);hold on
patch([1 -1 -1 1]*100, [1 1 -1 -1]*100, [0 0 0 0],'k')
alpha(0.2)
p_plot = plot3(position(1,:),position(2,:),-position(3,:),'b');
% p_plot = plot(position(1,:),position(2,:),'b');hold on

p_plot_tip =
    plot3(position(1,1),position(2,1),-position(3,1),'b^','MarkerSize',5);
grid on
% p_plot_tip = plot(position(1,1),position(2,1),'b^','MarkerSize',5);

xlim([min(position(1,1),-10)-5 max(position(1,1),10)+5])
ylim([min(position(2,1),-10)-5 max(position(2,1),10)+5])
zlim([min(-position(3,1),-10)-5 max(-position(3,1),10)+5])
xlabel('X [m]')
ylabel('Y [m]')
zlabel('Z [m]')
dataStreamArray = nan(21,numData);
saveDate = date;
saveTimeStamp = datestr(now,'HH-MM-SS-FFF');
```

```
tic
dataline = '';
for ii = 1:numData
    dataline = read(device,1,'char');
    while dataline(end)~= char(10)
        %        disp(dataline(end))
        dataline = [dataline read(device,1,'char')];
    end
    dataline = string(dataline(1:end-1));


    dataStream = double(split(dataline,','));
    if length(dataStream) == 21
        dataStreamArray(:,ii) = dataStream;
        position(:,ii) = dataStream(14:16)';
        position(2,ii) = -position(2,ii);
        if mod(ii,1) == 0
            delete(p_plot)
            delete(p_plot_tip)
            p_plot = plot3(position(1,:),position(2,:),-position(3,:),'b');hold on
            %        p_plot = plot(position(1,:),position(2,:),'b');hold on

            p_plot_tip =
                plot3(position(1,ii),position(2,ii),-position(3,ii),'b^','MarkerSize',5);
            %        p_plot_tip =
                plot(position(1,ii),position(2,ii),'b^','MarkerSize',5);

            xlim([min(min(position(1,1:ii)),-10)-5
                max(max(position(1,1:ii)),10)+5])
            ylim([min(min(position(2,1:ii)),-10)-5
                max(max(position(2,1:ii)),10)+5])
            zlim([min(min(-position(3,1:ii)),-10)-5
                max(max(-position(3,1:ii)),10)+5])
            title(['Elapsed Time: ' num2str(floor(toc)) ' s'])
            drawnow
        end
    end
    if ~ishandle(hWaitbar)
        % Stop the if cancel button was pressed
        disp('Stopped by user');
        break;
    end
    dataline = '';

end

if ~exist(['dataset\' date])
```

```
    mkdir(['dataset\' date])
end

kk = numel(dir(['dataset\' saveDate '\*.mat']));

save(['dataset\' saveDate '\exp' num2str(kk) '00_' saveTimeStamp
    '.mat'],'dataStreamArray')

% close(hWaitbar)
figure;plot3(position(1,:),position(2,:),-position(3,:),'b');hold on
plot3(position(1,ii),position(2,ii),-position(3,ii),'b^');
plot3(0,0,0,'rs');
axis equal

clear device
```

This section presents a demonstration process developed for the Sugar-Cube platform. The demonstration process allowed for visualizing real-time positioning solutions in a point cloud map of the environment. 30 trials of the demonstration were conducted over the 3-day conference session, and the navigation performance of the Sugar-Cube had a CEP of 0.53 m, an RMSE of 0.68 m, and a maximum error of 2.4 m.

A.2.5 Sugar-Cube: C/C++ Implementation

The Main Script

```
////////////////////////////////////////////////////////////////////////////////
//  April 2023
//  Author: Chi-Shih Jao <chishihj@uci.edu>
////////////////////////////////////////////////////////////////////////////////
//  Sugar_Cube_Platform.ino
////////////////////////////////////////////////////////////////////////////////
#include "src\ICM42605-master\ICM42605.h"
#include "src\ICM42605-master\I2Cdev.h"
#include "src\ExtendedKalmanFilter.h"
#include "src\ICP20100_Custom_Library\ICP20100.h"

#define ICP_CLK_Rate 12000000
#define ICP_CS_Pin 16
```

```
#define ICP_INT_PIN 17

#define SPI_PORT SPI // Your desired SPI port.  Used only when "USE_SPI" is
defined
#define CS_PIN 2   // Which pin you connect CS to. Used only when "USE_SPI" is
defined

#define WIRE_PORT Wire   // Your desired Wire port.   Used when "USE_SPI" is
not defined
#define AD0_VAL 1        // The value of the last bit of the I2C address. 0:
0x68,1: 0x69
#define ICM42605_intPin1 20 // interrupt1 pin definitions, significant motion

#ifdef USE_SPI
ICM_20948_SPI myICM; // If using SPI create an ICM_20948_SPI object
#else
ICM_20948_I2C myICM; // Otherwise create an ICM_20948_I2C object
#endif

#define I2C_BUS Wire // Define the I2C bus (Wire instance) you wish to use

I2Cdev i2c_0(&I2C_BUS); // Instantiate the I2Cdev object and point to the
    desired I2C
bus

// declare neccesary variables
double CurrentTime = 0, PrevTime = 0, loopPeriod = 0, InitialTime = micros(),
ElapsedTime = 0;
extern double yaw_Deg, zupt, a, e2, cLat, sLat, ax, R_N, R_M, h, g, dt;
extern int cal;
extern double f_u, f_v, f_w, gB_l_x, gB_l_y, gB_l_z, roll, pitch, roll_Deg,
    pitch_Deg,
 latDeg, lonDeg;
extern ArrayMatrix<4, 1, double> q_b2n_0, q_e2N0_l, q_e2N_l;
extern ArrayMatrix<3, 1, double> V_0, w_e2i_n_l, aB_l, gB_l;
extern ArrayMatrix<1, 3, double> e3;
extern ArrayMatrix<1, 15, double> x, Q_diag;
extern ArrayMatrix<15, 15, double> P, Q, Id;
extern ArrayMatrix<15, 1, double> dx_l;
extern ArrayMatrix<3, 15, double> H;
extern ArrayMatrix<3, 3, double> R, O33, I33, A_11;
extern ArrayMatrix<1, 10, double> zupt_l;
extern gyroStruct gyro;
extern acclStruct accl;
extern ArrayMatrix<3, 1> r_e_n, g_n, LLA_0;
extern estimationStruct est;
extern NaviStatus input_cal, input;
```

```
extern IMUreadings sensor;
extern trueStruct true_t;
extern setting_constructed_data simdata;
extern zuptStruct zuptState;
double u[15], u_mean[15] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    mag_bias[3]
= {0, 0, 0}, smoothed_mag[3] = {0, 0, 0}, mag_heading = 0;
;
ArrayMatrix<W, 6, double> v;
double kk = 0, ini_cal = 3000;
ArrayMatrix<3, 1, double> NED;
String outputString, outputWirelessString = String(), subString1, subString2,
subString3, subString4, subString5;

uint8_t Ascale = AFS_16G, Gscale = GFS_2000DPS, AODR = AODR_500Hz, GODR =
    GODR_500Hz;

float aRes, gRes;
// scale resolutions per LSB for the accel and gyro sensor
int16_t ICM42605Data[7];
// Stores the 16-bit signed sensor output
float Gtemperature;
// Stores the real internal gyro temperature in degrees Celsius
float ax_icm42065, ay_icm42065, az_icm42065, gx_icm42065, gy_icm42065,
    gz_icm42065;
// variables to hold latest accel/gyro data values
uint8_t ICM42605status;
bool newICM42605Data = false;
ICM42605 ICM42605(&i2c_0); // instantiate ICM42605 class

float abs_pressure, raw_temperature, abs_pressure_based, altittude_delta;
bool icp_change = 0;
// Sensor is an ICP20100 object
ICP20100 ICP_sensor(ICP_CLK_Rate, ICP_CS_Pin); // clock rate, chip select pin

void myinthandler1()
{
  newICM42605Data = true;
}

void ICP_Interrupt(void)
{
  if (ICP_sensor.measure())
  {
    icp_change = 1;
  }
}
```

```
103
104 void setup()
105 {
106
107   // initialize sensor
108   SERIAL_PORT_USB.begin(115200);
109   SERIAL_PORT.begin(115200);
110
111   pinMode(ICM42605_intPin1, INPUT);
112
113   Wire.begin();         // set master mode - on Teensy will defaut to Pin 19 - SCL
114 and Pin 18 - SDA
115   Wire.setClock(1000000); // I2C frequency at 400 kHz
116   delay(1000);
117
118   i2c_0.I2Cscan();
119
120   // Read the ICM42605 Chip ID register, this is a good test of communication
121   Serial.println("ICM42605 accel/gyro...");
122   byte c = ICM42605.getChipID(); // Read CHIP_ID register for ICM42605
123   Serial.print("ICM42605 ");
124   Serial.print("I AM ");
125   Serial.print(c, HEX);
126   Serial.print(" I should be ");
127   Serial.println(0x42, HEX);
128   Serial.println(" ");
129   delay(1000);
130
131   ICM42605.reset(); // software reset ICM42605 to default registers
132
133   // get sensor resolutions, only need to do this once
134   aRes = ICM42605.getAres(Ascale);
135   gRes = ICM42605.getGres(Gscale);
136
137   ICM42605.init(Ascale, Gscale, AODR, GODR);
138
139   if (c != 0x42)
140     Serial.println(" ICM42605 not functioning!");
141
142   attachInterrupt(ICM42605_intPin1, myinthandler1, RISING); // define interrupt
        for
143 intPin1 output of ICM42605
144
145   ICM42605.status();
146
147   delay(1000);
148   if (newICM42605Data == true)
```

```
  {
    newICM42605Data = false;         // reset newData flag
    ICM42605status = ICM42605.status(); // INT1 cleared on status read

    PrevTime = CurrentTime;
    CurrentTime = micros();
    loopPeriod = CurrentTime - PrevTime;
    ElapsedTime = CurrentTime - InitialTime;

    ICM42605.readData(ICM42605Data);

    convertAGMT2Array(ICM42605Data, ElapsedTime, loopPeriod / 1000000, u, 0);
    printMeasurementArray(u);
  }

  ICP_sensor.begin();
}
int loop_counter = 0;
bool mag_ini_status = false;
void loop()
{

  if (loop_counter < 200 && mag_ini_status == false)
  {
    loop_counter++;
  }
  else if (loop_counter == 200 && mag_ini_status == false)
  {
    mag_heading = 0;
    // mag_heading =
-atan((smoothed_mag[1]-mag_bias[1])/(smoothed_mag[0]-mag_bias[0]))*r2d;
    SERIAL_PORT_USB.println("Magnetometer calibration done!");
    mag_ini_status = true;
    loop_counter = 0;
  }
  else
  {

    if (newICM42605Data == true)
    {
      newICM42605Data = false;         // reset newData flag
      ICM42605status = ICM42605.status(); // INT1 cleared on status read

      PrevTime = CurrentTime;
      CurrentTime = micros();
      loopPeriod = CurrentTime - PrevTime;
      ElapsedTime = CurrentTime - InitialTime;
```

```
196
197       ICM42605.readData(ICM42605Data);
198
199       convertAGMT2Array(ICM42605Data, ElapsedTime, loopPeriod / 1000000, u,
200  altittude_delta);
201
202       attachInterrupt(digitalPinToInterrupt(ICP_INT_PIN), ICP_Interrupt, LOW);
203       if (icp_change)
204       {
205         abs_pressure = ICP_sensor.getPressuremBar(); // Get Pressure in [mBar]
206         raw_temperature = ICP_sensor.getTemperatureC(); // Get Temperature in [C]
207         altittude_delta = local_altitude(abs_pressure, abs_pressure_based);
208         icp_change = 0;
209       }
210       detachInterrupt(digitalPinToInterrupt(ICP_INT_PIN));
211
212       // printMeasurementArray(u);
213       if (kk < W)
214       {
215         v(kk, 0) = u[0] * 9.817269086191379;
216         v(kk, 1) = u[1] * 9.817269086191379;
217         v(kk, 2) = u[2] * 9.817269086191379;
218         v(kk, 3) = u[3] * d2r;
219         v(kk, 4) = u[4] * d2r;
220         v(kk, 5) = u[5] * d2r;
221
222         u_mean[0] += u[0];
223         u_mean[1] += u[1];
224         u_mean[2] += u[2];
225         u_mean[3] += u[3];
226         u_mean[4] += u[4];
227         u_mean[5] += u[5];
228         u_mean[6] += u[6];
229         u_mean[7] += u[7];
230         u_mean[8] += u[8];
231         u_mean[9] += u[9];
232         u_mean[10] += u[10];
233         u_mean[11] += u[11];
234         u_mean[12] += u[12];
235         u_mean[13] += u[13];
236         u_mean[14] += u[14];
237         // Serial.println("Initialization: Stage 1");
238         abs_pressure_based = abs_pressure;
239         kk++;
240       }
241       else
242       {
```

```
        v.Submatrix(Slice<0, W - 1>(), Slice<0, 6>()) = v.Submatrix(Slice<1, W>(),
Slice<0, 6>()) + 0;
        v(W - 1, 0) = u[0] * 9.817269086191379;
        v(W - 1, 1) = u[1] * 9.817269086191379;
        v(W - 1, 2) = u[2] * 9.817269086191379;
        v(W - 1, 3) = u[3] * d2r;
        v(W - 1, 4) = u[4] * d2r;
        v(W - 1, 5) = u[5] * d2r;
        // printMeasurementArray(v);
        if (kk >= W && kk < (ini_cal))
        {
          u_mean[0] += u[0];
          u_mean[1] += u[1];
          u_mean[2] += u[2];
          u_mean[3] += u[3];
          u_mean[4] += u[4];
          u_mean[5] += u[5];
          u_mean[6] += u[6];
          u_mean[7] += u[7];
          u_mean[8] += u[8];
          u_mean[9] += u[9];
          u_mean[10] += u[10];
          u_mean[11] += u[11];
          u_mean[12] += u[12];
          u_mean[13] += u[13];
          u_mean[14] += u[14];
          // Serial.println("Initialization: Stage 2");
          // Serial.print(u_mean[14]);
        }
        else if (kk == (ini_cal))
        {
          u_mean[0] = u_mean[0] / kk;
          u_mean[1] = u_mean[1] / kk;
          u_mean[2] = u_mean[2] / kk;
          u_mean[3] = u_mean[3] / kk;
          u_mean[4] = u_mean[4] / kk;
          u_mean[5] = u_mean[5] / kk;
          u_mean[6] = u_mean[6] / kk;
          u_mean[7] = u_mean[7] / kk;
          u_mean[8] = u_mean[8] / kk;
          u_mean[9] = u_mean[9] / kk;
          u_mean[10] = u_mean[10] / kk;
          u_mean[11] = u_mean[11] / kk;
          u_mean[12] = u_mean[12] / kk;
          u_mean[13] = u_mean[13] / kk;
          u_mean[14] = u_mean[14] / kk;
          NavigationInitialization(u_mean, mag_heading);
```

```
290         }
291         else if (kk > (ini_cal) && kk <= cal)
292         {
293           NavigationInitialIteration(v, u);
294         }
295         else
296         {
297           NavigationIteration(v, u);
298           if ((loop_counter % 5) == 0)
299           {
300             if (outputWirelessString.length() == 0)
301             {
302               outputWirelessString =
303 printAndOutputMeasurementAndNavigationSolutionsNEDAndLLA(sensor,
304 zuptState, est);
305             }
306             SERIAL_PORT.print(outputWirelessString.substring(0, min(
307 outputWirelessString.length(), 2)));
308             if (outputWirelessString.length() > 2)
309             {
310               outputWirelessString = outputWirelessString.substring(2);
311             }
312             else
313             {
314               outputWirelessString = "";
315             }
316           }
317         }
318         kk++;
319       }
320     }
321     loop_counter++;
322   }
323 }
324
325 // Below here are some helper functions to print the data nicely!
326 void printPaddedInt16b(int16_t val)
327 {
328   if (val > 0)
329   {
330     SERIAL_PORT.print(" ");
331     if (val < 10000)
332     {
333       SERIAL_PORT.print("0");
334     }
335     if (val < 1000)
336     {
```

```
      SERIAL_PORT.print("0");
    }
    if (val < 100)
    {
      SERIAL_PORT.print("0");
    }
    if (val < 10)
    {
      SERIAL_PORT.print("0");
    }
  }
  else
  {
    SERIAL_PORT.print("-");
    if (abs(val) < 10000)
    {
      SERIAL_PORT.print("0");
    }
    if (abs(val) < 1000)
    {
      SERIAL_PORT.print("0");
    }
    if (abs(val) < 100)
    {
      SERIAL_PORT.print("0");
    }
    if (abs(val) < 10)
    {
      SERIAL_PORT.print("0");
    }
  }
  SERIAL_PORT.print(abs(val));
}

void printRawAGMT(ICM_20948_AGMT_t agmt)
{
  SERIAL_PORT.print("RAW. Acc [ ");
  printPaddedInt16b(agmt.acc.axes.x);
  SERIAL_PORT.print(", ");
  printPaddedInt16b(agmt.acc.axes.y);
  SERIAL_PORT.print(", ");
  printPaddedInt16b(agmt.acc.axes.z);
  SERIAL_PORT.print(" ], Gyr [ ");
  printPaddedInt16b(agmt.gyr.axes.x);
  SERIAL_PORT.print(", ");
  printPaddedInt16b(agmt.gyr.axes.y);
  SERIAL_PORT.print(", ");
```

```
  printPaddedInt16b(agmt.gyr.axes.z);
  SERIAL_PORT.print(" ], Mag [ ");
  printPaddedInt16b(agmt.mag.axes.x);
  SERIAL_PORT.print(", ");
  printPaddedInt16b(agmt.mag.axes.y);
  SERIAL_PORT.print(", ");
  printPaddedInt16b(agmt.mag.axes.z);
  SERIAL_PORT.print(" ], Tmp [ ");
  printPaddedInt16b(agmt.tmp.val);
  SERIAL_PORT.print(" ]");
  // SERIAL_PORT.println();
}

void printFormattedFloat(float val, uint8_t leading, uint8_t decimals)
{
  float aval = abs(val);
  if (val < 0)
  {
    SERIAL_PORT.print("-");
  }
  else
  {
    SERIAL_PORT.print(" ");
  }
  for (uint8_t indi = 0; indi < leading; indi++)
  {
    uint32_t tenpow = 0;
    if (indi < (leading - 1))
    {
      tenpow = 1;
    }
    for (uint8_t c = 0; c < (leading - 1 - indi); c++)
    {
      tenpow *= 10;
    }
    if (aval < tenpow)
    {
      SERIAL_PORT.print("0");
    }
    else
    {
      break;
    }
  }
  if (val < 0)
  {
    SERIAL_PORT.print(-val, decimals);
```

```
  }
  else
  {
    SERIAL_PORT.print(val, decimals);
  }
}

void printScaledAGMT(ICM_20948_AGMT_t agmt)
{
  SERIAL_PORT.print("Scaled. Acc (mg) [ ");
  printFormattedFloat(myICM.accX(), 5, 2);
  SERIAL_PORT.print(", ");
  printFormattedFloat(myICM.accY(), 5, 2);
  SERIAL_PORT.print(", ");
  printFormattedFloat(myICM.accZ(), 5, 2);
  SERIAL_PORT.print(" ], Gyr (DPS) [ ");
  printFormattedFloat(myICM.gyrX(), 5, 2);
  SERIAL_PORT.print(", ");
  printFormattedFloat(myICM.gyrY(), 5, 2);
  SERIAL_PORT.print(", ");
  printFormattedFloat(myICM.gyrZ(), 5, 2);
  SERIAL_PORT.print(" ], Mag (uT) [ ");
  printFormattedFloat(myICM.magX(), 5, 2);
  SERIAL_PORT.print(", ");
  printFormattedFloat(myICM.magY(), 5, 2);
  SERIAL_PORT.print(", ");
  printFormattedFloat(myICM.magZ(), 5, 2);
  SERIAL_PORT.print(" ], Tmp (C) [ ");
  printFormattedFloat(myICM.temp(), 5, 2);
  SERIAL_PORT.print(" ]");
}

void printNumberAGMT(ICM_20948_AGMT_t agmt)
{
  printFormattedFloat(myICM.accX(), 5, 2);
  SERIAL_PORT.print(", ");
  printFormattedFloat(myICM.accY(), 5, 2);
  SERIAL_PORT.print(", ");
  printFormattedFloat(myICM.accZ(), 5, 2);
  SERIAL_PORT.print(", ");
  printFormattedFloat(myICM.gyrX(), 5, 2);
  SERIAL_PORT.print(", ");
  printFormattedFloat(myICM.gyrY(), 5, 2);
  SERIAL_PORT.print(", ");
  printFormattedFloat(myICM.gyrZ(), 5, 2);
  SERIAL_PORT.print(", ");
  printFormattedFloat(myICM.magX(), 5, 2);
```

```
478   SERIAL_PORT.print(", ");
479   printFormattedFloat(myICM.magY(), 5, 2);
480   SERIAL_PORT.print(", ");
481   printFormattedFloat(myICM.magZ(), 5, 2);
482   SERIAL_PORT.print(", ");
483   printFormattedFloat(myICM.temp(), 5, 2);
484 }
485 
486 void convertAGMT2Array(int16_t *ICM42605Data, double elapsedTime, double
        loopPeriod,
487 double *u, double altittude_meas)
488 {
489   u[0] = (float)ICM42605Data[1] * aRes;
490   u[1] = (float)ICM42605Data[2] * aRes;
491   u[2] = (float)ICM42605Data[3] * aRes;
492   u[3] = (float)ICM42605Data[4] * gRes;
493   u[4] = (float)ICM42605Data[5] * gRes;
494   u[5] = (float)ICM42605Data[6] * gRes;
495   u[6] = 0;
496   u[7] = elapsedTime;
497   u[8] = altittude_meas;
498   u[9] = 0;
499   u[10] = loopPeriod;
500   u[11] = 0;
501   u[12] = 0;
502   u[13] = 0;
503   u[14] = 0;
504 }
505 
506 void printMeasurementArray(double *u)
507 {
508   SERIAL_PORT_USB.print("Sensor = [");
509   SERIAL_PORT_USB.print(u[0], 5);
510   SERIAL_PORT_USB.print(", ");
511   SERIAL_PORT_USB.print(u[1], 5);
512   SERIAL_PORT_USB.print(", ");
513   SERIAL_PORT_USB.print(u[2], 5);
514   SERIAL_PORT_USB.print(", ");
515   SERIAL_PORT_USB.print(u[3], 5);
516   SERIAL_PORT_USB.print(", ");
517   SERIAL_PORT_USB.print(u[4], 5);
518   SERIAL_PORT_USB.print(", ");
519   SERIAL_PORT_USB.print(u[5], 5);
520   SERIAL_PORT_USB.print(", ");
521   SERIAL_PORT_USB.print(u[11], 5);
522   SERIAL_PORT_USB.print(", ");
523   SERIAL_PORT_USB.print(u[12], 5);
```

```
  SERIAL_PORT_USB.print(", ");
  SERIAL_PORT_USB.print(u[13], 5);
  SERIAL_PORT_USB.print(", ");
  SERIAL_PORT_USB.print(u[14], 5);
  SERIAL_PORT_USB.print(", ");
  SERIAL_PORT_USB.print(u[10], 5);
  SERIAL_PORT_USB.print(" ]");
  SERIAL_PORT_USB.println();
}

void printMeasurementArray(ArrayMatrix<W, 6, double> vArray)
{
  SERIAL_PORT_USB.print("Sensor = [");
  SERIAL_PORT_USB.print(vArray(0, 0));
  SERIAL_PORT_USB.print(", ");
  SERIAL_PORT_USB.print(vArray(0, 1));
  SERIAL_PORT_USB.print(", ");
  SERIAL_PORT_USB.print(vArray(0, 2));
  SERIAL_PORT_USB.print(", ");
  SERIAL_PORT_USB.print(vArray(0, 3));
  SERIAL_PORT_USB.print(", ");
  SERIAL_PORT_USB.print(vArray(0, 4));
  SERIAL_PORT_USB.print(", ");
  SERIAL_PORT_USB.print(vArray(0, 5));
  SERIAL_PORT_USB.print(" ]");
  SERIAL_PORT_USB.println();
}

double local_altitude(double P_mbar, double P0_mbar)
// Given a pressure measurement P (mbar) and the pressure at a baseline P0
    (mbar),
// return altitude (meters) above baseline.
{
  return (44330.0 * (1 - pow(P_mbar / P0_mbar, 1 / 5.255)));
}
```

Strapdown Inertial Navigation Systems

```
#include "StrapDownInertialNavigationSystems.h"

NaviStatus navSLN_ZUPT(IMUreadings sensor, NaviStatus input){
   setting_constructed_data simdata;
   NaviStatus output;
   ArrayMatrix<4,1,double> q_b2n, q_e2n, q, q_b2nNm_wErth;
```

```
7    ArrayMatrix<3,1,double> w_b2i_b, f_b, LLA, v_nWrtE_n, delta_b_Prev,
        delta_n_Prev, delta_n2e_Prev, w_e2i_n, dotLLA, w_n2e_n, dTheta_b, w_n2i_n,
        dTheta_n, r_e_n, g_n, dg_centropital, gl_n, w_aux, f_n, dTheta_n2e_n, temp;
8    double dt, Omega, a, e2, cLat, sLat, ax, R_N, R_M, h, lonDeg, latDeg;
9    TwoElementStruct tempStruct;
10   TwoVectorStruct tempVectorStruct;
11
12   simdata = para_settings_initilize();
13
14   w_b2i_b = sensor.w_b2i_b;
15   f_b     = sensor.f_b;
16   dt      = sensor.dt;
17
18   q_b2n         = input.q_b2n;
19   q_e2n         = input.q_e2n;
20   LLA           = input.LLA; // [lonRad; latRad; alt]
21   v_nWrtE_n     = input.v_nWrtE_n;
22   // No use
23   delta_b_Prev  = input.delta_b_Prev;
24   delta_n_Prev  = input.delta_n_Prev;
25   delta_n2e_Prev = input.delta_n2e_Prev;
26
27   // ------------------------------------------------------------------------
28   Omega = simdata.earthrate;       // rad/s (earth's rate)
29   a     = 6378137.0;             // m ( semi-major axis)
30   e2    = 6.694380004260835e-3; // EarthEccentricitySq
31   // ------------------------------------------------------------------------
32
33   // ------------------------------------------------------------------------
34   cLat = cos(LLA(1,0));
35   sLat = sin(LLA(1,0));
36   temp = {
37      cLat,
38      0,
39      sLat*(-1)
40   };
41
42   w_e2i_n = temp*Omega;
43
44
45   ax   = 1-e2*pow(sLat,2);
46   R_N  = a/sqrt(ax);
47   R_M  = R_N*(1-e2)/ax;
48   h    = LLA(2,0);
49   dotLLA(2,0) = v_nWrtE_n(2,0)*(-1);
50   w_n2e_n(0,0) = v_nWrtE_n(1,0)/a;
51   w_n2e_n(1,0) = v_nWrtE_n(0,0)/a*(-1);
```

```
52    w_n2e_n(2,0) = 0;
53
54    // ============================================================================
55    // Attitude
56    // ============================================================================
57
58    dTheta_b     = w_b2i_b * dt;
59    tempVectorStruct = qintegrator(q_inv(q_b2n), dTheta_b, delta_b_Prev,0);
60    q = tempVectorStruct.OneVector;
61    delta_b_Prev = tempVectorStruct.TwoVector;
62    q_b2nNm_wErth = q_inv(q);
63    w_n2i_n      = w_e2i_n + w_n2e_n;
64    dTheta_n     = w_n2i_n * dt;
65
66    tempVectorStruct = qintegrator(q_b2nNm_wErth, dTheta_n, delta_n_Prev,0);
67    q_b2n = tempVectorStruct.OneVector;
68    delta_n_Prev = tempVectorStruct.TwoVector;
69    // ============================================================================
70    // Vel.& Pos.
71    // ============================================================================
72    // gravity
73    r_e_n         = {R_N*e2*sLat*cLat*(-1),
74                     0,
75                     R_N*ax*(-1) - h};
76    g_n           = gravityModel(LLA(1,0), vectorNorm(r_e_n), a, e2);
77    dg_centropital = crossProduct(w_e2i_n, crossProduct(w_e2i_n, r_e_n));
78    gl_n          = g_n - dg_centropital;
79    // ============================================================================
80    w_aux    = w_e2i_n * 2 +  w_n2e_n;
81    f_n      = quatRot(q_b2n, f_b);
82    v_nWrtE_n = v_nWrtE_n + (f_n - crossProduct(w_aux, v_nWrtE_n) + gl_n) * dt ;
83    // ============================================================================
84    dotLLA(2,0) = -v_nWrtE_n(2,0);
85    w_n2e_n(0,0) = v_nWrtE_n(1,0)/a;
86    w_n2e_n(1,0) = -v_nWrtE_n(0,0)/a;
87    w_n2e_n(2,0) = 0;
88    // ============================================================================
89    dTheta_n2e_n = w_n2e_n * dt;
90    tempVectorStruct = qintegrator(q_e2n, dTheta_n2e_n, delta_n2e_Prev,0);
91    q_e2n = tempVectorStruct.OneVector;
92    delta_n2e_Prev = tempVectorStruct.TwoVector;
93    tempStruct = q_e2N_toLonLatDeg(q_e2n);
94    lonDeg = tempStruct.One;
95    latDeg = tempStruct.Two;
96    LLA(0,0) = lonDeg*M_PI/180.0;
97    LLA(1,0) = latDeg*M_PI/180.0;
98    LLA(2,0) = LLA(2,0) + dotLLA(2,0) * dt;
```

```
    // ================================================================================
    output.q_b2n        = q_b2n;
    output.q_e2n        = q_e2n;
    output.LLA          = LLA;
    output.v_nWrtE_n    = v_nWrtE_n;
    output.delta_b_Prev = delta_b_Prev;
    output.delta_n_Prev = delta_n_Prev;
    output.delta_n2e_Prev = delta_n2e_Prev;
    // ================================================================================
    return output;
};

//ICM20948
setting_constructed_data para_settings_initilize (void){

  setting_constructed_data simdata;

  simdata.timespan=3600.0;
  simdata.altitude=41.0;
  simdata.latitude=33.64304*M_PI/180.0;
  simdata.longitude=-117.84004*M_PI/180.0;
  simdata.Ts=1.0/370.0;
  simdata.M=10.0;
  simdata.N = round(simdata.timespan/simdata.Ts)+1;
  simdata.init_heading= 0*M_PI/180.0;
  simdata.init_vel = {
     0,
     0,
     0
  };
  simdata.earthrate=7.2921150e-5;
  simdata.a=6378137.0;
  simdata.e2=6.694380004260835e-3;
  simdata.sigma_a=0.04/60*sqrt(simdata.Ts) * 50;
  simdata.sigma_g=0.18*M_PI/180/60*sqrt(simdata.Ts)*10;
  simdata.gyro=simdata.sigma_g;
  simdata.accel=simdata.sigma_a;
  simdata.zupt_window=0.05;
  simdata.Window_size=round(simdata.zupt_window/simdata.Ts);
  simdata.Window_size_for_step_detector=simdata.Window_size*5;
  simdata.gamma=3e5;
  simdata.threshold=6;
  simdata.factor=exp(simdata.threshold);
  simdata.factor_step=exp(6);
  simdata.sigma_initial_acc_bias = {0,0,0};
  simdata.sigma_initial_gyro_bias = {0,0,0};
  simdata.sigma_initial_pos = {0,0,0};
```

```
  simdata.sigma_initial_vel = {0,0,0};
  simdata.sigma_initial_att = {0,0,0};
  simdata.sigma_initial_acc_scale = {0.0001,0.0001,0.0001};
  simdata.sigma_initial_gyro_scale = {0.0001,0.0001,0.0001};
  simdata.acc_bias_driving_noise=13e-6*9.81*sqrt(simdata.Ts)*1;
  simdata.gyro_bias_driving_noise=3.3/3600*M_PI/180*sqrt(simdata.Ts)*1;
  simdata.acc_SF_driving_noise = 1e-10;
  simdata.gyro_SF_driving_noise = 1e-10;
  simdata.acc_ortho_driving_noise = 1e-8;
  simdata.gyro_rot_driving_noise = 1e-8;
  simdata.gyro_ortho_driving_noise = 1e-8;
  simdata.sigma_vel = {
     0.02,
     0.02,
     0.02
  };
  simdata.sigma_dis=0.1;
  simdata.sigma_mag=10.0;
  simdata.ALT_rate = 20.0;
  simdata.alt_resolution = 0.1;
  simdata.cam_dis_std = 0.01;
  simdata.cam_dis_vx = 0.001;
  simdata.cam_dis_vy = 0.001;
  simdata.cam_dis_vz = 0.001;
  simdata.cam_roll_std = 0.01;
  simdata.cam_pitch_std = 0.01;
  simdata.cam_yaw_std = 0.01;
  simdata.alpha = -pow(exp(9),1);
  simdata.theta = 800.5;
  simdata.beta = 0;
  simdata.Window_size_nlos=50.0;
  simdata.threshold_nlos = 0.000000001;
  simdata.factor_nlos = exp(simdata.threshold_nlos);
  simdata.UWB_std = 1.0;

  return simdata;
};
```

EKF Library

```
#include "ExtendedKalmanFilter.h"

double yaw_Deg, zupt, a, e2, cLat, sLat, ax, R_N, R_M, h, g, dt;
int cal;
```

```
double f_u, f_v, f_w, gB_l_x, gB_l_y, gB_l_z, roll, pitch, roll_Deg, pitch_Deg,
    latDeg, lonDeg;
ArrayMatrix<4, 1, double> q_b2n_0, q_e2N0_l, q_e2N_l;
ArrayMatrix<3, 1, double> V_0, w_e2i_n_l, aB_l, gB_l;
ArrayMatrix<1, 3, double> e3;
ArrayMatrix<1, 15, double> x, Q_diag;
ArrayMatrix<15, 15, double> P, Q, Id, Q_mtx;
ArrayMatrix<15, 1, double> dx_l;
ArrayMatrix<3, 15, double> H_ZUPT;
ArrayMatrix<1, 15, double> H_ALT;
ArrayMatrix<3, 3, double> R_ZUPT, O33, I33, A_11;
ArrayMatrix<1, 1, double> R_ALT;
ArrayMatrix<1, W, double> zupt_l;
gyroStruct gyro;
acclStruct accl;
ArrayMatrix<3, 1, double> r_e_n, g_n, LLA_0;
estimationStruct est;
NaviStatus input_cal, input;
IMUreadings sensor;
trueStruct true_t;
setting_constructed_data simdata;
zuptStruct zuptState;
ArrayMatrix<4, 4, double> blkdiag(ArrayMatrix<3, 3, double> tempMTX1,
    ArrayMatrix<1, 1
, double> tempMTX2)
{
  ArrayMatrix<4, 4, double> tempMTXout;
  tempMTXout.Fill(0);
  for (int kk = 0; kk < 3; kk++)
  {
    tempMTXout(kk, kk) = tempMTX1(kk, kk);
  }
  tempMTXout(3, 3) = tempMTX2(0, 0);
  return tempMTXout;
}
ArrayMatrix<15, 15, double> eye(ArrayMatrix<15, 15, double> IdMTX)
{
  IdMTX.Fill(0);
  for (int jj = 0; jj < 15; jj++)
  {
    IdMTX(jj, jj) = 1;
  }
  return IdMTX;
}
ArrayMatrix<3, 3, double> eye(ArrayMatrix<3, 3, double> IdMTX)
{
  IdMTX.Fill(0);
```

```
  for (int jj = 0; jj < 3; jj++)
  {
    IdMTX(jj, jj) = 1;
  }
  return IdMTX;
}
ArrayMatrix<15, 15, double> diag(ArrayMatrix<1, 15, double> xVec)
{
  ArrayMatrix<15, 15, double> tempMTX;
  tempMTX.Fill(0);
  for (int kk = 0; kk < 15; kk++)
  {
    tempMTX(kk, kk) = xVec(0, kk);
  }
  return tempMTX;
}
ArrayMatrix<15, 15, double> diag(ArrayMatrix<15, 1, double> xVec)
{
  ArrayMatrix<15, 15, double> tempMTX;
  tempMTX.Fill(0);
  for (int kk = 0; kk < 15; kk++)
  {
    tempMTX(kk, kk) = xVec(kk);
  }
  return tempMTX;
}
ArrayMatrix<1, 15, double> diag(ArrayMatrix<15, 15, double> tempMTX)
{
  ArrayMatrix<1, 15, double> tempx;
  tempx.Fill(0);
  for (int kk = 0; kk < 15; kk++)
  {
    tempx(0, kk) = tempMTX(kk, kk);
  }
  return tempx;
}
ArrayMatrix<3, 3, double> diag(ArrayMatrix<1, 3, double> xVec)
{
  ArrayMatrix<3, 3, double> tempMTX;
  tempMTX.Fill(0);
  for (int kk = 0; kk < 3; kk++)
  {
    tempMTX(kk, kk) = xVec(0, kk);
  }
  return tempMTX;
}
ArrayMatrix<3, 3, double> diag(ArrayMatrix<3, 1, double> xVec)
```

```
{
  ArrayMatrix<3, 3, double> tempMTX;
  tempMTX.Fill(0);
  for (int kk = 0; kk < 3; kk++)
  {
    tempMTX(kk, kk) = xVec(kk, 0);
  }
  return tempMTX;
}
ArrayMatrix<1, 3, double> diag(ArrayMatrix<3, 3, double> tempMTX)
{
  ArrayMatrix<1, 3, double> tempx;
  tempx.Fill(0);
  for (int kk = 0; kk < 3; kk++)
  {
    tempx(0, kk) = tempMTX(kk, kk);
  }
  return tempx;
}
ArrayMatrix<1, 15, double> mtxElementSquare(ArrayMatrix<1, 15, double> xVec)
{
  ArrayMatrix<1, 15, double> x_square;
  x_square.Fill(0);
  for (int kk = 0; kk < 15; kk++)
  {
    x_square(0, kk) = pow(xVec(0, kk), 2);
  }
  return x_square;
}
ArrayMatrix<1, 3, double> mtxElementSquare(ArrayMatrix<1, 3, double> xVec)
{
  ArrayMatrix<1, 3, double> x_square;
  x_square.Fill(0);
  for (int kk = 0; kk < 3; kk++)
  {
    x_square(0, kk) = pow(xVec(0, kk), 2);
  }
  return x_square;
}
ArrayMatrix<1, W, double> mtxElementSquare(ArrayMatrix<1, W, double> xVec)
{
  ArrayMatrix<1, W, double> x_square;
  x_square.Fill(0);
  for (int kk = 0; kk < W; kk++)
  {
    x_square(0, kk) = pow(xVec(0, kk), 2);
  }
```

```
145   return x_square;
146 }
147 ArrayMatrix<1, 15, double> mtxElementSquare(ArrayMatrix<15, 1, double> xVec)
148 {
149   ArrayMatrix<1, 15, double> x_square;
150   x_square.Fill(0);
151   for (int kk = 0; kk < 15; kk++)
152   {
153     x_square(0, kk) = pow(xVec(kk, 0), 2);
154   }
155   return x_square;
156 }
157 ArrayMatrix<1, 3, double> mtxElementSquare(ArrayMatrix<3, 1, double> xVec)
158 {
159   ArrayMatrix<1, 3, double> x_square;
160   x_square.Fill(0);
161   for (int kk = 0; kk < 3; kk++)
162   {
163     x_square(0, kk) = pow(xVec(kk, 0), 2);
164   }
165   return x_square;
166 }
167 ArrayMatrix<1, W, double> mtxElementSquare(BLA::Matrix<W, 1> xVec)
168 {
169   ArrayMatrix<1, W, double> x_square;
170   x_square.Fill(0);
171   for (int kk = 0; kk < W; kk++)
172   {
173     x_square(0, kk) = pow(xVec(kk), 2);
174   }
175   return x_square;
176 }
177 ArrayMatrix<15, 15, double> multipleM(ArrayMatrix<15, 15, double> M,
      ArrayMatrix<15,
178 15, double> N)
179 {
180   ArrayMatrix<15, 15, double> output;
181   output.Fill(0);
182   for (int i = 0; i < 15; i++)
183   {
184     for (int j = 0; j < 15; j++)
185     {
186       for (int k = 0; k < 15; k++)
187       {
188         output(i, j) += M(i, k) * N(k, j);
189       }
190     }
```

```
  }
  return output;
}
ArrayMatrix<15, 15, double> powM(ArrayMatrix<15, 15, double> M, int n)
{
  ArrayMatrix<15, 15, double> output;
  output = eye(output);
  if (n == 0)
  {
    return output;
  }
  else
  {
    for (int kk = 0; kk < n; kk++)
    {
      output = multipleM(output, M);
    }
  }
  return output;
}
ArrayMatrix<15, 15, double> expoM(ArrayMatrix<15, 15, double> M)
{
  ArrayMatrix<15, 15, double> M_exp;
  double factorial_list[] = {1, 1, 2, 6, 24, 120, 720, 5040, 40320, 362880,
     3628800};
  M_exp.Fill(0);
  for (int kk = 0; kk < 7; kk++)
  {
    M_exp = M_exp + powM(M, kk) factorial_list[kk];
  }
  return M_exp;
}
void NavigationInitialization(double u_mean[], double mag_heading_ini)
{
  simdata = para_settings_initilize();
  yaw_Deg = mag_heading_ini;
  cal = 8 * 500; Number of initial time steps for calibration zupt=1;
  ZUPT switch 1 is on, 10 is off f_u = 0;
  f_v = 0;
  f_w = 0;
  gB_l_x = 0;
  gB_l_y = 0;
  gB_l_z = 0;
  gravity a = 6378137.0;
  m(semi - major axis) e2 = 6.694380004260835e-3;
  EarthEccentricitySq cLat = cos(simdata.latitude);
  sLat = sin(simdata.latitude);
```

```
237   ax = 1 - e2 * pow(sLat, 2);
238   R_N = a sqrt(ax);
239   R_M = R_N * (1 - e2) ax;
240   h = simdata.altitude;
241   r_e_n = {-R_N * e2 * sLat * cLat, 0, -R_N * ax - h};
242   g_n = gravityModel(simdata.latitude, vectorNorm(r_e_n), a, e2);
243   g = vectorNorm(g_n);
244   f_u = u_mean[0] * g;
245   f_v = u_mean[1] * g;
246   f_w = u_mean[2] * g;
247   gB_l_x = u_mean[3] * d2r;
248   gB_l_y = u_mean[4] * d2r;
249   gB_l_z = u_mean[5] * d2r;
250   roll = atan2(-f_v, -f_w);
251   pitch = atan2(f_u, sqrt(f_v * f_v + f_w * f_w));
252   roll_Deg = roll * r2d;
253   pitch_Deg = pitch * r2d;
254   q_b2n_0 = Imu2YPRdeg_to_qb2n(roll_Deg, pitch_Deg, yaw_Deg);
255   latDeg = simdata.latitude * r2d;
256   lonDeg = simdata.longitude * r2d;
257   h = simdata.altitude;
258   V_0.Fill(0);
259   dt = simdata.Ts;
260   true_t.t = u_mean[7];
261   q_e2N0_l = lonLatDegTo_q_e2N(lonDeg, latDeg);
262   Quaternion of navigation to earth frame combined with transport rate q_e2N_l =
263 integVel_into_q_e2N(q_e2N0_l, h, V_0, dt);
264   LLA_0 = {simdata.longitude, simdata.latitude, h};
265   input.q_b2n = q_b2n_0;
266   input.q_e2n = q_e2N_l;
267   input.LLA = LLA_0;
268   input.v_nWrtE_n = V_0;
269   input.delta_b_Prev.Fill(0);
270   input.delta_n_Prev.Fill(0);
271   input.delta_n2e_Prev.Fill(0);
272   sensor.dt = dt;
273   true_t.dt = dt;
274   est.t = true_t.t;
275   n = length(est.t);
276   est.q_b2n_l = input.q_b2n;
277   est.q_e2n_l = input.q_e2n;
278   est.LLA_l = input.LLA;
279   est.v_nWrtE_n_l = input.v_nWrtE_n;
280   set gyro noise characteristics gyro.sigma_AWN = 2e-4 * d2r * 0;
281   rad gyro.sigma_ARW = simdata.sigma_g sqrt(simdata.Ts) * 1;
282   rad sqrt(s) gyro.sigma_RRW = simdata.gyro_bias_driving_noise sqrt(simdata.Ts) *
        1;
```

```
283   (rad s) sqrt(s) gyro.Bias = ~(simdata.sigma_initial_gyro_bias);
284   rad s set accel noise characteristics accl.sigma_VWN = 4.5e-4 * 0;
285   m s accl.sigma_VRW = simdata.sigma_a sqrt(simdata.Ts) * 1;
286   m s sqrt(s) accl.sigma_AcRW = simdata.acc_bias_driving_noise sqrt(simdata.Ts) *
          1;
287   m s ^ 2 sqrt(s) accl.Bias = ~(simdata.sigma_initial_acc_bias);
288   m s ^ 2 w_e2i_n_l = earthRateInBody(0, 0, 0, LLA_0(1, 0) * 180.0 M_PI);
289   att, vel, pos, gB, aB e3.Fill(1);
290   Q_diag = ((((e3 * pow(gyro.sigma_ARW, 2) || e3 * pow(accl.sigma_VRW, 2)) || e3
          * 0)
291 || e3 * pow(gyro.sigma_RRW, 2)) || e3 * pow(accl.sigma_AcRW, 2)) * true_t.dt;
292   noise matrix in the EKF x = ((((simdata.sigma_initial_att || simdata.
293 sigma_initial_vel) || simdata.sigma_initial_pos) ||
        simdata.sigma_initial_gyro_bias
294 3.0) || simdata.sigma_initial_acc_bias 3.0);
295   initialize state in the EKF P = diag(mtxElementSquare(x));
296   H_ZUPT.Fill(0);
297   H_ZUPT(0, 3) = 1;
298   H_ZUPT(1, 4) = 1;
299   H_ZUPT(2, 5) = 1;
300   H_ALT.Fill(0);
301   H_ALT(0, 8) = 1;
302   R_ZUPT = diag(mtxElementSquare(simdata.sigma_vel));
303   R_ALT(0, 0) = simdata.alt_resolution;
304   aB_l.Fill(0);
305   gB_l = {gB_l_x, gB_l_y, gB_l_z};
306   dx_l.Fill(0);
307   Id = eye(Id);
308   zupt_l.Fill(0);
309   zupt_l(0, 0) = 1;
310   O33.Fill(0);
311   I33 = eye(I33);
312   A_11 = skew(w_e2i_n_l) * (-1);
313   input_cal = input;
314   return;
315 }
316 void printMatrix(ArrayMatrix<15, 15, double> X_mtx)
317 {
318   for (int ii = 0; ii < 15; ii++)
319   {
320     for (int jj = 0; jj < 15; jj++)
321     {
322       OUTPUT_PORT.print(X_mtx(ii, jj), 15);
323       OUTPUT_PORT.print("");
324     }
325     OUTPUT_PORT.println(" ");
326   }
```

```
  OUTPUT_PORT.println("");
}

void printMatrix(ArrayMatrix<3, 3, double> X_mtx)
{
  for (int ii = 0; ii < 3; ii++)
  {
    for (int jj = 0; jj < 3; jj++)
    {
      OUTPUT_PORT.print(X_mtx(ii, jj), 15);
      OUTPUT_PORT.print(" ");
    }
    OUTPUT_PORT.println("");
  }
  OUTPUT_PORT.println(" ");
}
void NavigationInitialIteration(ArrayMatrix<W, 6, double> v, double u[])
{
  zuptStruct zuptState;
  ArrayMatrix<3, 3, double> A14, A21;
  ArrayMatrix<15, 15, double> A, F_mtx, Q_mtx, test1, test2;
  sensor.w_b2i_b
      << v(0, 3),
      v(0, 4), v(0, 5);
  sensor.w_b2i_b = sensor.w_b2i_b - gB_l;
  sensor.f_b = {v(0, 0), v(0, 1), v(0, 2)};
  sensor.f_b = sensor.f_b - aB_l;
  zuptState = SHOE_detector(~v);
  sensor.dt = u[10];
  input_cal = navSLN_ZUPT(sensor, input_cal);
  A14 = quat2dcos(input_cal.q_b2n) * (-1);
  A21 = skew(A14 * sensor.f_b * (-1));
  A = (((((((A_11 || O33) || O33) || A14) || O33) && ((((A21 || O33) || O33) ||
      O33)
|| A14 * (-1))) && ((((O33 || I33) || O33) || O33) || O33)) && ((((O33 || O33) ||
O33) || O33) || O33)) && ((((O33 || O33) || O33) || O33) || O33);
  F_mtx = expoM(A * sensor.dt);
  Q_mtx = diag(Q_diag);
  P = ((F_mtx * P) * (~F_mtx)) + Q_mtx;
  ArrayMatrix<4, 4, double> S;
  ArrayMatrix<15, 4, double> K;
  ArrayMatrix<4, 1, double> z;
  bool zupt_on = false, alt_on = false;
  ArrayMatrix<1, 1, double> z_ALT = {u[8] + simdata.altitude - input_cal.LLA(2,
      0)};
  if (true)
  {
```

```
    z = input_cal.v_nWrtE_n && z_ALT;
    ArrayMatrix<4, 15, double> H = H_ZUPT && H_ALT;
    ArrayMatrix<4, 4, double> R = blkdiag(R_ZUPT, R_ALT);
    S = (((H * P) * (~H)) + R);
    K = (P * (~H)) * S.Inverse();
    P = (Id - K * H) * P;
    dx_l = K * z;
    gB_l = gB_l + dx_l.Submatrix(Slice<9, 12>(), Slice<0, 1>());
    aB_l = aB_l + dx_l.Submatrix(Slice<12, 15>(), Slice<0, 1>());
    input_cal.LLA(1, 0) = input_cal.LLA(1, 0) - dx_l(6, 0) simdata.a;
    input_cal.LLA(0, 0) = input_cal.LLA(0, 0) - (dx_l(7, 0)
       simdata.a)cos(input_cal.
LLA(1, 0));
    input_cal.LLA(2, 0) = input_cal.LLA(2, 0) + dx_l(8, 0);
    input_cal.q_e2n = lonLatDegTo_q_e2N(input_cal.LLA(0, 0) * r2d,
       input_cal.LLA(1, 0)
 * r2d);
    input_cal.v_nWrtE_n = input_cal.v_nWrtE_n - dx_l.Submatrix(Slice<3, 6>(),
       Slice<0,
 1>());
    input_cal.q_b2n = q_mult(input_cal.q_b2n, theta2quat(dx_l.Submatrix(Slice<0,
       3>(),
 Slice<0, 1>()) * (-1)));
  }
  P = (P + (~P))2.0;
  est.q_b2n_l = input_cal.q_b2n;
  est.q_e2n_l = input_cal.q_e2n;
  est.LLA_l = input_cal.LLA;
  est.v_nWrtE_n_l = input_cal.v_nWrtE_n;
}
void NavigationIteration(ArrayMatrix<W, 6, double> v, double u[])
{
  ArrayMatrix<3, 3, double> A14, A21;
  ArrayMatrix<15, 15, double> A, F_mtx;
  sensor.w_b2i_b = {v(0, 3), v(0, 4), v(0, 5)};
  sensor.w_b2i_b = sensor.w_b2i_b - gB_l;
  sensor.f_b = {v(0, 0), v(0, 1), v(0, 2)};
  sensor.f_b = sensor.f_b - aB_l;
  zuptState = SHOE_detector(~v);
  sensor.dt = u[10];
  input = navSLN_ZUPT(sensor, input);
  A14 = quat2dcos(input.q_b2n) * (-1);
  A21 = skew((A14)*sensor.f_b * (-1));
  A = (((((((A_11 || O33) || O33) || A14) || O33) && ((((A21 || O33) || O33) ||
     O33)
|| A14 * (-1))) && ((((O33 || I33) || O33) || O33) || O33)) && ((((O33 || O33) ||
O33) || O33) || O33)) && ((((O33 || O33) || O33) || O33) || O33);
```

```
414   F_mtx = expoM(A * sensor.dt);
415   P = ((F_mtx * P) * (~F_mtx)) + diag(Q_diag);
416   bool zupt_on = false, alt_on = false;
417   int z_size = 0;
418   if (zuptState.zupt_l == zupt)
419   {
420     z_size = z_size + 3;
421     zupt_on = true;
422   }
423   if (!isnan(u[8]))
424   {
425     z_size = z_size + 1;
426     alt_on = true;
427   }
428   if (zupt_on || alt_on)
429   {
430     if (z_size == 4)
431     {
432       ArrayMatrix<4, 4, double> S;
433       ArrayMatrix<15, 4, double> K;
434       ArrayMatrix<4, 15, double> H;
435       ArrayMatrix<4, 4, double> R;
436       ArrayMatrix<4, 1, double> z;
437       ArrayMatrix<1, 1, double> z_ALT = {u[8] + simdata.altitude - input.LLA(2,
          0)};
438       z = input.v_nWrtE_n && z_ALT;
439       H = H_ZUPT && H_ALT;
440       R = blkdiag(R_ZUPT, R_ALT);
441       S = (((H * P) * (~H)) + R);
442       K = (P * (~H)) * S.Inverse();
443       P = (Id - K * H) * P;
444       dx_l = K * z;
445       gB_l = gB_l + dx_l.Submatrix(Slice<9, 12>(), Slice<0, 1>());
446       aB_l = aB_l + dx_l.Submatrix(Slice<12, 15>(), Slice<0, 1>());
447       input.LLA(1, 0) = input.LLA(1, 0) - dx_l(6, 0) simdata.a;
448       input.LLA(0, 0) = input.LLA(0, 0) - dx_l(7, 0) simdata.a cos(input.LLA(1,
          0));
449       input.LLA(2, 0) = input.LLA(2, 0) + dx_l(8, 0);
450       input.q_e2n = lonLatDegTo_q_e2N(input.LLA(0, 0) * r2d, input.LLA(1, 0) *
          r2d);
451       input.v_nWrtE_n = input.v_nWrtE_n - dx_l.Submatrix(Slice<3, 6>(), Slice<0,
452 1>());
453       input.q_b2n = q_mult(input.q_b2n, theta2quat(dx_l.Submatrix(Slice<0, 3>(),
454 Slice<0, 1>()) * (-1)));
455     }
456     else if (z_size == 3)
457     {
```

```
458       ArrayMatrix<3, 3, double> S;
459       ArrayMatrix<15, 3, double> K;
460       ArrayMatrix<3, 15, double> H;
461       ArrayMatrix<3, 3, double> R;
462       ArrayMatrix<3, 1, double> z;
463       ArrayMatrix<1, 1, double> z_ALT = {u[8] + simdata.altitude - input.LLA(2,
              0)};
464       z = input.v_nWrtE_n;
465       H = H_ZUPT;
466       R = R_ZUPT;
467       S = (((H * P) * (~H)) + R);
468       K = (P * (~H)) * S.Inverse();
469       P = (Id - K * H) * P;
470       dx_l = K * z;
471       gB_l = gB_l + dx_l.Submatrix(Slice<9, 12>(), Slice<0, 1>());
472       aB_l = aB_l + dx_l.Submatrix(Slice<12, 15>(), Slice<0, 1>());
473       input.LLA(1, 0) = input.LLA(1, 0) - dx_l(6, 0) simdata.a;
474       input.LLA(0, 0) = input.LLA(0, 0) - dx_l(7, 0) simdata.a cos(input.LLA(1,
              0));
475       input.LLA(2, 0) = input.LLA(2, 0) + dx_l(8, 0);
476       input.q_e2n = lonLatDegTo_q_e2N(input.LLA(0, 0) * r2d, input.LLA(1, 0) *
              r2d);
477       input.v_nWrtE_n = input.v_nWrtE_n - dx_l.Submatrix(Slice<3, 6>(), Slice<0,
478 1>());
479       input.q_b2n = q_mult(input.q_b2n, theta2quat(dx_l.Submatrix(Slice<0, 3>(),
480 Slice<0, 1>()) * (-1)));
481     }
482     else
483     {
484       ArrayMatrix<1, 1, double> S;
485       ArrayMatrix<15, 1, double> K;
486       ArrayMatrix<1, 15, double> H;
487       ArrayMatrix<1, 1, double> R;
488       ArrayMatrix<1, 1, double> z;
489       ArrayMatrix<1, 1, double> z_ALT = {u[8] + simdata.altitude - input.LLA(2,
              0)};
490       z = z_ALT;
491       H = H_ALT;
492       R = R_ALT;
493       S = (((H * P) * (~H)) + R);
494       K = (P * (~H)) * S.Inverse();
495       P = (Id - K * H) * P;
496       dx_l = K * z;
497       gB_l = gB_l + dx_l.Submatrix(Slice<9, 12>(), Slice<0, 1>());
498       aB_l = aB_l + dx_l.Submatrix(Slice<12, 15>(), Slice<0, 1>());
499       input.LLA(1, 0) = input.LLA(1, 0) - dx_l(6, 0) simdata.a;
```

```
500       input.LLA(0, 0) = input.LLA(0, 0) - dx_l(7, 0) simdata.a cos(input.LLA(1,
              0));
501       input.LLA(2, 0) = input.LLA(2, 0) + dx_l(8, 0);
502       input.q_e2n = lonLatDegTo_q_e2N(input.LLA(0, 0) * r2d, input.LLA(1, 0) *
              r2d);
503       input.v_nWrtE_n = input.v_nWrtE_n - dx_l.Submatrix(Slice<3, 6>(), Slice<0,
504 1>());
505       input.q_b2n = q_mult(input.q_b2n, theta2quat(dx_l.Submatrix(Slice<0, 3>(),
506 Slice<0, 1>()) * (-1)));
507     }
508   }
509   P = (P + (~P))2.0;
510   est.q_b2n_l = input.q_b2n;
511   est.q_e2n_l = input.q_e2n;
512   est.LLA_l = input.LLA;
513   est.v_nWrtE_n_l = input.v_nWrtE_n;
514 }
515 ArrayMatrix<6, 1, double> MatrixRowSum(ArrayMatrix<6, W, double> tempMtx)
516 {
517   ArrayMatrix<6, 1, double> tempMean;
518   tempMean.Fill(0);
519   for (int kk = 0; kk < W; kk++)
520   {
521     tempMean(0, 0) = tempMean(0, 0) + tempMtx(0, kk);
522     tempMean(1, 0) = tempMean(1, 0) + tempMtx(1, kk);
523     tempMean(2, 0) = tempMean(2, 0) + tempMtx(2, kk);
524     tempMean(3, 0) = tempMean(3, 0) + tempMtx(3, kk);
525     tempMean(4, 0) = tempMean(4, 0) + tempMtx(4, kk);
526     tempMean(5, 0) = tempMean(5, 0) + tempMtx(5, kk);
527   }
528   return tempMean;
529 }
530 ArrayMatrix<3, 1, double> MatrixRowSum(ArrayMatrix<3, W, double> tempMtx)
531 {
532   ArrayMatrix<3, 1, double> tempMean;
533   tempMean.Fill(0);
534   for (int kk = 0; kk < W; kk++)
535   {
536     tempMean(0, 0) = tempMean(0, 0) + tempMtx(0, kk);
537     tempMean(1, 0) = tempMean(1, 0) + tempMtx(1, kk);
538     tempMean(2, 0) = tempMean(2, 0) + tempMtx(2, kk);
539   }
540   return tempMean;
541 }
542 double ArraySum(ArrayMatrix<1, W, double> tempArray)
543 {
544   double tempMean = 0;
```

```
  for (int kk = 0; kk < W; kk++)
  {
    tempMean = tempMean + tempArray(0, kk);
  }
  return tempMean;
}
double ArraySum(ArrayMatrix<3, 1, double> tempArray)
{
  double tempMean = 0;
  for (int kk = 0; kk < 3; kk++)
  {
    tempMean = tempMean + tempArray(kk, 0);
  }
  return tempMean;
}
zuptStruct SHOE_detector(ArrayMatrix<6, W, double> u)
{
  zuptStruct zuptStateTemp;
  double total_x, total_y, total_z, c = W, sigma2_a, sigma2_g, g, total, zuptTemp;
  ArrayMatrix<3, 1, double> u_n;
  ArrayMatrix<3, 1, double> test_x;
  BLA::Matrix<2, 1> test_y;
  ArrayMatrix<3, W, double> tempSubMTX;
  ArrayMatrix<1, W, double> tempSubArrayOne, tempSubArrayTwo;
  total_x = 1;
  total_y = 1;
  total_z = 1;
  test_x.Fill(0);
  test_y = test_x.Submatrix(Slice<0, 2>(), Slice<0, 1>());
  c = W;
  sigma2_a = pow((simdata.sigma_a / simdata.Ts), 2);
  sigma2_g = pow((simdata.sigma_g / simdata.Ts), 2);
  g = 9.796;
  tempSubMTX = u.Submatrix(Slice<0, 3>(), Slice<0, W>());
  u_n = MatrixRowSum(tempSubMTX) / W;
  u_n = u_n / vectorNorm(u_n);
  Unit vector along the specific force u.Submatrix(Slice<0, 1>(), Slice<0, W>()) =
u.Submatrix(Slice<0, 1>(), Slice<0, W>()) - u_n(0, 0) * g;
  u.Submatrix(Slice<1, 2>(), Slice<0, W>()) = u.Submatrix(Slice<1, 2>(), Slice<0,
W>()) - u_n(1, 0) * g;
  u.Submatrix(Slice<2, 3>(), Slice<0, W>()) = u.Submatrix(Slice<2, 3>(), Slice<0,
W>()) - u_n(2, 0) * g;
  tempSubArrayOne = u.Submatrix(Slice<0, 1>(), Slice<0, W>());
  tempSubArrayTwo = u.Submatrix(Slice<3, 4>(), Slice<0, W>());
  total_x = ArraySum(mtxElementSquare(tempSubArrayOne)) / sigma2_a +
ArraySum(mtxElementSquare(tempSubArrayTwo)) sigma2_g;
  tempSubArrayOne = u.Submatrix(Slice<1, 2>(), Slice<0, W>());
```

```
  tempSubArrayTwo = u.Submatrix(Slice<4, 5>(), Slice<0, W>());
  total_y = ArraySum(mtxElementSquare(tempSubArrayOne)) / sigma2_a +
ArraySum(mtxElementSquare(tempSubArrayTwo)) sigma2_g;
  tempSubArrayOne = u.Submatrix(Slice<2, 3>(), Slice<0, W>());
  tempSubArrayTwo = u.Submatrix(Slice<5, 6>(), Slice<0, W>());
  total_z = ArraySum(mtxElementSquare(tempSubArrayOne)) / sigma2_a +
ArraySum(mtxElementSquare(tempSubArrayTwo)) sigma2_g;
  total_x = total_x / c;
  total_y = total_y / c;
  total_z = total_z / c;
  total = total_x + total_y + total_z;
  if (total < simdata.factor)
  {
    zuptTemp = 1;
  }
  else
  {
    zuptTemp = 0;
  }
  zuptStateTemp = (zuptStruct){.zupt_l = zuptTemp, .sum_IMU = total, .sum_IMU_x =
total_x, .sum_IMU_y = total_y, .sum_IMU_z = total_z};
  return zuptStateTemp;
}
ArrayMatrix<3, 1, double> LLA2NED(ArrayMatrix<3, 1, double> LLA_cur)
{
  ArrayMatrix<3, 1, double> NED;
  NED(0, 0) = (LLA_cur(1, 0) - LLA_0(1, 0)) * simdata.a;
  NED(1, 0) = (LLA_cur(0, 0) - LLA_0(0, 0)) * cos(LLA_cur(1, 0)) * simdata.a;
  NED(2, 0) = (LLA_cur(2, 0) - LLA_0(2, 0)) * (-1);
  return NED;
}
void printNavigationSolutionLLA(ArrayMatrix<3, 1, double> LLA_cur)
{
  OUTPUT_PORT.print("Longtitude, Lattitude, Alttitude:");
  OUTPUT_PORT.print(" [");
  OUTPUT_PORT.print(LLA_cur(0, 0), 10);
  OUTPUT_PORT.print(",");
  OUTPUT_PORT.print(LLA_cur(1, 0), 10);
  OUTPUT_PORT.print(" ,");
  OUTPUT_PORT.print(LLA_cur(2, 0), 10);
  OUTPUT_PORT.print(" ]");
  OUTPUT_PORT.println("");
  OUTPUT_PORT.print("(Initial)Longtitude, Lattitude, Alttitude:");
  OUTPUT_PORT.print(" [");
  OUTPUT_PORT.print(LLA_0(0, 0), 10);
  OUTPUT_PORT.print(",");
  OUTPUT_PORT.print(LLA_0(1, 0), 10);
```

```
639   OUTPUT_PORT.print(" ,");
640   OUTPUT_PORT.print(LLA_0(2, 0), 10);
641   OUTPUT_PORT.print(" ]");
642   OUTPUT_PORT.println("");
643 }
644 void printNavigationSolution(ArrayMatrix<3, 1, double> NED)
645 {
646   OUTPUT_PORT.print("North, East, Down (m):");
647   OUTPUT_PORT.print(" [");
648   OUTPUT_PORT.print(NED(0, 0), 2);
649   OUTPUT_PORT.print(",");
650   OUTPUT_PORT.print(NED(1, 0), 2);
651   OUTPUT_PORT.print(" ,");
652   OUTPUT_PORT.print(NED(2, 0), 2);
653   OUTPUT_PORT.print(" ]");
654   OUTPUT_PORT.println("");
655 }
656 void printVelocity(ArrayMatrix<3, 1, double> vel_sol)
657 {
658   OUTPUT_PORT.print("Velocities (m s):");
659   OUTPUT_PORT.print(" [");
660   OUTPUT_PORT.print(vel_sol(0, 0), 2);
661   OUTPUT_PORT.print(",");
662   OUTPUT_PORT.print(vel_sol(1, 0), 2);
663   OUTPUT_PORT.print(" ,");
664   OUTPUT_PORT.print(vel_sol(2, 0), 2);
665   OUTPUT_PORT.print(" ]");
666   OUTPUT_PORT.println("");
667 }
668 void printOrientation(ThreeElementStruct oreintationVec)
669 {
670   OUTPUT_PORT.print("Roll, pitch, yaw (deg):");
671   OUTPUT_PORT.print(" [");
672   OUTPUT_PORT.print(oreintationVec.One, 2);
673   OUTPUT_PORT.print(",");
674   OUTPUT_PORT.print(oreintationVec.Two, 2);
675   OUTPUT_PORT.print(" ,");
676   OUTPUT_PORT.print(oreintationVec.Three, 2);
677   OUTPUT_PORT.print(" ]");
678   OUTPUT_PORT.println("");
679 }
680 void printMeasurementAndNavigationSolutions(IMUreadings currentSensor, zuptStruct
681 currentZUPT, estimationStruct estNavSolution)
682 {
683   ArrayMatrix<3, 1, double> posVec;
684   ThreeElementStruct oreintationVec;
685   oreintationVec = qb2nImu2YPRdeg(estNavSolution.q_b2n_l);
```

```
686   posVec = LLA2NED(estNavSolution.LLA_l);
687   OUTPUT_PORT.print(currentSensor.dt, 4);
688   OUTPUT_PORT.print(',');
689   OUTPUT_PORT.print(currentSensor.f_b(0), 4);
690   OUTPUT_PORT.print(' ,');
691   OUTPUT_PORT.print(currentSensor.f_b(1), 4);
692   OUTPUT_PORT.print(' ,');
693   OUTPUT_PORT.print(currentSensor.f_b(2), 4);
694   OUTPUT_PORT.print(' ,');
695   OUTPUT_PORT.print(currentSensor.w_b2i_b(0), 4);
696   OUTPUT_PORT.print(' ,');
697   OUTPUT_PORT.print(currentSensor.w_b2i_b(1), 4);
698   OUTPUT_PORT.print(' ,');
699   OUTPUT_PORT.print(currentSensor.w_b2i_b(2), 4);
700   OUTPUT_PORT.print(' ,');
701   OUTPUT_PORT.print(oreintationVec.One, 4);
702   OUTPUT_PORT.print(' ,');
703   OUTPUT_PORT.print(oreintationVec.Two, 4);
704   OUTPUT_PORT.print(' ,');
705   OUTPUT_PORT.print(oreintationVec.Three, 4);
706   OUTPUT_PORT.print(' ,');
707   OUTPUT_PORT.print(estNavSolution.v_nWrtE_n_l(0), 4);
708   OUTPUT_PORT.print(' ,');
709   OUTPUT_PORT.print(estNavSolution.v_nWrtE_n_l(1), 4);
710   OUTPUT_PORT.print(' ,');
711   OUTPUT_PORT.print(estNavSolution.v_nWrtE_n_l(2), 4);
712   OUTPUT_PORT.print(' ,');
713   OUTPUT_PORT.print(posVec(0), 4);
714   OUTPUT_PORT.print(' ,');
715   OUTPUT_PORT.print(posVec(1), 4);
716   OUTPUT_PORT.print(' ,');
717   OUTPUT_PORT.print(posVec(2), 4);
718   OUTPUT_PORT.print(' ,');
719   OUTPUT_PORT.print(currentZUPT.zupt_l, 4);
720   OUTPUT_PORT.print(' ,');
721   OUTPUT_PORT.print(currentZUPT.sum_IMU, 4);
722   OUTPUT_PORT.println();
723 }
724
725 String printAndOutputMeasurementAndNavigationSolutions(IMUreadings currentSensor,
726                                                        zuptStruct currentZUPT,
727 estimationStruct
728 estNavSolution)
729 {
730
731   String ouputString = "";
732
```

```
733   ArrayMatrix<3, 1, double> posVec;
734   ThreeElementStruct oreintationVec;
735   oreintationVec = qb2nImu2YPRdeg(estNavSolution.q_b2n_l);
736   posVec = LLA2NED(estNavSolution.LLA_l);
737   ouputString = String(ouputString + String(currentSensor.dt, 4));
738   ouputString = String(ouputString + String(","));
739   ouputString = String(ouputString + String(currentSensor.f_b(0), 4));
740   ouputString = String(ouputString + String(","));
741   ouputString = String(ouputString + String(currentSensor.f_b(1), 4));
742   ouputString = String(ouputString + String(","));
743   ouputString = String(ouputString + String(currentSensor.f_b(2), 4));
744   ouputString = String(ouputString + String(","));
745   ouputString = String(ouputString + String(currentSensor.w_b2i_b(0), 4));
746   ouputString = String(ouputString + String(","));
747   ouputString = String(ouputString + String(currentSensor.w_b2i_b(1), 4));
748   ouputString = String(ouputString + String(","));
749   ouputString = String(ouputString + String(currentSensor.w_b2i_b(2), 4));
750   ouputString = String(ouputString + String(","));
751   ouputString = String(ouputString + String(oreintationVec.One, 4));
752   ouputString = String(ouputString + String(","));
753   ouputString = String(ouputString + String(oreintationVec.Two, 4));
754   ouputString = String(ouputString + String(","));
755   ouputString = String(ouputString + String(oreintationVec.Three, 4));
756   ouputString = String(ouputString + String(","));
757   ouputString = String(ouputString + String(estNavSolution.v_nWrtE_n_l(0), 4));
758   ouputString = String(ouputString + String(","));
759   ouputString = String(ouputString + String(estNavSolution.v_nWrtE_n_l(1), 4));
760   ouputString = String(ouputString + String(","));
761   ouputString = String(ouputString + String(estNavSolution.v_nWrtE_n_l(2), 4));
762   ouputString = String(ouputString + String(","));
763   ouputString = String(ouputString + String(posVec(0), 4));
764   ouputString = String(ouputString + String(","));
765   ouputString = String(ouputString + String(posVec(1), 4));
766   ouputString = String(ouputString + String(","));
767   ouputString = String(ouputString + String(posVec(2), 4));
768   ouputString = String(ouputString + String(","));
769   ouputString = String(ouputString + String(currentZUPT.zupt_l, 0));
770   ouputString = String(ouputString + String(","));
771   ouputString = String(ouputString + String(currentZUPT.sum_IMU, 0));
772   ouputString = String(ouputString + String("\n"));
773   return ouputString;
774 }
775
776 String printAndOutputMeasurementAndNavigationSolutionsLLA(IMUreadings
      currentSensor,
777                                                          zuptStruct currentZUPT,
778 estimationStruct
```

```
estNavSolution)
{

  String ouputString = "";

  ArrayMatrix<3, 1, double> posVec;
  ThreeElementStruct oreintationVec;
  oreintationVec = qb2nImu2YPRdeg(estNavSolution.q_b2n_l);
  ouputString = String(ouputString + String(currentSensor.dt, 4));
  ouputString = String(ouputString + String(","));
  ouputString = String(ouputString + String(currentSensor.f_b(0), 4));
  ouputString = String(ouputString + String(","));
  ouputString = String(ouputString + String(currentSensor.f_b(1), 4));
  ouputString = String(ouputString + String(","));
  ouputString = String(ouputString + String(currentSensor.f_b(2), 4));
  ouputString = String(ouputString + String(","));
  ouputString = String(ouputString + String(currentSensor.w_b2i_b(0), 4));
  ouputString = String(ouputString + String(","));
  ouputString = String(ouputString + String(currentSensor.w_b2i_b(1), 4));
  ouputString = String(ouputString + String(","));
  ouputString = String(ouputString + String(currentSensor.w_b2i_b(2), 4));
  ouputString = String(ouputString + String(","));
  ouputString = String(ouputString + String(oreintationVec.One, 4));
  ouputString = String(ouputString + String(","));
  ouputString = String(ouputString + String(oreintationVec.Two, 4));
  ouputString = String(ouputString + String(","));
  ouputString = String(ouputString + String(oreintationVec.Three, 4));
  ouputString = String(ouputString + String(","));
  ouputString = String(ouputString + String(estNavSolution.v_nWrtE_n_l(0), 4));
  ouputString = String(ouputString + String(","));
  ouputString = String(ouputString + String(estNavSolution.v_nWrtE_n_l(1), 4));
  ouputString = String(ouputString + String(","));
  ouputString = String(ouputString + String(estNavSolution.v_nWrtE_n_l(2), 4));
  ouputString = String(ouputString + String(","));
  ouputString = String(ouputString + String(estNavSolution.LLA_l(0) * r2d, 8));
  ouputString = String(ouputString + String(","));
  ouputString = String(ouputString + String(estNavSolution.LLA_l(1) * r2d, 8));
  ouputString = String(ouputString + String(","));
  ouputString = String(ouputString + String(estNavSolution.LLA_l(2), 4));
  ouputString = String(ouputString + String(","));
  ouputString = String(ouputString + String(currentZUPT.zupt_l, 0));
  ouputString = String(ouputString + String(","));
  ouputString = String(ouputString + String(currentZUPT.sum_IMU, 0));
  ouputString = String(ouputString + String("\n"));
  return ouputString;
}

```

```
String printAndOutputMeasurementAndNavigationSolutionsNEDAndLLA(IMUreadings
                                                                 currentSensor,
                                                                 zuptStruct
currentZUPT,
estimationStruct
estNavSolution)
{

  String ouputString = "";

  ArrayMatrix<3, 1, double> posVec;
  ThreeElementStruct oreintationVec;
  oreintationVec = qb2nImu2YPRdeg(estNavSolution.q_b2n_l);
  posVec = LLA2NED(estNavSolution.LLA_l);

  ouputString = String(ouputString + String(currentSensor.dt, 4));
  ouputString = String(ouputString + String(","));
  ouputString = String(ouputString + String(currentSensor.f_b(0), 4));
  ouputString = String(ouputString + String(","));
  ouputString = String(ouputString + String(currentSensor.f_b(1), 4));
  ouputString = String(ouputString + String(","));
  ouputString = String(ouputString + String(currentSensor.f_b(2), 4));
  ouputString = String(ouputString + String(","));
  ouputString = String(ouputString + String(currentSensor.w_b2i_b(0), 4));
  ouputString = String(ouputString + String(","));
  ouputString = String(ouputString + String(currentSensor.w_b2i_b(1), 4));
  ouputString = String(ouputString + String(","));
  ouputString = String(ouputString + String(currentSensor.w_b2i_b(2), 4));
  ouputString = String(ouputString + String(","));
  ouputString = String(ouputString + String(oreintationVec.One, 4));
  ouputString = String(ouputString + String(","));
  ouputString = String(ouputString + String(oreintationVec.Two, 4));
  ouputString = String(ouputString + String(","));
  ouputString = String(ouputString + String(oreintationVec.Three, 4));
  ouputString = String(ouputString + String(","));
  ouputString = String(ouputString + String(estNavSolution.v_nWrtE_n_l(0), 4));
  ouputString = String(ouputString + String(","));
  ouputString = String(ouputString + String(estNavSolution.v_nWrtE_n_l(1), 4));
  ouputString = String(ouputString + String(","));
  ouputString = String(ouputString + String(estNavSolution.v_nWrtE_n_l(2), 4));
  ouputString = String(ouputString + String(","));
  ouputString = String(ouputString + String(posVec(0), 4));
  ouputString = String(ouputString + String(","));
  ouputString = String(ouputString + String(posVec(1), 4));
  ouputString = String(ouputString + String(","));
  ouputString = String(ouputString + String(posVec(2), 4));
  ouputString = String(ouputString + String(","));
```

```
  ouputString = String(ouputString + String(currentZUPT.zupt_l, 0));
  ouputString = String(ouputString + String(","));
  ouputString = String(ouputString + String(currentZUPT.sum_IMU, 0));
  ouputString = String(ouputString + String(","));
  ouputString = String(ouputString + String(estNavSolution.LLA_l(0) * r2d, 8));
  ouputString = String(ouputString + String(","));
  ouputString = String(ouputString + String(estNavSolution.LLA_l(1) * r2d, 8));
  ouputString = String(ouputString + String(","));
  ouputString = String(ouputString + String(estNavSolution.LLA_l(2), 4));
  ouputString = String(ouputString + String("\n"));
  return ouputString;
}

String printAndOutputMeasurementAndNavigationSolutionsShort(IMUreadings
currentSensor, zuptStruct currentZUPT, estimationStruct estNavSolution)
{

  String ouputString = "";

  ArrayMatrix<3, 1, double> posVec;
  ThreeElementStruct oreintationVec;
  oreintationVec = qb2nImu2YPRdeg(estNavSolution.q_b2n_l);
  posVec = LLA2NED(estNavSolution.LLA_l);

  ouputString = String(ouputString + String(currentSensor.dt, 4));
  ouputString = String(ouputString + String(","));
  ouputString = String(ouputString + String(currentSensor.f_b(0), 3));
  ouputString = String(ouputString + String(","));
  ouputString = String(ouputString + String(currentSensor.f_b(1), 3));
  ouputString = String(ouputString + String(","));
  ouputString = String(ouputString + String(currentSensor.f_b(2), 3));
  ouputString = String(ouputString + String(","));
  ouputString = String(ouputString + String(currentSensor.w_b2i_b(0), 3));
  ouputString = String(ouputString + String(","));
  ouputString = String(ouputString + String(currentSensor.w_b2i_b(1), 3));
  ouputString = String(ouputString + String(","));
  ouputString = String(ouputString + String(currentSensor.w_b2i_b(2), 3));
  ouputString = String(ouputString + String(","));
  ouputString = String(ouputString + String(posVec(0), 2));
  ouputString = String(ouputString + String(","));
  ouputString = String(ouputString + String(posVec(1), 2));
  ouputString = String(ouputString + String(","));
  ouputString = String(ouputString + String(posVec(2), 2));
  ouputString = String(ouputString + String(","));
  ouputString = String(ouputString + String(currentZUPT.zupt_l, 0));
  ouputString = String(ouputString + String("\n"));
  return ouputString;
```

```
}

String printAndOutputMeasurementAndNavigationSolutionsSplit(IMUreadings
currentSensor, zuptStruct currentZUPT, estimationStruct estNavSolution)
{

  String ouputString = "";

  ArrayMatrix<3, 1, double> posVec;
  ThreeElementStruct oreintationVec;
  oreintationVec = qb2nImu2YPRdeg(estNavSolution.q_b2n_l);
  posVec = LLA2NED(estNavSolution.LLA_l);

  ouputString = String(ouputString + String(currentSensor.dt, 4));
  ouputString = String(ouputString + String(","));
  ouputString = String(ouputString + String(currentSensor.f_b(0), 3));
  ouputString = String(ouputString + String(","));
  ouputString = String(ouputString + String(currentSensor.f_b(1), 3));
  ouputString = String(ouputString + String(","));
  ouputString = String(ouputString + String(currentSensor.f_b(2), 3));
  ouputString = String(ouputString + String(","));
  ouputString = String(ouputString + String(currentSensor.w_b2i_b(0), 3));
  ouputString = String(ouputString + String(","));
  ouputString = String(ouputString + String(currentSensor.w_b2i_b(1), 3));
  ouputString = String(ouputString + String(","));
  ouputString = String(ouputString + String(currentSensor.w_b2i_b(2), 3));
  ouputString = String(ouputString + String(","));

  ouputString = String(ouputString + String(oreintationVec.One, 2));
  ouputString = String(ouputString + String(","));
  ouputString = String(ouputString + String("@"));
  ouputString = String(ouputString + String(oreintationVec.Two, 2));
  ouputString = String(ouputString + String(","));

  ouputString = String(ouputString + String(oreintationVec.Three, 2));
  ouputString = String(ouputString + String(","));
  ouputString = String(ouputString + String(estNavSolution.v_nWrtE_n_l(0), 2));
  ouputString = String(ouputString + String(","));
  ouputString = String(ouputString + String(estNavSolution.v_nWrtE_n_l(1), 2));
  ouputString = String(ouputString + String(","));
  ouputString = String(ouputString + String(estNavSolution.v_nWrtE_n_l(2), 2));
  ouputString = String(ouputString + String(","));
  ouputString = String(ouputString + String(posVec(0), 1));
  ouputString = String(ouputString + String(","));
  ouputString = String(ouputString + String(posVec(1), 1));
  ouputString = String(ouputString + String(","));
  ouputString = String(ouputString + String(posVec(2), 1));
```

```
  ouputString = String(ouputString + String(","));
  ouputString = String(ouputString + String(currentZUPT.zupt_l, 0));
  ouputString = String(ouputString + String(","));
  ouputString = String(ouputString + String(currentZUPT.sum_IMU, 0));
  ouputString = String(ouputString + String("&"));
  return ouputString;
}
```

Utility Library

```
#include "INS_lib.h"

ThreeElementStruct b321tang(ArrayMatrix<3,3,double> b) {
   // [roll,pitch,yaw] = b321tang(b)
   // Input:
   //       b    =  coordinate transformation matrix obtained by 321 rotation
   //               sequence, that is first rotate about 3-axis by 'yaw' angle,
   //               next rotate aboue 2-axis by 'pitch' angle, and finally
   //               rotate about 1-axis by 'roll' angle.
   // Output:
   //       roll = 'roll' angle about 1-axis (x-axis), deg,
   //       pitch = 'pitch' angle about 2-axis (y-axis), deg,
   //       yaw  =  'yaw' angle about 3-axis (z-axis), deg,
   //
   //  Note that there is an ambiguity in roll and yaw when pitch is +/- 90 deg.
   ThreeElementStruct oreintation_in_degree;
   double roll_deg, aux, pitch_deg, yaw_deg;

   roll_deg = atan2(b(1,2), b(2,2)) * r2d;
   aux = sqrt(1.0 - pow(b(0,2),2) );
   pitch_deg = atan2(-b(0,2), aux) * r2d;
   yaw_deg  = atan2(b(0,1), b(0,0)) * r2d;
   oreintation_in_degree = (ThreeElementStruct){.One = roll_deg, .Two =
       pitch_deg, .
Three = yaw_deg};
   return oreintation_in_degree;
}

TwoElementStruct Ce2n_toLLA(ArrayMatrix<3,3,double> C_e2n){

TwoElementStruct Lon_Lat_deg;

double D13, D23, Lon_deg, D31, D32, D33, aux, Lat_deg;

```

```
D13 = C_e2n(2,0);
D23 = C_e2n(2,1);

Lon_deg = atan2(-D23,-D13)*180.0/M_PI;

D31 = C_e2n(0,2);
D32 = C_e2n(1,2);
D33 = C_e2n(2,2);

aux = sqrt(D31*D31 + D32*D32);

Lat_deg = atan2(-D33, aux)*180.0/M_PI;

Lon_Lat_deg.One = Lon_deg;
Lon_Lat_deg.Two = Lat_deg;

return Lon_Lat_deg;
}

int sgn(double v) {
  if (v < 0) {return -1;}
  if (v > 0) {return 1;}
  return 0;
}

double vectorNorm(ArrayMatrix<4,1,double> v){
   return sqrt(v(0)*v(0) + v(1)*v(1) + v(2)*v(2) + v(3)*v(3));
}

double vectorNorm(ArrayMatrix<3,1,double> v){
   return sqrt(v(0)*v(0) + v(1)*v(1) + v(2)*v(2));
}

double vectorNorm(ArrayMatrix<2,1,double> v){
   return sqrt(v(0)*v(0) + v(1)*v(1));
}

double vectorNorm(ArrayMatrix<1,4,double> v){
   return sqrt(v(0,0)*v(0,0) + v(0,1)*v(0,1) + v(0,2)*v(0,2) + v(0,3)*v(0,3));
}

double vectorNorm (ArrayMatrix<1,3,double> v){
   return sqrt(v(0,0)*v(0,0) + v(0,1)*v(0,1) + v(0,2)*v(0,2));
}

double vectorNorm(ArrayMatrix<1,2,double> v){
   return sqrt(v(0,0)*v(0,0) + v(0,1)*v(0,1));
```

```
}

SevenElementAndVectorStruct dcos2quat(ArrayMatrix<3,3,double> d){
   // ============================================================================
   // convert 3x3 direction cosine matrix d to 4-vector quaternion q.
   // If q maps frame A to frame B, then d maps vectors in
   // frame A to frame B. q & d represent the orientation of B with
   // respect to A.
   //  NOTE: In theory, the resulting quaternion should be a unit
   //        quaternion. But normalization is used.
   //
   // d = 3x3 direction cosine matrix mapping frame A to frame B.
   //
   // q = quaternion state vector (4 elements), representing the
   //    orientation of frame B with respect to frame A. In terms of
   //    the Euler rotation of theta radians about the unit vector e
   //    which maps frame A to frame B:
   //
   //      q(1:3) = e(1:3)*sin(theta/2) = V = vector part of q
   //      q(4)   =       cos(theta/2) = S = scalar part of q
   //
   // any orientation has two associated quaternions, one of which is -1*the
   // other. The canonical one is the one for which q(4) is non-negative,
   // and corresponds to an Euler angle in the range 0 - pi.
   // the variable "fix", below, is used to negate the quaternion on
   // the fly, if necessary.
   //
   // Reference: eq 12-14, p. 415, Wertz, "Spacecraft Attitude
   //                                  Determination and Control"
   //
   // ============================================================================
   SevenElementAndVectorStruct output;
   ArrayMatrix<4,1,double> q;
   double a1,a2,a3,a4,b,ei, inv_4ei, fix;

   q.Fill(0);
   // select combination of diagonal dir cos elements that give largest
   // denominator in initial quaternion equations to avoid numerical problems
        a1 = d(0,0) + d(1,1) + d(2,2);
        a2 = d(0,0) - d(1,1) - d(2,2);
        a3 = -d(0,0) + d(1,1) - d(2,2);
        a4 = -d(0,0) - d(1,1) + d(2,2);

        b  = max(max(max(a1, a2), a3), a4 );

        ei = 0.5*sqrt(1 + b); // this is equation 12-14a from Wertz p. 415
        inv_4ei = 0.25/ei;
```

```
128     // now compute initial quaternions using formulas which are appropriate given
129     // the denominator element just selected
130     if (b == a1){
131          q(0) = (d(1,2) - d(2,1))*inv_4ei;
132          q(1) = (d(2,0) - d(0,2))*inv_4ei;
133          q(2) = (d(0,1) - d(1,0))*inv_4ei;
134          q(3) = ei;
135     }
136     else if (b == a2) {
137          q(0) = ei;
138          q(1) = (d(0,1) + d(1,0))*inv_4ei;
139          q(2) = (d(2,0) + d(0,2))*inv_4ei;
140          q(3) = (d(1,2) - d(2,1))*inv_4ei;
141     }
142     else if (b == a3) {
143          q(0) = (d(0,1) + d(1,0))*inv_4ei;
144          q(1) = ei;
145          q(2) = (d(1,2) + d(2,1))*inv_4ei;
146          q(3) = (d(2,0) - d(0,2))*inv_4ei;
147     }
148     else {
149          q(0) = (d(2,0) + d(0,2))*inv_4ei;
150          q(1) = (d(2,1) + d(1,2))*inv_4ei;
151          q(2) = ei;
152          q(3) = (d(0,1) - d(1,0))*inv_4ei;
153     }
154     // ensure q is canonical
155     fix = sgn(q(3));
156     if (fix==0){
157        fix=1;
158     }
159
160     q(0) = fix*q(0);
161     q(1) = fix*q(1);
162     q(2) = fix*q(2);
163     q(3) = fix*q(3);
164
165      if (abs(vectorNorm(q)) > 1e-40){
166         q = q/vectorNorm(q);
167     }
168      else{
169         q = {0,
170        0,
171        0,
172        1
173        };
174     }
```

```
175     output.OneVector = q;
176     output.Two = a1;
177     output.Three = a2;
178     output.Four = a3;
179     output.Five = a4;
180     output.Six = b;
181     output.Seven = ei;
182     return output;
183 }
184 
185 ArrayMatrix<3,3,double> skew(ArrayMatrix<3,1,double> x){
186 
187 ArrayMatrix<3,3,double> S_temp;
188 S_temp = {0, -x(2), x(1),
189         x(2),0,-x(0),
190        -x(1),x(0),0};
191     return (S_temp);
192 }
193 
194 ArrayMatrix<3,3,double> trueCosine(ArrayMatrix<3,1,double> vArray){
195 
196     // ------------------------------------------------------------------------
197     ArrayMatrix<3,3,double> I3,C,Ex,EEt;
198     ArrayMatrix<3,1,double> e_th;
199     double norm_v, cosTh, sinTh;
200     I3 << 1,0,0,
201           0,1,0,
202           0,0,1;
203     norm_v = vectorNorm(vArray);
204     if (norm_v < 1e-15){
205     C = I3;
206     return C;
207     }
208 
209     e_th = vArray/norm_v;
210     EEt  = (e_th*(~e_th));
211     Ex   = skew(e_th);
212     cosTh = cos(norm_v);
213     sinTh = sin(norm_v);
214     // ------------------------------------------------------------------------
215     C = I3*cosTh + (e_th*(~e_th))*(1-cosTh) - skew(e_th)*sinTh;
216     // ------------------------------------------------------------------------
217     return C;
218 }
219 
220 ArrayMatrix<3,1,double> earthRateInBody(double roll,double pitch,double
      yaw,double
```

```
221 latitude){
222
223    ArrayMatrix<3,3,double> C_roll, C_pitch, C_yaw, C_n2b;
224    ArrayMatrix<3,1,double> w_e2i_b, w_e2i_n, temp_vector;
225    double Omega = 7.292115060085166e-005, scl, lat_rad, roll_rad, pitch_rad,
           yaw_rad;
226
227    scl = M_PI/180.0;
228
229    lat_rad  = latitude*scl;
230    roll_rad = roll*scl;
231    pitch_rad = pitch*scl;
232    yaw_rad  = yaw*scl;
233
234    w_e2i_n = {cos(lat_rad),
235            0,
236            -sin(lat_rad)};
237
238    w_e2i_n = w_e2i_n*Omega;
239
240    temp_vector = { roll_rad,
241               0,
242               0
243               };
244    C_roll = trueCosine(temp_vector);
245    temp_vector = { 0,
246               pitch_rad,
247               0
248               };
249    C_pitch = trueCosine(temp_vector);
250    temp_vector = { 0,
251               0,
252               yaw_rad
253               };
254    C_yaw   = trueCosine(temp_vector);
255
256    C_n2b = C_roll * C_pitch * C_yaw;
257
258    w_e2i_b = C_n2b * w_e2i_n;
259    return w_e2i_b;
260 }
261
262 ArrayMatrix<3,3,double> euler1(double i, double a){
263    // function m=euler1(i,a)
264    // Calculates elementary rotation matrix corresponding to
265    // rotation thru angle a (in radians) about axis i.
266    // Copied from John L. Junkins presentation AAS 89-060
```

```
267     // m (3,3) maps the vector in original coordinates to new coordinates
268     //        i.e, Vnew = m * Voriginal
269     double ca, sa;
270     ArrayMatrix<3,3,double> m;
271     ca=cos(a);
272     sa=sin(a);
273 
274     if (i==1){ // rotation about x-axis
275     m={1,  0, 0,
276        0, ca, sa,
277        0,-sa, ca};
278     }
279     if (i==2){ // rotation about y-axis
280     m={ca, 0,-sa,
281        0, 1,  0,
282        sa, 0, ca};
283     }
284     if (i==3){ // rotation about z-axis
285     m={ ca, sa, 0,
286        -sa, ca, 0,
287        0,  0, 1};
288     }
289     return m;
290 }
291 
292 ArrayMatrix<3,1,double> gravityModel(double Lat, double P, double a, double e2) {
293 
294     double GM, c20, Latc, P2, a2, ax, GMoverP2, sLatc, cLatc, g_n, g_d, Alpha,
            cAlpha,
295  sAlpha, g_N, g_D;
296     ArrayMatrix<3,1,double> g;
297 
298     GM  = 3.986005e14; // m^3/s^2 GravitationalConstant
299 
300     c20 = -sqrt(5)*4.8416685e-4;
301     Latc = atan((1-e2)*tan(Lat));
302 
303     // geocentric
304     P2 = P*P;
305     a2 = a*a;
306     ax = 3*c20*a2/P2;
307     GMoverP2 = GM/P2;
308     sLatc = sin(Latc);
309     cLatc = cos(Latc);
310     g_n = -GMoverP2 * ax*sLatc*cLatc;
311     g_d =  GMoverP2 * (1 + ax/2.0*(3*pow(sLatc,2)-1));
312 
```

```
313     g_n = -1.0 * g_n; // match Savage's model ?
314
315     // geodetic
316     Alpha = Lat - Latc;
317     cAlpha = cos(Alpha);
318     sAlpha = sin(Alpha);
319     g_N = g_n*cAlpha + g_d*sAlpha;
320     g_D = -g_n*sAlpha + g_d*cAlpha;
321
322     g = {g_N,
323         0,
324         g_D};
325
326     return g;
327 }
328
329 ArrayMatrix<4,1,double> q_canonicalize(ArrayMatrix<4,1,double> q_in){
330
331     ArrayMatrix<4,1,double> q_out;
332     q_out = q_in;
333     //k = find(q_in(4,:) < 0);
334     if (q_in(3) < 0){
335        q_out = q_out*(-1);
336     }
337     return q_out;
338 }
339
340 ArrayMatrix<4,1,double> Imu2YPRdeg_to_qb2n(double roll_deg, double pitch_deg,
        double
341 yaw_deg){
342     //
343     // Rotation 3-2-1 from nav. to b-frame
344     // Yaw (Heading) -> Pitch -> Roll
345     SevenElementAndVectorStruct tempStruct;
346     ArrayMatrix<4,1,double> q_b2n, q;
347     ArrayMatrix<3,3,double> Cz, Cy, Cx, C_n2b;
348     double en;
349
350     en = 1;
351
352     q_b2n = {0,
353           0,
354           0,
355           0};
356
357     Cz = euler1(3,yaw_deg*d2r);
358     Cy = euler1(2,pitch_deg*d2r);
```

```
359    Cx = euler1(1,roll_deg*d2r);
360    C_n2b = Cx*Cy*Cz;
361    tempStruct = dcos2quat((~C_n2b));
362    q = tempStruct.OneVector;
363    q_b2n = q_canonicalize(q);
364
365    return q_b2n;
366 }
367
368 ArrayMatrix<4,1,double> theta2quat(ArrayMatrix<3,1,double> theta){
369    // theta in rad.
370    ArrayMatrix<3,1,double> x, rotAxs;
371    ArrayMatrix<4,1,double> q;
372    double xNorm, sAnglDiv2, cAnglDiv2;
373    x = theta;
374    xNorm = sqrt(x(0)*x(0)+x(1)*x(1)+x(2)*x(2));
375    rotAxs.Fill(0);
376    if (xNorm>0){
377       rotAxs = {x(0)/xNorm,
378              x(1)/xNorm,
379              x(2)/xNorm};
380    }
381    sAnglDiv2 = sin(xNorm/2);
382    cAnglDiv2 = cos(xNorm/2);
383    q = { rotAxs(0)* sAnglDiv2,
384         rotAxs(1)* sAnglDiv2,
385         rotAxs(2)* sAnglDiv2,
386         cAnglDiv2};
387    return q;
388 }
389
390 ArrayMatrix<4,1,double> q_mult(ArrayMatrix<4,1,double> a,
     ArrayMatrix<4,1,double> b){
391    //
392    //    This function performs:  q = q  * q    where q is a quaternion
393    //                              c   a    b
394    // ------------------------------------------------------------------------
395    double a1,a2,a3,a4,b1,b2,b3,b4,d1,d2,d3,d4;
396    ArrayMatrix<4,1,double> c, c_out;
397
398     a1 = a(0);
399     a2 = a(1);
400     a3 = a(2);
401     a4 = a(3);
402
403     b1 = b(0);
404     b2 = b(1);
```

```
405     b3 = b(2);
406     b4 = b(3);
407 
408    d1 =  a4*b1 - a3*b2 + a2*b3 + a1*b4 ;
409    d2 =  a3*b1 + a4*b2 - a1*b3 + a2*b4 ;
410    d3 = - a2*b1 + a1*b2 + a4*b3 + a3*b4 ;
411    d4 = - a1*b1 - a2*b2 - a3*b3 + a4*b4 ;
412 
413    c = {d1,
414        d2,
415       d3,
416       d4};
417 
418    c_out = q_canonicalize( c );
419    return c_out;
420 }
421 
422 ArrayMatrix<4,1,double> integVel_into_q_e2N(ArrayMatrix<4,1,double> q_e2N0,
     double h,
423 ArrayMatrix<3,1,double> Vn, double dt){
424 
425    double a;
426    ArrayMatrix<3,1,double> dth;
427    ArrayMatrix<4,1,double> dq, q_e2N;
428    a = 6378137.0 + h;
429    dth = {Vn(1)/a,
430         -Vn(0)/a,
431          0};
432    dth = dth*dt;
433    dq = theta2quat(dth);
434    q_e2N.Fill(0);
435    q_e2N = q_e2N0;
436 
437    return q_e2N;
438 }
439 
440 ArrayMatrix<3,1,double> LLA2rWrtEinE(ArrayMatrix<3,1,double> LLA){
441 
442     double EarthSemiMajorAxis = 6378137.0, EarthEccentricitySq =
443 6.694380004260835e-003, cLon, sLon, cLat, sLat, h, R_N;
444    ArrayMatrix<3,1,double> r_wrtE_e;
445 
446    cLon  = cos(LLA(0));
447    sLon  = sin(LLA(0));
448    cLat  = cos(LLA(1));
449    sLat  = sin(LLA(1));
450 
```

```
451     R_N  = EarthSemiMajorAxis/sqrt(1-EarthEccentricitySq*pow(sLat,2));
452
453     h = LLA(2);
454
455     r_wrtE_e = {(R_N + h)*cLat*cLon,
456             (R_N + h)*cLat*sLon,
457             (R_N*(1-EarthEccentricitySq) + h)*sLat};
458
459     return r_wrtE_e;
460 }
461
462 ArrayMatrix<3,1,double> LLA2rWrtEinE(ArrayMatrix<3,1,double> LLA, double
463 EarthSemiMajorAxis, double EarthEccentricitySq){
464
465     double cLon, sLon, cLat, sLat, h, R_N;
466     ArrayMatrix<3,1,double> r_wrtE_e;
467
468     cLon  = cos(LLA(0));
469     sLon  = sin(LLA(0));
470     cLat  = cos(LLA(1));
471     sLat  = sin(LLA(1));
472
473     R_N  = EarthSemiMajorAxis/sqrt(1-EarthEccentricitySq*pow(sLat,2));
474
475     h = LLA(2);
476
477     r_wrtE_e = {(R_N + h)*cLat*cLon,
478             (R_N + h)*cLat*sLon,
479             (R_N*(1-EarthEccentricitySq) + h)*sLat};
480
481     return r_wrtE_e;
482 }
483
484 ArrayMatrix<4,1,double> lonLatDegTo_q_e2N(double lonDeg, double latDeg){
485
486     ArrayMatrix<4,1,double> q_e2N;
487     ArrayMatrix<3,3,double> C_e2N;
488     double cLon, sLon, cLat, sLat;
489     SevenElementAndVectorStruct tempMtx;
490     q_e2N.Fill(0);
491
492     cLon = cosd(lonDeg);
493     sLon = sind(lonDeg);
494     cLat = cosd(latDeg);
495     sLat = sind(latDeg);
496
497     C_e2N = {sLat*cLon*(-1), sLat*sLon*(-1), cLat,
```

```
498             sLon*(-1),       cLon,      0,
499           cLon*cLat*(-1), cLat*sLon*(-1), sLat*(-1)};
500    tempMtx = dcos2quat(C_e2N);
501     q_e2N = tempMtx.OneVector;
502    return q_e2N;
503 }
504
505 ArrayMatrix<3,3,double> quat2dcos(ArrayMatrix<4,1,double> q){
506    // compute direction cosine matrix from quaternion
507    //-----------------------------------------------------
508    double q1,q2,q3,q4,a11,a12,a13,a21,a22,a23,a31,a32,a33;
509    ArrayMatrix<3,3,double> aa;
510    q1 = q(0,0);
511    q2 = q(1,0);
512    q3 = q(2,0);
513    q4 = q(3,0);
514
515    a11   = q1*q1 - q2*q2 - q3*q3 + q4*q4;
516    a12   = (q1*q2 + q3*q4)*2;
517    a13   = (q1*q3 - q2*q4)*2;
518    a21   = (q1*q2 - q3*q4)*2;
519    a22   = -q1*q1 + q2*q2 - q3*q3 + q4*q4;
520    a23   = (q2*q3 + q1*q4)*2;
521    a31   = (q1*q3 + q2*q4)*2;
522    a32   = (q2*q3 - q1*q4)*2;
523    a33   = -q1*q1 - q2*q2 + q3*q3 + q4*q4;
524
525    aa = {a11,a12,a13,
526          a21,a22,a23,
527          a31,a32,a33};
528    return aa;
529 }
530
531 TwoElementStruct q_e2N_toLonLatDeg(ArrayMatrix<4,1,double> q_e2N){
532
533    TwoElementStruct output;
534    double lonDeg, latDeg;
535    ArrayMatrix<3,3,double> C_e2n;
536
537    lonDeg = 0;
538    latDeg = 0;
539     C_e2n = quat2dcos(q_e2N);
540     output = Ce2n_toLLA(C_e2n);
541
542    return output;
543 }
544
```

```
ArrayMatrix<4,1,double> q_inv(ArrayMatrix<4,1,double> q_in ){
   ArrayMatrix<4,1,double> q_out;
   q_out = q_in;
   q_out(0) = q_out(0) * (-1);
   q_out(1) = q_out(1) * (-1);
   q_out(2) = q_out(2) * (-1);
   return q_out;
}

ThreeElementStruct qb2nImu2YPRdeg(ArrayMatrix<4,1,double> q_b2n, ArrayMatrix<3,3,
double> C_bRef2b){
   // Rotation 3-2-1 from nav. to b-frame
   // Yaw (Heading) -> Pitch -> Roll
   // If specified C_bRef2b then
   // Rotation 3-2-1 from nav. to bRef-frame where
   //  C_bRef2b is constant offset from b-frame
   ThreeElementStruct output;
   double roll_deg, pitch_deg, yaw_deg, nq;
   ArrayMatrix<4,1,double> q_b2n_use;
   ArrayMatrix<3,3,double> C_bRef2n, C_b2n;

   roll_deg = 0;
   pitch_deg = roll_deg;
   yaw_deg  = roll_deg;

    nq = vectorNorm(q_b2n);
    q_b2n_use = q_b2n;
    if (nq>0.0){
       q_b2n_use = q_b2n_use/nq;
   }
    C_b2n = quat2dcos(q_b2n_use);
    C_bRef2n = C_b2n * C_bRef2b;
    output = b321tang((~C_bRef2n));

   return output;
}

ThreeElementStruct qb2nImu2YPRdeg(ArrayMatrix<4,1,double> q_b2n){

   // Rotation 3-2-1 from nav. to b-frame
   // Yaw (Heading) -> Pitch -> Roll
   // If specified C_bRef2b then
   // Rotation 3-2-1 from nav. to bRef-frame where
   //  C_bRef2b is constant offset from b-frame
   ThreeElementStruct output;
   double roll_deg, pitch_deg, yaw_deg, nq;
   ArrayMatrix<4,1,double> q_b2n_use;
```

```
592    ArrayMatrix<3,3,double> C_bRef2n, C_b2n, C_bRef2b;
593
594    C_bRef2b << 1,0,0,
595            0,1,0,
596            0,0,1;
597
598    roll_deg = 0;
599    pitch_deg = roll_deg;
600    yaw_deg  = roll_deg;
601
602     nq = vectorNorm(q_b2n);
603     q_b2n_use = q_b2n;
604     if (nq>0.0){
605        q_b2n_use = q_b2n_use/nq;
606    }
607     C_b2n = quat2dcos(q_b2n_use);
608     C_bRef2n = C_b2n * C_bRef2b;
609     output = b321tang((~C_bRef2n));
610
611    return output;
612 }
613
614
615 TwoVectorStruct qintegrator(ArrayMatrix<4,1,double> q, ArrayMatrix<3,1,double>
       dTheta,
616  ArrayMatrix<3,1,double> deltaPrev, int flag) {
617    // ------------------------------------------------------
618    // Input :
619    // q        (4 x 1) initial quaternion
620    // dTheta   (3 x 1) current angle increment
621    // deltaPrev (3 x 1) increment used in propagating quaternion
622    // Output :
623    // q_o      (4 x 1) propagated quaternion
624    // delta    (3 x 1) increment for the next cycle
625    // ------------------------------------------------------
626
627    TwoVectorStruct output;
628    ArrayMatrix<4,1,double> q0,q_o, q1, q2;
629    ArrayMatrix<3,1,double> delta, e;
630    ArrayMatrix<4,1,double> dq0;
631    ArrayMatrix<4,4,double> E,dTh,Id4;
632    ArrayMatrix<1,1,double> tempValue;
633    double nrm_dth;
634
635    delta.Fill(0);
636
637    switch (flag) {
```

```
638       case 0 :
639          nrm_dth = vectorNorm(dTheta);
640
641          if (nrm_dth < 1e-40) {
642             e.Fill(0);}
643          else{
644             e = dTheta/nrm_dth;
645          }
646          dq0 = {e(0)*sin(nrm_dth/2.0),
647                e(1)*sin(nrm_dth/2.0),
648                e(2)*sin(nrm_dth/2.0),
649                cos(nrm_dth/2.0)};
650          q0 = q_mult(q, dq0);
651          q_o = q0;
652          break;
653       case 1 :
654          nrm_dth = vectorNorm(dTheta);
655          if (nrm_dth < 1e-40){
656             e.Fill(0);}
657          else{
658             e = dTheta/nrm_dth;
659          }
660          E = {0, e(2), -e(1), e(0),
661             -e(2), 0, e(0), e(1),
662             e(1), -e(0), 0, e(2),
663             -e(0), -e(1), -e(2), 0};
664          dTh = E*nrm_dth;
665          q1 = q + dTh*q*0.5;
666          q_o = q1;
667          break;
668       case 2 :
669          nrm_dth = vectorNorm(dTheta);
670          if (nrm_dth < 1e-40){
671             e.Fill(0);}
672          else{
673             e = dTheta/nrm_dth;
674          }
675          E = {0, e(2), -e(1), e(0),
676             -e(2), 0, e(0), e(1),
677             e(1), -e(0), 0, e(2),
678             -e(0), -e(1), -e(2), 0};
679          Id4 = {1, 0, 0, 0,
680                0, 1, 0, 0,
681                0, 0, 1, 0,
682                0, 0, 0, 1};
683          q2 = (Id4*cos(nrm_dth/2.0)+E*sin(nrm_dth/2.0))*q;
684          q_o = q2;
```

```
685         break;
686       default:
687         break;
688    }
689    tempValue = ((~q_o)*q_o);
690    q_o = q_o * (1.5 - tempValue(0,0)*0.5);
691    output.OneVector = q_o;
692    output.TwoVector = delta;
693
694    return output;
695 }
696
697 ArrayMatrix<3,1,double> quatRot(ArrayMatrix<4,1,double> q,
       ArrayMatrix<3,1,double> V){
698    //    This function performs: W_vec = q * V where
699    //    q is a quaternions (4, N) (actully matrix)
700    //    V is vector (3 x N)
701    //    W_vec is matrix (3, N);
702    //    The i-th column in W_vec(:,i) is rotated by q(:,i) vector v
703    //! ============================================================================
704    double q0, q1, q2, q3, ax0, ax12, ax30, ax23, ax20, ax13, ax10, Q11, Q12, Q13,
          Q21
705 , Q22, Q23, Q31, Q32, Q33, w1, w2, w3;
706    ArrayMatrix<3,1,double> W_vec;
707    ArrayMatrix<1,3,double> X;
708    q0 = q(3,0);
709    q1 = q(0,0);
710    q2 = q(1,0);
711    q3 = q(2,0);
712
713    ax0  = 2*pow(q0,2);
714    ax12 = 2*q1*q2;
715    ax30 = 2*q3*q0;
716    ax23 = 2*q2*q3;
717    ax20 = 2*q2*q0;
718    ax13 = 2*q1*q3;
719    ax10 = 2*q1*q0;
720
721    Q11 = ax0-1+ 2*pow(q1,2);
722    Q12 = ax12 + ax30;
723    Q13 = ax13 - ax20;
724
725    Q21 = ax12 - ax30;
726    Q22 = ax0 - 1 + 2*pow(q2,2);
727    Q23 = ax23 + ax10;
728
729    Q31 = ax13 + ax20;
```

```
730     Q32 = ax23 - ax10;
731     Q33 = ax0 - 1 + 2*pow(q3,2);
732 
733     //w1 = [Q11(:) Q12(:) Q13(:)] * v;
734     //w2 = [Q21(:) Q22(:) Q23(:)] * v;
735     //w3 = [Q31(:) Q32(:) Q33(:)] * v;
736 
737     X = ~V;
738     w1 = Q11* X(0,0) + Q12* X(0,1) + Q13*X(0,2);
739     w2 = Q21* X(0,0) + Q22* X(0,1) + Q23*X(0,2);
740     w3 = Q31* X(0,0) + Q32* X(0,1) + Q33*X(0,2);
741 
742     W_vec = {w1,
743         w2,
744         w3};
745 
746     return W_vec;
747 }
748 
749 ArrayMatrix<3,1,double> quatToAngl(ArrayMatrix<4,1,double> qArray){
750 
751     ArrayMatrix<3,1,double> thetaArray, x, theta;
752     ArrayMatrix<4,1,double> q;
753     double angl, xNorm;
754 
755     thetaArray.Fill(0);
756 
757     q = qArray;
758     x(0) = q(0);
759     x(1) = q(1);
760     x(2) = q(2);
761     //x = q(1:3);
762     xNorm = vectorNorm(x);
763     theta.Fill(0);
764     if (xNorm>0){
765        angl = atan2(xNorm, q(3))*2;
766        theta = x/xNorm*angl;
767     }
768     thetaArray = theta;
769 
770     return thetaArray;
771 }
772 
773 ArrayMatrix<3,1,double> crossProduct(ArrayMatrix<3,1,double> v_A,
        ArrayMatrix<3,1,
774 double> v_B) {
775     ArrayMatrix<3,1,double> c_P;
```

```
776     c_P(0) = v_A(1) * v_B(2) - v_A(2) * v_B(1);
777     c_P(1) = -(v_A(0) * v_B(2) - v_A(2) * v_B(0));
778     c_P(2) = v_A(0) * v_B(1) - v_A(1) * v_B(0);
779     return c_P;
780 }
```

Appendix B

MATLAB Codes

B.1 ZUPT-aided INS With Sensor Fusion

B.1.1 The Main Script

```
clc;
clear;
close all;

progress_bar = waitbar(0,'Initialization...');
tic

currDir = pwd;
datasetDir = [currDir, '\dataset','\2021_11_16'];

IMU_select = 11;
filename = 'exp1600';
if IMU_select == 1
    data_filename = [filename '_VN'];
elseif IMU_select == 11
    data_filename = [filename '_L'];
elseif IMU_select == 12
    data_filename = [filename '_R'];
end

addpath([currDir, '\lib\INS_lib']);
```

```
addpath([currDir, '\lib\EKF_lib']);
addpath([currDir, '\lib\utility_lib']);
addpath([currDir, '\lib\plot_lib']);
addpath([currDir, '\lib\temp_com_lib']);
addpath([currDir, '\lib\Stance Phase Detectors']);
addpath(datasetDir);
load([data_filename '.mat']);

if isfile([datasetDir '\' data_filename '_GT.mat'])
    GT_file_exit = 1;
    load([data_filename '_GT.mat']);
else
    GT_file_exit = 0;
end

figDspl = [];
dspl.enblCov = 1;
simdata=settings_constructed_data(IMU_select);
d2r = pi/180;
r2d = 180/pi;
% gravity
g = GravityConstant();

u = u(:,100:end); % elimintate random initial readings from Lab-On-Shoe
u(8,:) = u(8,:) - u(8,1);

IMU_dt = floor(mean(1./u(11,100:200)));

% origin of the navigation frame
LLA_origin = [simdata.longitude; simdata.latitude; simdata.altitude];

% set initial position
if GT_file_exit
    ini_pos = est_GT.pos(:,1);
    %    t_ini = timeMismatchFromAccel(u,est_GT);
    yaw_Deg = initialYaw(est_GT);
    if IMU_select == 1
        ini_yaw_table = table2struct(readtable('initial_yaw_VN.xlsx'));
        yaw_Deg = 0;
        ini_time_table = table2struct(readtable('initial_time_VN.xlsx'));
        t_ini = ini_time_table.(filename);
        if isnan(t_ini)
            t_ini = timeMismatchFromAccel(u,est_GT);
        end
    elseif IMU_select == 11
        ini_yaw_table = table2struct(readtable('initial_yaw_L.xlsx'));
        yaw_Deg = ini_yaw_table.(filename);
```

```
        ini_time_table = table2struct(readtable('initial_time_L.xlsx'));
        t_ini = ini_time_table.(filename);
        if isnan(t_ini)
            t_ini = timeMismatchFromAccel(u,est_GT);
        end
    elseif IMU_select == 12
        %         yaw_Deg = yaw_Deg;
        ini_yaw_table = table2struct(readtable('initial_yaw_R.xlsx'));
        yaw_Deg = ini_yaw_table.(filename);
        ini_time_table = table2struct(readtable('initial_time_R.xlsx'));
        t_ini = ini_time_table.(filename);
        if isnan(t_ini)
            t_ini = timeMismatchFromAccel(u,est_GT);
        end
    end
else
    ini_pos = [0;0;0];
    yaw_Deg = 0;
    t_ini = 0;
end
% t_ini = 0;
simdata.latitude = simdata.latitude + ini_pos(2)/simdata.a;
simdata.longitude = simdata.longitude + ini_pos(1)/simdata.a...
    /cos(simdata.latitude);
simdata.altitude = simdata.altitude + ini_pos(3);
u(8,:) = u(8,:) + t_ini;

%% ------------------------------------------------------------------------
% ------------------------------------------------------------------------
%
%  Estimated Navigation Solution
%
% ------------------------------------------------------------------------
% ------------------------------------------------------------------------
cal = 5*IMU_dt; % Number of initial time steps for calibration

zupt = 1; % ZUPT switch 1 is on, 10 is off
ifALT = 1;
sol_name = 'INS';

if zupt == 1
    sol_name = [sol_name '_ZUPT'];
end
if ifALT == 10
    sol_name = [sol_name '_ALT'];
end

```

```
% converting accelerometer readouts to m/s^2 and gyroscope to rad/s
v = averageIMUReadings(u,1);
v = convertAltimeterReadings(u,v,simdata);

v = convertIMUReadings(v,g);

% calculating initial states for the EKF
input = initializeEKFState(v(2:cal,1:3),zeros(3, 1),yaw_Deg,simdata);
IMUBiasState = initializeIMUBias(zeros(3,1),mean(v(cal-400+1:cal, 4:6))');

sensor.dt = simdata.Ts;
true.t = u(8,1:end-1);
true.dt = simdata.Ts;

n = length(true.t);

est = initializeEstimationState(true,input,n);

% Set up EKF
[Q_diag,P] = setEKFProcessNoiseMTX(true.dt,simdata);
[H_ZUPT,R_ZUPT] = constructZUPTMeasurementMTX(size(P,1),simdata);
[H_ZARU,R_ZARU] = constructZARUMeasurementMTX(size(P,1),simdata);
[H_ALT,R_ALT] = constructALTMeasurementMTX(size(P,1),simdata);

ZUPTDetection = initializeDetectionState(n);
ZARUDetection = initializeDetectionState(n);

dx = zeros(size(P,1),1);
W = simdata.Window_size; % ZUPTing window size

dx_hist = nan(length(dx),n);

input_cal = input;

progress_bar_res = floor(0.01*n);

% fine alignment
for i=2:cal

    if mod(i,progress_bar_res) == 0
        waitbar(floor(i/n*100)/100,progress_bar,['Estimating solution... ',...
            num2str(round(toc,1)),' s']);
    end
    sensor = getNthIMUMeasurements(v,u,IMUBiasState,i);
    sensor = getNthALTMeasurements(sensor,v,i);
    if i < n - W + 2
        [ZUPTDetection.state(i:i+W-1),ZUPTDetection.statistics(i:i+W-1),...
```

```
163             ZUPTDetection.statistics_x(i:i+W-1),ZUPTDetection.statistics_y(i:i+W-1),...
164             ZUPTDetection.statistics_z(i:i+W-1)] = SHOE_detector(v(i:i+W-1,
                    :)',simdata);
165         [ZARUDetection.state(i:i+W-1),ZARUDetection.statistics(i:i+W-1),...
166             ZARUDetection.statistics_x(i:i+W-1),ZARUDetection.statistics_y(i:i+W-1),...
167             ZARUDetection.statistics_z(i:i+W-1)] = ZARU_detector(v(i:i+W-1,
                    :)',simdata);
168     end
169     input_cal = navSLN_ZUPT(sensor, input_cal);
170     A = setEKFPropagationMTX(sensor,input_cal,simdata);
171     P = EKFPropagationStep(A,Q_diag,P,sensor);
172
173     z = [];
174     H = [];
175     R = [];
176     if 1
177         [z,H,R] = addEKFMeasurements(z,H,R,input_cal.v_nWrtE_n,H_ZUPT,...
178             R_ZUPT*0.001); % ZUPT
179     end
180     if 1 && ZARUDetection.state(i) == 1 %&& i > cal - 400
181         [z,H,R] = addEKFMeasurements(z,H,R,sensor.w_b2i_b,H_ZARU,R_ZARU); % ZARU
182     end
183
184
185     if ~isempty(z)
186
187         [P,S,dx] = EKFUpdateStep(z,H,R,P);
188         [input_cal,IMUBiasState] =
                UpdateEKFState(input_cal,IMUBiasState,dx,simdata);
189
190         dx_hist(:,i) = dx;
191     end
192     P  = (P+P')/2;
193     est.q_b2n(:,i)    = input_cal.q_b2n;
194     est.q_e2n(:,i)    = input_cal.q_e2n;
195     est.LLA(:,i)      = input_cal.LLA;
196     est.v_nWrtE_n(:,i) = input_cal.v_nWrtE_n;
197     kf.dx(:,i)    = dx;
198     kf.gBias(:,i)    = IMUBiasState.gBias;
199     kf.aBias(:,i)    = IMUBiasState.aBias;
200     kf.diagP(:,i) = diag(P);
201 end
202
203 for i=cal+1:length(est.t)
204
205     if mod(i,progress_bar_res) == 0
206         waitbar(floor(i/n*100)/100,progress_bar,['Estimating solution... ',...
```

```
207             num2str(round(toc,1)),' s']);
208     end
209
210     sensor = getNthIMUMeasurements(v,u,IMUBiasState,i);
211     sensor = getNthALTMeasurements(sensor,v,i);
212
213     if i < n - W + 2
214         [ZUPTDetection.state(i:i+W-1),ZUPTDetection.statistics(i:i+W-1),...
215             ZUPTDetection.statistics_x(i:i+W-1),ZUPTDetection.statistics_y(i:i+W-1),...
216             ZUPTDetection.statistics_z(i:i+W-1)] = SHOE_detector(v(i:i+W-1,
                    :)',simdata);
217         [ZARUDetection.state(i:i+W-1),ZARUDetection.statistics(i:i+W-1),...
218             ZARUDetection.statistics_x(i:i+W-1),ZARUDetection.statistics_y(i:i+W-1),...
219             ZARUDetection.statistics_z(i:i+W-1)] = ZARU_detector(v(i:i+W-1,
                    :)',simdata);
220     end
221
222     input = navSLN_ZUPT(sensor, input);
223
224     A = setEKFPropagationMTX(sensor,input,simdata);
225
226     P = EKFPropagationStep(A,Q_diag,P,sensor);
227
228     z = [];
229     H = [];
230     R = [];
231     if (ZUPTDetection.state(i) == zupt)
232         zZUPT = input.v_nWrtE_n;
233         [z,H,R] = addZUPTMeasurements(z,H,R,zZUPT,H_ZUPT,R_ZUPT);
234     else
235         zZUPT = nan(3,1);
236     end
237     if 1 && ifALT == 1 && ~isnan(sensor.alt)
238         zALT = sensor.alt - input.LLA(3);
239         [z,H,R] = addEKFMeasurements(z,H,R,zALT,H_ALT,R_ALT);
240     else
241         zALT = nan;
242     end
243     if ~isempty(z)
244         %
245         [P,S,dx] = EKFUpdateStep(z,H,R,P);
246
247         [input,IMUBiasState] = UpdateEKFState(input,IMUBiasState,dx,simdata);
248
249         dx_hist(:,i) = dx;
250     end
251     P  = (P+P')/2;
```

```
252     est.q_b2n(:,i)    = input.q_b2n;
253     est.q_e2n(:,i)    = input.q_e2n;
254     est.LLA(:,i)      = input.LLA;
255     est.v_nWrtE_n(:,i) = input.v_nWrtE_n;
256     kf.dx(:,i)    = dx;
257     kf.gBias(:,i)    = IMUBiasState.gBias;
258     kf.aBias(:,i)    = IMUBiasState.aBias;
259     kf.diagP(:,i) = diag(P);
260 end
261 % ------------------------------------------------------------------------
262 est.pos = LLAToNED(est.LLA,LLA_origin,simdata);
263 est.Northing = est.pos(2,:);
264 est.Easting = est.pos(1,:);
265 est.Down = est.pos(3,:);
266 est.rpyDeg = quaternionB2NToDegree(est.q_b2n);
267 %
268 disp(['Implementation elapsed time: ' num2str(round(toc,2)) ' s'])
269
270 %% Compare with Ground Truth
271 if GT_file_exit
272     waitbar(1,progress_bar,'Calculating errors...');
273     est.pos(3,:) = -est.pos(3,:);
274
275     [est_align,est_GT_align] = alignTrajectory(est,est_GT);
276     error = computeNavigationError(est_align,est_GT_align);
277     error_info = errorComputation(error.pos,est.pos);
278
279     if 1
280         figure
281         plotEstvsRefPath(est_align,est_GT_align,"3D");
282     end
283     if 1
284         figure
285         plotEstvsRefPath(est_align,est_GT_align,"Z");
286     end
287 end
288 %% ------------------------------------------------------------------------
289 % ------------------------------------------------------------------------
290 %
291 %  Results Plotting
292 %
293 % ------------------------------------------------------------------------
294 % ------------------------------------------------------------------------
295 waitbar(1,progress_bar,'Ploting...');
296 if (1)
297     figure
298     plotAccelReadings(u,cal,length(est.t))
```

```
end
% ------------------------------------------------------------------------
if (1)
    figure
    plotGyroReadings(u,cal,length(est.t))
end
% ------------------------------------------------------------------------
if (1)
    figure
    plotAccelvsZUPT(u,ZUPTDetection,cal,length(est.t)-1)
end
% ------------------------------------------------------------------------
if (1)
    figure
    plotGyrovsZUPT(u,ZUPTDetection,cal,length(est.t)-1)
end
% ------------------------------------------------------------------------
if (1)
    figure
    plotTemperatureReadings(u,cal,length(est.t))
end
% ------------------------------------------------------------------------
if (1)
    figure
    plotAltimeterReadings(u,cal,length(est.t))
end
if (1)
    figure
    plotAltimeterHeight(v,est,cal,length(est.t))
end
% ------------------------------------------------------------------------
if (1)
    figure
    plotAngleUncertainty(est,kf,cal,length(est.t))
end
% ------------------------------------------------------------------------
if (0) % turn this on later
    figure
    plotAngleWUncertainty(est,kf,cal,length(est.t))
end
% ------------------------------------------------------------------------
if (1)
    figure
    plotAngleEstimates(est,cal,length(est.t))
end
% ------------------------------------------------------------------------
if (1)
```

```
346     figure
347     plotVelocityEstimates(est,ZUPTDetection,cal,length(est.t)-1);
348 end
349 % ------------------------------------------------------------------------
350 if (1)
351     figure
352     plotVelocityCorrection(est,kf,ZUPTDetection,cal,length(est.t))
353 end
354 % ------------------------------------------------------------------------
355 if (1)
356     figure
357     plotVelocityUncertainty(est,kf,ZUPTDetection,cal,length(est.t))
358 end
359 %% -----------------------------------------------------------------------
360 if (0)
361     figure
362     plotPositionUncertainty(est,kf,ZUPTDetection,cal,length(est.t))
363 end
364
365 %% -----------------------------------------------------------------------
366 if (1)
367     figure
368     plotHorizontalPath(est,cal,length(est.t))
369 end
370 % ------------------------------------------------------------------------
371 if(1)
372     figure
373     plotPathLLA(est,cal,length(est.t))
374 end
375 % ------------------------------------------------------------------------
376 if (1)
377     figure
378     plot3DPath(est,cal,length(est.t),'r','')
379 end
380 % ------------------------------------------------------------------------
381 if (0) % anamation
382     figure
383     animatePath(est,cal,length(est.t))
384 end
385 % ------------------------------------------------------------------------
386 if (1)
387     align_time = 48;
388     figure
389     plot3DRotatedPath(est,cal,length(est.t),align_time*IMU_dt)
390     figure
391     plot2DRotatedPath(est,cal,length(est.t),align_time*IMU_dt)
392 end
```

```
% ------------------------------------------------------------------------
if (1)
    figure
    plotPositionUncertainty(est,kf,ZUPTDetection,cal,length(est.t))
end
% ------------------------------------------------------------------------
if GT_file_exit
    figure
    plotDisplacementError(error)
    figure
    plotDisplacementXError(error)
    figure
    plotDisplacementYError(error)
    figure
    plotDisplacementZError(error)
end
%% ------------------------------------------------------------------------
if (1)
    figure
    plotAccelBias(est,kf,cal,length(est.t))
end
% ------------------------------------------------------------------------
if (1) % turn this on later
    figure
    plotGyroBias(est,kf,cal,length(est.t))
end
% ------------------------------------------------------------------------
if (1)
    figure
    plotZUPTStatistics(est,ZUPTDetection,cal,length(est.t),simdata)
end
% ------------------------------------------------------------------------
if (1)
    figure
    plotZARUStatistics(est,ZARUDetection,cal,length(est.t),simdata)
end
% display navigation results
dispNavigationResults(est,cal,length(est.t))

% save navigation results
INS_info.IMU_readouts = u(1:6,2:end);
INS_info.ZUPT_flags = ZUPTDetection.state;
INS_info.velocity = est.v_nWrtE_n;
INS_info.trajectory = [est.Northing;est.Easting;est.Down];
INS_info.heading = est.rpyDeg; % roll pitch yaw
INS_info.timestamp = est.t;
INS_info.covariance_mtx = kf.diagP;
```

```
close(progress_bar);
```

B.1.2 Parameter Settings

```
function s=settings_constructed_data
s=0;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%              IMU Selection                  %%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% selecting what IMU is chosen
% 0 = MPU9250,
% 1 = VN-200,
% 2 = ADIS16485,
% 3 = ADIS16497-3,
% 4 = SmartBug
% 5 = simulated comsumer-grade device,
% 6 = simulated industrial-grade
% device,
% 7 = simulated tactical-grade device,
% 8 = simulated
% navigation-grade device,
% 9 = VN-100,
% 10 = ADIS16497-3 with ADIS_EVAL,
% 11 = Lab-On-Shoe 2.0,
% 12 = Sugar-Cube Lab (ICM-20948),
% 13 = Open Shoe (4xMPU-9150)
% other number = simulated device,
imu_id = 12;
if imu_id == 0
    disp('----MPU9250----')
elseif imu_id == 1
    disp('----VN-200----')
elseif imu_id == 2
    disp('----ADIS16485----')
elseif imu_id == 3
    disp('----ADIS16497-3 w/ Lab-On-Shoe----')
elseif imu_id == 4
    disp('----SmartBug----')
elseif imu_id == 5
    disp('----Simulated comsumer-grade device----')
elseif imu_id == 6
    disp('----Simulated industrial-grade device----')
elseif imu_id == 7
    disp('----Simulated tactical-grade device----')
```

```
elseif imu_id == 8
    disp('----Simulated navigation-grade device----')
elseif imu_id == 9
    disp('----VN-100----')
elseif imu_id == 10
    disp('----ADIS16497-3 w/ ADIS_EVAL----')
elseif imu_id == 11
    disp('----ADIS16497-3 w/ Lab-On-Shoe 2.0----')
elseif imu_id == 12
    disp('----Sugar-Cube Lab (ICM-20948)----')
elseif imu_id == 13
    disp('----Open Shoe (4xMPU-9150)----')
else
    disp('----Simulated device----')
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%              GENERAL PARAMETERS       %%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

global simdata;

simdata.c = 299792458; % speed of light, m/s

% Total time span [s]
simdata.timespan=3600;

simdata.altitude=0; % Ali's Lab

simdata.latitude = 33.642576800000000*pi/180;

simdata.longitude = -1.178447567000000e+02*pi/180;

% Sampling period [s]
if imu_id == 0
    simdata.Ts=1/100; % for MPU-9250
elseif imu_id == 1
    simdata.Ts=1/800; % for VN-200
elseif imu_id == 2
    simdata.Ts=1/120; % for ADIS16485
elseif imu_id == 3
    simdata.Ts=1/120; % for Lab-On-Shoe
elseif imu_id == 4
    simdata.Ts=1/100; % for SmartBug
elseif imu_id == 5
    simdata.Ts=1/200; % for comsumer
elseif imu_id == 6
```

```
    simdata.Ts=1/200; % for industrial
elseif imu_id == 7
    simdata.Ts=1/200; % for tactical
elseif imu_id == 8
    simdata.Ts=1/200; % for navigation
elseif imu_id == 9
    simdata.Ts=1/400; % for VN-100
elseif imu_id == 10
    simdata.Ts=1/(4250/5)*1; % for ADIS_EVAL
elseif imu_id == 11
    simdata.Ts=1/1000; % for Lab-On-Shoe 2.0
elseif imu_id == 12
    simdata.Ts=1/370; % for Sugar-Cube
elseif imu_id == 13
    simdata.Ts=1/300; % for OpenShoe
else
    simdata.Ts = 1/800; % for Simulated device
end

disp(['|Sampling Freq = ',num2str(round(1/simdata.Ts)),'Hz |']);

% Number of iteration for DCM update
simdata.M=10;

% Time steps
simdata.N = round(simdata.timespan/simdata.Ts)+1;

% Initial orientatoin [rad] (North is zero degrees)
simdata.init_heading= 0*pi/180;

% Initial velocity (u, v, w)-axis [m/s]
simdata.init_vel=[0 0 0]';

% Earth rotation rate [rad/s]
simdata.earthrate=7.2921150e-5;

% Earth radius [m]
simdata.a=6378137;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%              ZUPTING SETUPS                 %%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Standard deviation of the acceleromter noise [m/s^2]. This is used to
% control the zero-velocity detectors trust in the accelerometer data.

if imu_id == 0
```

```
135     simdata.sigma_a=300e-6*9.81*sqrt(simdata.Ts);
136     % for MPU-9250
137 elseif imu_id == 1
138     simdata.sigma_a=0.14e-3*9.81*sqrt(simdata.Ts) * 10;
139     % for VN-200
140 elseif imu_id == 2
141     simdata.sigma_a=0.023/60*sqrt(simdata.Ts) * 1;
142     % ADIS16485
143 elseif imu_id == 3
144     simdata.sigma_a=0.04/60*sqrt(simdata.Ts) * 1;
145     % for Lab-On-Shoe
146 elseif imu_id == 4
147     simdata.sigma_a=70e-6*9.81*sqrt(simdata.Ts) * 500;
148     % SmartBug
149 elseif imu_id == 5
150     simdata.sigma_a=70e-6*9.81*sqrt(simdata.Ts) * 1;
151     % for comsumer
152 elseif imu_id == 6
153     simdata.sigma_a=7e-6*9.81*sqrt(simdata.Ts) * 1;
154     % for industrial
155 elseif imu_id == 7
156     simdata.sigma_a=0.7e-6*sqrt(simdata.Ts) * 1;
157     % for tactical
158 elseif imu_id == 8
159     simdata.sigma_a=0.07e-6*sqrt(simdata.Ts) * 1;
160     % for navigation
161 elseif imu_id == 9
162     simdata.sigma_a=0.14e-3*9.81*sqrt(simdata.Ts) * 1;
163     % for VN-100
164 elseif imu_id == 10
165     simdata.sigma_a=0.04/60*sqrt(simdata.Ts) * 50;
166     % ADIS16497 for ADIS_EVAL
167 elseif imu_id == 11
168     simdata.sigma_a=0.04/60*sqrt(simdata.Ts) * 50;
169     % ADIS16497 for Lab-On-Shoe 2.0
170 elseif imu_id == 12
171     simdata.sigma_a=0.04/60*sqrt(simdata.Ts) * 50;
172     % for ICM-20948
173 elseif imu_id == 13
174     simdata.sigma_a=0.14e-3*9.81*sqrt(simdata.Ts) * 500;
175     % Open Shoe
176 else
177     simdata.sigma_a=0.04/60*sqrt(simdata.Ts) * 1;
178     % Simulated device
179 end
180
181 % Standard deviation of the gyroscope noise [rad/s]. This is used to
```

```
% control the zero-velocity detectors trust in the gyroscope data.

if imu_id == 0
    simdata.sigma_g=0.01*pi/180*sqrt(simdata.Ts);
    %  MPU-9250
elseif imu_id == 1
    simdata.sigma_g=0.21 *pi/180/60*sqrt(simdata.Ts)*10;
    %  VN-200
elseif imu_id == 2
    simdata.sigma_g=0.3 *pi/180/60*sqrt(simdata.Ts)*1;
    % ADIS16485
elseif imu_id == 3
    simdata.sigma_g=0.18*pi/180/60*sqrt(simdata.Ts)*1;
    % ADIS16497 for Lab-On-Shoe
elseif imu_id == 4
    simdata.sigma_g=0.0028*pi/180*sqrt(simdata.Ts)*10;
    % SmartBug
elseif imu_id == 5
    simdata.sigma_g=0.5*pi/180/60*sqrt(simdata.Ts);
    % comsumer
elseif imu_id == 6
    simdata.sigma_g=0.5*pi/180/60*sqrt(simdata.Ts)*1;
    % industrial
elseif imu_id == 7
    simdata.sigma_g=0.05 *pi/180/60*sqrt(simdata.Ts)*1;
    % tactical
elseif imu_id == 8
    simdata.sigma_g=0.001*pi/180/60*sqrt(simdata.Ts)*1;
    % navigation
elseif imu_id == 9
    simdata.sigma_g=0.21 *pi/180/60*sqrt(simdata.Ts)*1;
    % VN-100
elseif imu_id == 10
    simdata.sigma_g=0.18*pi/180/60*sqrt(simdata.Ts)*10;
    % ADIS16497 for ADIS_EVAL
elseif imu_id == 11
    simdata.sigma_g=0.18*pi/180/60*sqrt(simdata.Ts)*10;
    % ADIS16497 for Lab-On-Shoe 2.0
elseif imu_id == 12
    simdata.sigma_g=0.18*pi/180/60*sqrt(simdata.Ts)*10;
    % ICM-20948
elseif imu_id == 13
    simdata.sigma_g=0.21 *pi/180/60*sqrt(simdata.Ts)*500;
    % Open Shoe
else
    simdata.sigma_g=10/60*pi/180*sqrt(simdata.Ts)*1;
    % Simulated device
```

```
end

% For ZUPT detector

simdata.gyro = simdata.sigma_g; % ARW
simdata.accel = simdata.sigma_a; % VRW

% Window size of the zero-velocity detector [samples]
% Sampling period [s]
zupt_window = 0.05;%seconds

if imu_id == 0
    simdata.Window_size=round(zupt_window/simdata.Ts);
    simdata.Window_size_for_step_detector = simdata.Window_size*5;
elseif imu_id == 1
    simdata.Window_size=round(zupt_window/simdata.Ts/1)*1;
    simdata.Window_size_for_step_detector = simdata.Window_size*5;
elseif imu_id == 2
    simdata.Window_size=round(zupt_window/simdata.Ts);
    simdata.Window_size_for_step_detector = simdata.Window_size*5;
elseif imu_id == 3
    simdata.Window_size=round(zupt_window/simdata.Ts);
    simdata.Window_size_for_step_detector = simdata.Window_size*5;
elseif imu_id == 4
    simdata.Window_size=round(zupt_window/simdata.Ts)*2;
    simdata.Window_size_for_step_detector = simdata.Window_size*5;
elseif imu_id == 5
    simdata.Window_size=round(zupt_window/simdata.Ts);
    simdata.Window_size_for_step_detector = simdata.Window_size*5;
elseif imu_id == 6
    simdata.Window_size=round(zupt_window/simdata.Ts);
    simdata.Window_size_for_step_detector = simdata.Window_size*5;
elseif imu_id == 7
    simdata.Window_size=round(zupt_window/simdata.Ts);
    simdata.Window_size_for_step_detector = simdata.Window_size*5;
elseif imu_id == 8
    simdata.Window_size=round(zupt_window/simdata.Ts);
    simdata.Window_size_for_step_detector = simdata.Window_size*5;
elseif imu_id == 9
    simdata.Window_size=round(zupt_window/simdata.Ts);
    simdata.Window_size_for_step_detector = simdata.Window_size*5;
elseif imu_id == 10
    simdata.Window_size=round(zupt_window/simdata.Ts/1);
    simdata.Window_size_for_step_detector = simdata.Window_size*5;
elseif imu_id == 11
    simdata.Window_size=round(zupt_window/simdata.Ts/1);
    simdata.Window_size_for_step_detector = simdata.Window_size*5;
```

```
elseif imu_id == 12
    simdata.Window_size=round(zupt_window/simdata.Ts/1);
    simdata.Window_size_for_step_detector = simdata.Window_size*5;
elseif imu_id == 13
    simdata.Window_size=round(zupt_window/simdata.Ts/1);
    simdata.Window_size_for_step_detector = simdata.Window_size*5;
else
    %    simdata.Window_size=5*10; % for Simulated device
    simdata.Window_size=round(zupt_window/simdata.Ts/2);
    simdata.Window_size_for_step_detector = simdata.Window_size*5;
end

simdata.Window_size_for_step_detector = simdata.Window_size*5;
simdata.Window_size_for_dynamic_covariance = simdata.Window_size/5;

% Threshold used in the zero-velocity detector. If the test statistics are
% below this value the zero-velocity hypothesis is chosen.
simdata.gamma=0.3e5;
simdata.gamma=3e5;

if imu_id == 0
    simdata.factor = exp(11); % for MPU-9250
elseif imu_id == 1
    threshold = 5;
    simdata.factor = exp(threshold); % for VN-200
    simdata.factor_step = exp(6.3);
elseif imu_id == 2
    threshold = 11;
    simdata.factor = exp(threshold); % for ADIS16485
    simdata.factor_step = exp(11);
elseif imu_id == 3
    threshold = 4.6;
    simdata.factor = exp(threshold); % for Lab-On-Shoe
    simdata.factor_step = exp(14);
elseif imu_id == 4
    threshold = 12.2;
    simdata.factor = exp(threshold); % for SmartBug
    simdata.factor_step = exp(14);
elseif imu_id == 5
    threshold = 6.2;
    simdata.factor = exp(threshold); % for comsumer
    simdata.factor_step = exp(11);
elseif imu_id == 6
    threshold = 9.1;
    simdata.factor = exp(threshold); % for industrial
    simdata.factor_step = exp(11);
elseif imu_id == 7
```

```
    threshold = 16.8;
    simdata.factor = exp(threshold); % for tactical
    simdata.factor_step = exp(11);
elseif imu_id == 8
    threshold = 8;
    simdata.factor = exp(threshold); % for navigation
    simdata.factor_step = exp(11);
elseif imu_id == 9
    threshold = 13;
    simdata.factor = exp(threshold); % for VN-100
    simdata.factor_step = exp(11);
elseif imu_id == 10
    threshold = 5.5;
    simdata.factor = exp(threshold); % for ADIS_EVAL
    simdata.factor_step = exp(12);
elseif imu_id == 11
    threshold = 2.0;
    simdata.factor = exp(threshold); % for Lab-On-Shoe 2.0
    simdata.factor_step = exp(5);
elseif imu_id == 12
    threshold = 6;
    simdata.factor = exp(threshold); % for Sugar-Cube Lab 20948
    simdata.factor_step = exp(14);
elseif imu_id == 13
    threshold = -7.4; % for stationary case
    simdata.factor = exp(threshold); % for OpenShoe
    simdata.factor_step = exp(14);
else
    threshold = 7;
    simdata.factor = exp(threshold); % for Simulated device
    simdata.factor_step = exp(10.5);
end

disp(['ZUPT thresholds = ',num2str(threshold) ])

% Thresholds when other sensors are included
simdata.factor2 = exp(6);
simdata.factorDVS = exp(14); % DVS-aided SHOE Detector

if imu_id == 0
    simdata.factor = exp(11); % for MPU-9250
elseif imu_id == 1
    simdata.factorDownwardSONAR = exp(12); % UA-SHOE Detector
    simdata.factorShoeHeight = 0.045; % USPD Detector
elseif imu_id == 2
    simdata.factorDownwardSONAR = exp(12); % UA-SHOE Detector
    simdata.factorShoeHeight = 0.045; % USPD Detector
```

```
370 elseif imu_id == 3
371     simdata.factorDownwardSONAR = exp(12); % UA-SHOE Detector
372     simdata.factorShoeHeight = 0.045; % USPD Detector
373 elseif imu_id == 4
374     simdata.factorDownwardSONAR = exp(12); % UA-SHOE Detector
375     simdata.factorShoeHeight = 0.045; % USPD Detector
376 elseif imu_id == 5
377     simdata.factorDownwardSONAR = exp(12); % UA-SHOE Detector
378     simdata.factorShoeHeight = 0.045; % USPD Detector
379 elseif imu_id == 6
380     simdata.factorDownwardSONAR = exp(12); % UA-SHOE Detector
381     simdata.factorShoeHeight = 0.045; % USPD Detector
382 elseif imu_id == 7
383     simdata.factorDownwardSONAR = exp(12); % UA-SHOE Detector
384     simdata.factorShoeHeight = 0.045; % USPD Detector
385 elseif imu_id == 8
386     simdata.factorDownwardSONAR = exp(12); % UA-SHOE Detector
387     simdata.factorShoeHeight = 0.045; % USPD Detector
388 elseif imu_id == 9
389     simdata.factorDownwardSONAR = exp(12); % UA-SHOE Detector
390     simdata.factorShoeHeight = 0.045; % USPD Detector
391 elseif imu_id == 10
392     simdata.factorDownwardSONAR = exp(12); % UA-SHOE Detector
393     simdata.factorShoeHeight = 0.045; % USPD Detector
394 elseif imu_id == 11
395     simdata.factorDownwardSONAR = exp(1.2); % UA-SHOE Detector
396     simdata.factorShoeHeight = 0.09; % USPD Detector
397 elseif imu_id == 12
398     simdata.factorDownwardSONAR = exp(12); % UA-SHOE Detector
399     simdata.factorShoeHeight = 0.045; % USPD Detector
400 elseif imu_id == 13
401     simdata.factorDownwardSONAR = exp(12); % UA-SHOE Detector
402     simdata.factorShoeHeight = 0.045; % USPD Detector
403 else
404     simdata.factorDownwardSONAR = exp(12); % UA-SHOE Detector
405     simdata.factorShoeHeight = 0.045; % USPD Detector
406 end
407
408 % Diagonal elements of the initial state covariance matrix (P).
409 % MPU9250: Turn on bias: 5 deg/s 60 mg for x&y 80 mg for z 60 mg = 0.59 m/s^2
410 simdata.sigma_initial_acc_bias=1e-4*ones(1,3)*10; % Accelerometer biases [m/s^2]
411 simdata.sigma_initial_gyro_bias=5*pi/180/3600*ones(1,3); % Gyroscope biases
        [rad/s]
412 simdata.sigma_initial_pos=1e-3*ones(1,3)*0.1; % Position [m]
413 simdata.sigma_initial_vel=1e-3*ones(1,3)*1; % Velocity [m/s]
414 simdata.sigma_initial_att=(2* pi/180*ones(1,3))*0.1; % 2 deg Attitude [rad]
415 simdata.sigma_initial_acc_scale=0.0001*ones(1,3); % Accelerometer scale factors
```

```
416 simdata.sigma_initial_gyro_scale=0.00001*ones(1,3); % Gyroscope scale factors
417
418 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
419 %      For initial state matrix Q      %
420 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
421
422 % Process noise for modeling the drift in accelerometer biases (x,y,z
423 % platform coordinate axis) [m/s^2]. (In-Run Bias Stability)
424
425 if imu_id == 0
426     simdata.acc_bias_driving_noise=8e-3*9.81*sqrt(simdata.Ts);
427     % 8mg no data for MPU-9250
428 elseif imu_id == 1
429     simdata.acc_bias_driving_noise=40e-6*9.81*sqrt(simdata.Ts)*1;
430     % 0.04mg VN-200
431 elseif imu_id == 2
432     simdata.acc_bias_driving_noise=32e-6*9.81*sqrt(simdata.Ts);
433     % 32ug for ADIS16485
434 elseif imu_id == 3
435     simdata.acc_bias_driving_noise=13e-6*9.81*sqrt(simdata.Ts)*1;
436     % 13ug for Lab-On-Shoe
437 elseif imu_id == 4
438     simdata.acc_bias_driving_noise=700e-6*9.81*sqrt(simdata.Ts);
439     % 0.70mg for SmartBug
440 elseif imu_id == 5
441     simdata.acc_bias_driving_noise=50e-3*9.81*sqrt(simdata.Ts);
442     % >50mg for comsumer-grade device
443 elseif imu_id == 6
444     simdata.acc_bias_driving_noise=1e-3*9.81*sqrt(simdata.Ts);
445     % 1~50mg for industrial-grade device
446 elseif imu_id == 7
447     simdata.acc_bias_driving_noise=0.05e-3*9.81*sqrt(simdata.Ts);
448     % 0.05~1mg for tactical-grade device
449 elseif imu_id == 8
450     simdata.acc_bias_driving_noise=0.001e-3*9.81*sqrt(simdata.Ts);
451     % <0.05mg for navigation-grade device
452 elseif imu_id == 9
453     simdata.acc_bias_driving_noise=40e-6*9.81*sqrt(simdata.Ts)*100;
454     % 0.04mg VN-100
455 elseif imu_id == 10
456     simdata.acc_bias_driving_noise=13e-6*9.81*sqrt(simdata.Ts)*0.1;
457     % 13ug for ADIS_EVAL
458 elseif imu_id == 11
459     simdata.acc_bias_driving_noise=13e-6*9.81*sqrt(simdata.Ts)*1;
460     % 13ug for Lab-On-Shoe
461 elseif imu_id == 12
462     simdata.acc_bias_driving_noise=13e-6*9.81*sqrt(simdata.Ts)*1;
```

```
    % Sugar-Cube Lab ICM-20948
elseif imu_id == 13
    simdata.acc_bias_driving_noise=40e-6*9.81*sqrt(simdata.Ts)*1;
    % Open Shoe
else
    simdata.acc_bias_driving_noise=4.9e-6*9.81*sqrt(simdata.Ts)*0.1;
    % 4.9ug for Simulated device
end

% Process noise for modeling the drift in gyroscope biases (x,y,z platform
% coordinate axis) [rad/s]. (In-Run Bias Stability)
if imu_id == 0
    simdata.gyro_bias_driving_noise= 0.1 *pi/180*sqrt(simdata.Ts);
    % 0.1 degree/s for MPU-9250
elseif imu_id == 1
    simdata.gyro_bias_driving_noise= 10/3600 *pi/180*sqrt(simdata.Ts)*1;
    % 10 degree/hr for VN-200
elseif imu_id == 2
    simdata.gyro_bias_driving_noise= 6.25/3600 *pi/180*sqrt(simdata.Ts);
    % 6.25 degree/hr for ADIS16485
elseif imu_id == 3
    simdata.gyro_bias_driving_noise= 3.3/3600 *pi/180*sqrt(simdata.Ts)*0.35;
    % 3.3 degree/hr for ADIS16497-3 for Right Lab-On-Shoe
elseif imu_id == 4
    simdata.gyro_bias_driving_noise= 0.028 *pi/180*sqrt(simdata.Ts)*1;
    % 0.028 degree/s for SmartBug
elseif imu_id == 5
    simdata.gyro_bias_driving_noise= 100/3600 *pi/180*sqrt(simdata.Ts)*10;
    % > 100 degree/h for comsumer-grade device
elseif imu_id == 6
    simdata.gyro_bias_driving_noise= 10/3600 *pi/180*sqrt(simdata.Ts)*1;
    % 10~100 degree/h for industrial-grade device
elseif imu_id == 7
    simdata.gyro_bias_driving_noise= 0.1/3600 *pi/180*sqrt(simdata.Ts);
    % 0.01~10 degree/h for tactical-grade device
elseif imu_id == 8
    simdata.gyro_bias_driving_noise= 0.001/3600 *pi/180*sqrt(simdata.Ts);
    % <0.01 degree/h for navigation-grade device
elseif imu_id == 9
    simdata.gyro_bias_driving_noise= 10/3600 *pi/180*sqrt(simdata.Ts)*0.1;
    % 10 degree/hr for VN-100
elseif imu_id == 10
    simdata.gyro_bias_driving_noise= 3.3/3600 *pi/180*sqrt(simdata.Ts)*0.1;
    % 3.3 degree/hr for ADIS16497-3 for Left Lab-On-Shoe
elseif imu_id == 11
    simdata.gyro_bias_driving_noise= 3.3/3600 *pi/180*sqrt(simdata.Ts)*0.35;
    % 3.3 degree/hr for ADIS16497-3 for Right Lab-On-Shoe
```

```
elseif imu_id == 12
    simdata.gyro_bias_driving_noise= 3.3/3600 *pi/180*sqrt(simdata.Ts)*0.35;
    % Sugar-Cube Lab ICM-20948
elseif imu_id == 13
    simdata.gyro_bias_driving_noise= 10/3600 *pi/180*sqrt(simdata.Ts)*1;
    % Open Shoe
else
    simdata.gyro_bias_driving_noise= 8e-4 *pi/180*sqrt(simdata.Ts)*0.1;
    % for Simulated device
end

% Process noise for modeling the scale factor errors in accelerometers
simdata.acc_SF_driving_noise = 1e-10;

% Process noise for modeling the scale factor errors in gyroscopes
simdata.gyro_SF_driving_noise = 1e-10;

% Process noise for modeling the non-orthogonality in accelerometers
simdata.acc_ortho_driving_noise = 1e-8;

% Process noise for modeling the rotation misalignment in gyroscopes
simdata.gyro_rot_driving_noise = 1e-8;

% Process noise for modeling the non-orthogonality in gyroscopes
simdata.gyro_ortho_driving_noise = 1e-8;

% Pseudo zero-velocity update measurement noise covariance (R). The
% covariance matrix is assumed diagonal.
% Errors in the velocity measurement

if imu_id == 0 % for MPU-9250
    simdata.sigma_vel=[1 1 1]*0.001;  %[m/s]
elseif imu_id == 1 % VN-200
    simdata.sigma_vel=[1 1 1]*0.05;  %[m/s]
elseif imu_id == 2 % for ADIS16485
    simdata.sigma_vel=[1 1 1]*0.001;  %[m/s]
elseif imu_id == 3 % for Lab-On-Shoe
    simdata.sigma_vel=[1 1 1]*0.02;  %[m/s]
elseif imu_id == 4 % for SmartBug
    simdata.sigma_vel=[1 1 1]*0.02;  %[m/s]
elseif imu_id == 5 % for Comsumer-grade
    simdata.sigma_vel=[1 1 1]*0.003;  %[m/s]
elseif imu_id == 6 % for industrial-grade
    simdata.sigma_vel=[1 1 1]*0.02;  %[m/s]
elseif imu_id == 7 % for tactical-grade
    simdata.sigma_vel=[1 1 1]*0.02;  %[m/s]
elseif imu_id == 8 % for navigation-grade
```

```
    simdata.sigma_vel=[1 1 1]*0.001;  %[m/s]
elseif imu_id == 9 % VN-100
    simdata.sigma_vel=[1 1 1]*0.005;  %[m/s]
elseif imu_id == 10 % for ADIS_EVAL
    simdata.sigma_vel=[1 1 1]*0.02;  %[m/s]
elseif imu_id == 11 % for Lab-On-Shoe
    simdata.sigma_vel=[1 1 1]*0.02;  %[m/s]
elseif imu_id == 12 % for Sugar-Cube Lab
    simdata.sigma_vel=[1 1 1]*0.02;  %[m/s]
elseif imu_id == 13 % VN-200
    simdata.sigma_vel=[1 1 1]*0.001;  %[m/s]

else
    simdata.sigma_vel=[1 1 1]*0.02;  %[m/s]
end

% Errors in the displacement measurement in ranging sensor
simdata.sigma_dis=1; %[m] for UWB

% Errors in the magnetic field measurement in magnetometer
simdata.sigma_mag=10; %[nT]

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%            Altimeter SETUPS          %%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if imu_id == 0
    % How many IMU data per Altimeter data
    simdata.ALT_rate = 20;
    % altimeter resolution
    simdata.alt_resolution = 0.1; % meters
elseif imu_id == 1
    % How many IMU data per Altimeter data
    simdata.ALT_rate = 20;
    % altimeter resolution
    simdata.alt_resolution = 0.1; % meters
elseif imu_id == 2
    % How many IMU data per Altimeter data
    simdata.ALT_rate = 20;
    % altimeter resolution
    simdata.alt_resolution = 0.1; % meters
elseif imu_id == 3
    % How many IMU data per Altimeter data
    simdata.ALT_rate = 20;
    % altimeter resolution
    simdata.alt_resolution = 0.1; % meters
elseif imu_id == 4
    % How many IMU data per Altimeter data
```

```
604     simdata.ALT_rate = 20;
605     % altimeter resolution
606     simdata.alt_resolution = 0.1; % meters
607 elseif imu_id == 5
608     % How many IMU data per Altimeter data
609     simdata.ALT_rate = 20;
610     % altimeter resolution
611     simdata.alt_resolution = 0.1; % meters
612 elseif imu_id == 6
613     % How many IMU data per Altimeter data
614     simdata.ALT_rate = 20;
615     % altimeter resolution
616     simdata.alt_resolution = 0.1; % meters
617 elseif imu_id == 7
618     % How many IMU data per Altimeter data
619     simdata.ALT_rate = 20;
620     % altimeter resolution
621     simdata.alt_resolution = 0.1; % meters
622 elseif imu_id == 8
623     % How many IMU data per Altimeter data
624     simdata.ALT_rate = 20;
625     % altimeter resolution
626     simdata.alt_resolution = 0.1; % meters
627 elseif imu_id == 9
628     % How many IMU data per Altimeter data
629     simdata.ALT_rate = 20;
630     % altimeter resolution
631     simdata.alt_resolution = 0.1; % meters
632 elseif imu_id == 10
633     % How many IMU data per Altimeter data
634     simdata.ALT_rate = 20;
635     % altimeter resolution
636     simdata.alt_resolution = 0.1; % meters
637 elseif imu_id == 11
638     % How many IMU data per Altimeter data
639     simdata.ALT_rate = 20;
640     % altimeter resolution
641     simdata.alt_resolution = 100; % meters
642 elseif imu_id == 12
643     % How many IMU data per Altimeter data
644     simdata.ALT_rate = 20;
645     % altimeter resolution
646     simdata.alt_resolution = 0.1; % meters
647 elseif imu_id == 13
648     % How many IMU data per Altimeter data
649     simdata.ALT_rate = 20;
650     % altimeter resolution
```

```
651     simdata.alt_resolution = 0.1; % meters
652 else
653     % How many IMU data per Altimeter data
654     simdata.ALT_rate = 20;
655     % altimeter resolution
656     simdata.alt_resolution = 0.1; % meters
657 end
658 
659 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
660 %%        relative position SETUPS     %%
661 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
662 simdata.cam_dis_std = 1.5;
663 simdata.cam_dis_vx = 0.001;
664 simdata.cam_dis_vy = 0.001;
665 simdata.cam_dis_vz = 0.001;
666 
667 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
668 %%     relative orientation SETUPS     %%
669 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
670 simdata.cam_roll_std = 0.01;
671 simdata.cam_pitch_std = 0.01;
672 simdata.cam_yaw_std = 0.01;
673 
674 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
675 %%           Adaptive ZUPT setup        %%
676 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
677 simdata.alpha = -exp(9)^1;
678 simdata.theta = 800.5;
679 simdata.beta = 0;
680 
681 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
682 %%              UWB setup               %%
683 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
684 simdata.Window_size_nlos=50; % for MPU-9250
685 threshold_nlos = 0.000000001;
686 simdata.factor_nlos = exp(threshold_nlos); % for Lab-On-Shoe
687 simdata.UWB_std =1.5;
688 
689 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
690 %%              Dynamic Covariance                %%
691 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
692 
693 % for Sugar-Cube
694 simdata.dynamic_variance_hyper_para_alpha = [0 0 0]+(exp(-4.7));
695 simdata.dynamic_variance_hyper_para_gamma = 1.8;
696 simdata.dynamic_variance_hyper_para_beta = [1 1 1]*0.02;
697 simdata.dynamic_variance_hyper_para_psi = [-1,-1,-1]*6;
```

```
698
699 end
```

B.2 Pedestrian Navigation Simulation

B.2.1 The Main Script

```
1  tic
2  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
3  % Start of the code
4  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
5  clc;
6  clear;
7  close all;
8
9  currDir = pwd;
10 datasetDir = [currDir, '\dataset','\combineSet'];
11
12 addpath(datasetDir);
13 addpath([currDir, '\lib\utility\'])
14 addpath([currDir, '\lib\INS\'])
15 addpath([currDir, '\lib\plot\'])
16 addpath([currDir, '\lib\walker\'])
17 addpath([currDir, '\lib\runner\'])
18 addpath([currDir, '\lib\FFT_analysis\'])
19 addpath([currDir, '\lib\IMU_noise_model\'])
20 addpath([currDir, '\lib\navigation\'])
21 addpath([currDir, '\lib\gait_analysis\'])
22 addpath([currDir, '\lib\stance_phase_detection\'])
23 addpath([currDir, '\lib\vicon\'])
24
25 %% Model Parameters:
26 model = 0; % if 0 walker, otherwise, runner
27 algorithm = 2; % 0: INS, 1:ZUPT, 2:EKF-ZUPT
28 numSteps = 28;
29
30 %% Noise Parameters:
31 sensorModel = 'VectorNav';  % Options: ICM_20948, VectorNav
32 device = 2;     % Add Noise to the IMU model: if 1 add Noise: if 2, real
33 %% Experiment parameter
34 expSensorName = 'VN';
35
```

```
%% Plot Parameters:
% Speed Animation: 500 samples jump in the for loop for animation
speed = 100;
% Animation of the walker: 0 = no plot; 1 = plot;
ifPlotWalker = 1;
ifPlotNavigation = 0;
ifPlotSave = 0;
% Local Frame Animation: 0 = no local frame plot; 1 = local frame plot;
ifLocalFrame = 0;
tcpUnity = 0;    % Enable/Disable TCP communication with unity:

%% Navigation Parameters:
d2r = pi/180;
r2d = 180/pi;
fs = 800;      % Sampling rate of simulated IMU measurements
lattitude_ini = 0.587173734015058; % Degree % starting at Irivne
longitude_ini = -2.056779010626648; % Degree
earth_radius = 6378137; % Earth radius
earthrate=7.2921150e-5; % Earth rate, rad/s
g = 9.8198;      % Gravity
e2  = 6.694380004260835e-3; % EarthEccentricitySq

%% operation loop
resultTable = zeros(1,16);
% simTrialVector = 0:(2^16-1);
% simTrialVector = [128,64,32,224,240,248,232,228,252,254,253,255];
simTrialVector = 0b11111111;
% expTrialVector = 1:20;
expTrialVector = 1;
repeatNum = 20;
for kk=1:length(simTrialVector)
    noiseSettingInd = decimalToBinaryVector(simTrialVector(kk),8);
    noiseSettings.whiteNoise = noiseSettingInd(1);
    noiseSettings.flickerNoise = noiseSettingInd(2);
    noiseSettings.randomWalk = noiseSettingInd(3);
    noiseSettings.misalignment = noiseSettingInd(4);
    noiseSettings.scaleFactor = noiseSettingInd(5);
    noiseSettings.iniBias = noiseSettingInd(6);
    noiseSettings.FSR = noiseSettingInd(7);
    noiseSettings.bandWidth = noiseSettingInd(8);
    resultsubTable = zeros(repeatNum,9);
    close all;
    figure(100);
    figure(101);
    figure(102);
    for jj = 1:repeatNum
        clc
```

```
        if device ==2
            dataname = ['exp' num2str(expTrialVector(jj)) '00'];
            disp(dataname);
        else
            disp(['Trial #' num2str(jj) ', Configuraion #' ...
                num2str(simTrialVector(kk))]);
        end
        %% Pre-processing trajectory obtained from camera
        if model == 0
            fs_GT = 2286; % Sampling rate of the ground truth dataset
            bodyPose = generateFootPoseRigidWalker(numSteps,speed, ...
                ifLocalFrame,ifPlotWalker,tcpUnity);

            leftleg_legnth = vecnorm(([bodyPose(1).x,bodyPose(1).y, ...
                bodyPose(1).z]-[bodyPose(3).x,bodyPose(1).y, ...
                bodyPose(3).z])');
            rightleg_legnth = vecnorm(([bodyPose(2).x,bodyPose(2).y, ...
                bodyPose(2).z]-[bodyPose(3).x,bodyPose(2).y, ...
                bodyPose(3).z])');
        else
            fs_GT = 2226; % Sampling rate of the ground truth dataset
            bodyPose = generateFootPoseSpringyWalker(numSteps,speed, ...
                ifLocalFrame,ifPlotWalker,tcpUnity);

            leftFoot2Knee_length = vecnorm(([bodyPose(1).x,bodyPose(1).y, ...
                bodyPose(1).z]-[bodyPose(3).x,bodyPose(3).y, ...
                bodyPose(3).z])');
            rightFoot2Knee_length = vecnorm(([bodyPose(2).x,bodyPose(2).y, ...
                bodyPose(2).z]-[bodyPose(4).x,bodyPose(4).y, ...
                bodyPose(4).z])');
            leftKnee2Hip_length = vecnorm(([bodyPose(3).x,bodyPose(3).y, ...
                bodyPose(3).z]-[bodyPose(5).x,bodyPose(3).y, ...
                bodyPose(5).z])');
            rightKnee2Hip_length = vecnorm(([bodyPose(3).x,bodyPose(3).y, ...
                bodyPose(3).z]-[bodyPose(5).x,bodyPose(4).y, ...
                bodyPose(5).z])');
            leftleg_legnth = leftFoot2Knee_length + leftKnee2Hip_length;
            rightleg_legnth = rightFoot2Knee_length + rightKnee2Hip_length;
        end

        % synchronized sample rate of simulation with IMU measurements
        groundTrue = groundTruthGeneration(bodyPose(2),fs_GT,fs);

        %% reconstruct IMU measurements

        % calculate IMU measurement from ground truth position
        [sensor_raw,groundTrueStates_raw] = simulateIMUFromPose(groundTrue, ...
```

```
130             fs,longitude_ini,lattitude_ini);
131         if device == 0
132             disp('No noise applied on the synthetic IMU')
133             sensor = sensor_raw;
134             groundTrueStates = groundTrueStates_raw;
135         elseif device == 1
136             disp(['Noise sensor model: ' sensorModel])
137             noise_sensor = addIMUNoise(sensor_raw,sensorModel,noiseSettings);
138             sensor = noise_sensor;
139             groundTrueStates = groundTrueStates_raw;
140         elseif device == 2
141             % Load experimental data
142             [real_sensor,sim_sensor,groundTrueStatesSensor] = loadRealIMUData( ...
143                 dataname,expSensorName,lattitude_ini,longitude_ini);
144             disp('Real sensor')
145             sensor = real_sensor;
146             groundTrueStates = groundTrueStatesSensor;
147         end
148         if ifPlotNavigation==1
149             figure
150             plotSimulatedIMU(sensor)
151         end
152         %% implement navigation equations based on simulated IMU measurements
153
154         if algorithm == 0
155             disp('Standalone INS')
156             est = strapDownINS(sensor,groundTrueStates);
157         elseif algorithm == 1
158             disp('ZUPT-aided INS')
159             est = INSVelocityReset(sensor,groundTrueStates);
160             if ifPlotNavigation==1
161                 plotZUPTAndIMU(sensor,est);
162             end
163         else
164             disp('ZUPT-aided INS with ESKF')
165             est = ZUPTwEKF(sensor,groundTrueStates,ifLocalFrame);
166             if ifPlotNavigation==1
167                 plotZUPTAndIMU(sensor,est);
168             end
169         end
170
171         angleToNorth = calculateAngleToNorth(est,groundTrueStates,30);
172         est_North = rotateNavigation(est,angleToNorth);
173         est_Plot = est_North; % chose which cooridinate to plot
174         %% Compute errors
175         error = computeNavErrorWRef(groundTrueStates,est_Plot);
176         if ifPlotNavigation==1
```

```
177             figure
178             plotNavError(error)
179             disp(['Distance to Origin: ', num2str(norm(error.pos(:,end))) ...
180                 , ' [m]']);
181 
182             figure;
183             plot(est_Plot.timestamps(1:length(est_Plot.dotv_n)), ...
184                 est_Plot.dotv_n');
185             legend('x','y','z')
186             title('Acceleration From Motion')
187             xlabel('time, s')
188             ylabel('m/s^2')
189 
190             figure;
191             plot(est_Plot.timestamps(1:length(est_Plot.w_n2b_b)), ...
192                 est_Plot.w_n2b_b'*180/pi);
193             legend('x','y','z')
194             title('Angular Rate From Motion')
195             xlabel('time, s')
196             ylabel('dps')
197         end
198         end_ind = length(error.pos)-round(0.05/sensor.dt(1));
199         resultsubTable(jj,:) = [max(max(error.pos(:,1:end_ind))),...
200             sum(vecnorm(error.pos(:,1:end_ind)))/end_ind...
201             sum(vecnorm(error.pos(1:2,1:end_ind)))/end_ind, ...
202             sum(error.pos(3,1:end_ind))/end_ind, norm(error.pos(:,end_ind)),...
203             norm(error.pos(1:2,end_ind)), est_Plot.pos(:,end_ind)'];
204         %% Plot Grouth True results and INS results
205         % plot velocity
206         GT_plot = groundTrueStates_raw;
207         if ifPlotNavigation==1
208             figure
209             plotComparedVelocity(GT_plot,est_Plot)
210             % plot orientation
211             figure
212             plotComparedOrientation(GT_plot,est_Plot)
213             % plot position
214             figure;
215             plotComparedPath(groundTrueStates,est);
216             figure
217             plotComparedPostion(GT_plot,est_Plot)
218             % ylim([-1 1])
219             % zlim([-1 2])
220         end
221 
222         figure(100);
223         plotPostion(est_Plot);hold on
```

```
        drawnow
        figure(101);
        plotXYPostion(est_Plot);hold on
        drawnow
        figure(102);
        plotXZPostion(est_Plot);hold on
        drawnow
    end
    if ifPlotSave == 1
        if device == 2
            saveas(figure(100),[pwd '\result plots\sim_figure\exp_' ...
                expSensorName,'.fig']);
            saveas(figure(101),[pwd '\result plots\sim_figure\exp_' ...
                expSensorName,'XY.fig']);
            saveas(figure(102),[pwd '\result plots\sim_figure\exp_' ...
                expSensorName,'XZ.fig']);
            save([pwd '\result plots\sim_result\exp_' expSensorName,'.mat'], ...
                'resultsubTable');
        else
            saveas(figure(100),[pwd '\result plots\sim_figure\simConfiguration' ...
                num2str(simTrialVector(kk)),'.fig']);
            saveas(figure(101),[pwd '\result plots\sim_figure\simConfiguration' ...
                num2str(simTrialVector(kk)),'XY.fig']);
            saveas(figure(102),[pwd '\result plots\sim_figure\simConfiguration' ...
                num2str(simTrialVector(kk)),'XZ.fig']);
            save([pwd '\result plots\sim_result\simConfiguration' ...
                num2str(simTrialVector(kk)),'.mat'],'resultsubTable');
        end
    end
    resultTable(kk,:) = [noiseSettingInd,max(resultsubTable(:,1)),...
        sum(resultsubTable(:,2))/repeatNum,...
        sum(resultsubTable(:,3))/repeatNum,sum(resultsubTable(:,4))/repeatNum,...
        sum(resultsubTable(:,5))/repeatNum,...
        sum(resultsubTable(:,6))/repeatNum, sum(resultsubTable(:,9))/repeatNum,...
        median(resultsubTable(:,6))];
end
if ifPlotSave == 1
    if device == 2
        save([pwd '\result plots\sim_result\expTotal.mat'],'resultTable');
    else
        save([pwd '\result plots\sim_result\simTotal.mat'],'resultTable');
    end
end
%% end of code
toc
```

B.3 Custom Libraries

B.3.1 Inertial Navigation Library

DCM orthogonalization

```
function C_out = DCM_othogonalization(C_in)

r_x = C_in(1,:);
r_y = C_in(2,:);
error = sum(r_x.*r_y);
t_0 = r_x-1/2*error*r_y;
t_1 = r_y-1/2*error*r_x;
t_2 = (skew(t_0)*t_1')';

C_out = [t_0;t_1;t_2];

end
```

DCM to Euler Angle conversion

```
function eul_vect = dcm2eulrCSJ(DCMbn)
%EULR2DCM      Euler angle vector to direction cosine
%              matrix conversion.
%
%  eul_vect = dcm2eulrCSJ(DCMbn)
%
%   INPUTS
%       DCMbn = 3x3 direction cosine matrix providing the
%             transformation from the body frame
%             to the navigation frame
%   OUTPUTS
%       eul_vect(1) = roll angle in radians
%
%       eul_vect(2) = pitch angle in radians
%
%       eul_vect(3) = yaw angle in radians
%
%
phi = atan2(DCMbn(3,2),DCMbn(3,3));
theta = atan2(-DCMbn(3,1),sqrt(DCMbn(3,2)^2+DCMbn(3,3)^2));
```

```
psi = atan2(DCMbn(2,1),DCMbn(1,1));
eul_vect = [phi theta psi];
```

DCM to Quaternion conversion

```
function qua_vec = dcm2quaCSJ(DCMbn)
%DCM2QUA      Direction cosine matrix to quaternion conversion.
%
%   INPUTS
%       DCMbn = 3x3 direction cosine matrix providing the
%             transformation from the body frame
%             to the navigation frame
%
%   OUTPUTS
%       qua_vec = 4 element quaternion vector
%               = [a b c d]
%       where: a = cos(MU/2)
%              b = (MUx/MU)*sin(MU/2)
%              c = (MUy/MU)*sin(MU/2)
%              d = (MUz/MU)*sin(MU/2)
%       where: MUx, MUy, MUz are the components of the angle vector
%              MU is the magnitude of the angle vector
%
%   NOTE
%       The algorithm assumes small angular displacements.
%

%   REFERENCE: Titterton, D. and J. Weston, STRAPDOWN
%              INERTIAL NAVIGATION TECHNOLOGY, Peter
%              Peregrinus Ltd. on behalf of the Institution
%              of Electrical Engineers, London, 1997.

if nargin<1,error('insufficient number of input arguments'),end
%
a = 0.5*sqrt(1+DCMbn(1,1)+DCMbn(2,2)+DCMbn(3,3));
tmp = inv(4*a);
b = tmp*(DCMbn(3,2)-DCMbn(2,3));
c = tmp*(DCMbn(1,3)-DCMbn(3,1));
d = tmp*(DCMbn(2,1)-DCMbn(1,2));

qua_vec = [a b c d];
```

Euler Angle to DCM Angle conversion

```
function DCMnb = eulr2dcmCSJ(eul_vect)
%EULR2DCM     Euler angle vector to direction cosine
%             matrix conversion.
%
%  DCMnb = eulr2dcm(eul_vect)
%
%   INPUTS
%       eul_vect(1) = roll angle in radians
%
%       eul_vect(2) = pitch angle in radians
%
%       eul_vect(3) = yaw angle in radians
%
%   OUTPUTS
%       DCMnb = 3x3 direction cosine matrix providing the
%            transformation from the navigation frame
%            to the body frame
%

%   REFERENCE: Titterton, D. and J. Weston, STRAPDOWN
%              INERTIAL NAVIGATION TECHNOLOGY, Peter
%              Peregrinus Ltd. on behalf of the Institution
%              of Electrical Engineers, London, 1997.
%
  if nargin<1,error('insufficient number of input arguments'),end

  phi = eul_vect(1); theta = eul_vect(2); psi = eul_vect(3);

  cpsi = cos(psi); spsi = sin(psi);
  cthe = cos(theta); sthe = sin(theta);
  cphi = cos(phi); sphi = sin(phi);

  C1 = [cpsi spsi 0; ...
       -spsi cpsi 0; ...
        0    0   1];
  C2 = [cthe   0 -sthe; ...
       0   1   0; ...
       sthe    0   cthe];
  C3 = [1  0    0; ...
       0 cphi sphi; ...
       0    -sphi   cphi];

  DCMnb = C3*C2*C1;
```

Euler Angle to Quaternion conversion

```
function qua_vec = eulr2quaCSJ(eul_vect)
%EULR2QUA      Euler angle vector to quaternion conversion.
%
%  qua_vec = eulr2quaAMS(eul_vect)
%
%   INPUTS
%       eul_vect(1) = roll angle in radians
%
%       eul_vect(2) = pitch angle in radians
%
%       eul_vect(3) = yaw angle in radians
%
%   OUTPUTS
%       qua_vec = 4 element quaternion vector
%               = [a b c d]
%       where: a = cos(MU/2)
%              b = (MUx/MU)*sin(MU/2)
%              c =
%              d =
%       where: MUx, MUy, MUz are the components of the angle vector
%              MU is the magnitude of the angle vector
%

%   October 2009: Corrected sign error in computation of 'd'
%
%   REFERENCE: Titterton, D. and J. Weston, STRAPDOWN
%              INERTIAL NAVIGATION TECHNOLOGY, Peter
%              Peregrinus Ltd. on behalf of the Institution
%              of Electrical Engineers, London, 1997.

  if nargin<1,error('insufficient number of input arguments'),end

  phi = eul_vect(1); theta = eul_vect(2); psi = eul_vect(3);

  cpsi2 = cos(psi/2); spsi2 = sin(psi/2);
  cthe2 = cos(theta/2); sthe2 = sin(theta/2);
  cphi2 = cos(phi/2); sphi2 = sin(phi/2);

  a = cphi2*cthe2*cpsi2 + sphi2*sthe2*spsi2;
  b = sphi2*cthe2*cpsi2 - cphi2*sthe2*spsi2;
  c = cphi2*sthe2*cpsi2 + sphi2*cthe2*spsi2;
  d = cphi2*cthe2*spsi2 + sphi2*sthe2*cpsi2;

  qua_vec = [a b c d];
```

Quaternion to DCM conversion

```
function DCMbn = qua2dcmCSJ(qua_vec)
%   DCMbn = qua2dcmAMS(qua_vec)
%
%   INPUTS
%       qua_vec = 4 element quaternion vector
%               = [a b c d]
%       where: a = cos(MU/2)
%              b = (MUx/MU)*sin(MU/2)
%              c = (MUy/MU)*sin(MU/2)
%              d = (MUz/MU)*sin(MU/2)
%       where: MUx, MUy, MUz are the components of the angle vector
%              MU is the magnitude of the angle vector
%
%   OUTPUTS
%       DCMbn = 3x3 direction cosine matrix providing the
%             transformation from the body frame
%             to the navigation frame

a = qua_vec(1);b = qua_vec(2);c = qua_vec(3);d = qua_vec(4);

DCMbn = [a^2+b^2-c^2-d^2, 2*(b*c-a*d), 2*(b*d+a*c);
    2*(b*c+a*d), a^2-b^2+c^2-d^2, 2*(c*d-a*b);
    2*(b*d-a*c), 2*(c*d+a*b), a^2-b^2-c^2+d^2];
```

Gimbal Rate to Body Rate conversion

```
function w_b2n_n = gimbalRate2bodyRate(euler_vect_kp1,euler_vect_k,dt)

roll_kp1 = euler_vect_kp1(1);
pitch_kp1 = euler_vect_kp1(2);
yaw_kp1 = euler_vect_kp1(3);

roll_k = euler_vect_k(1);
pitch_k = euler_vect_k(2);
yaw_k = euler_vect_k(3);

C3 = [cos(roll_k) 0 -sin(roll_k);0 1 0;sin(roll_k) 0 cos(roll_k)];
C2 = [1 0 0;0 cos(pitch_k) sin(pitch_k);0 -sin(pitch_k) cos(pitch_k)];

dot_roll = (roll_kp1-roll_k)/dt;
dot_pitch = (pitch_kp1-pitch_k)/dt;
dot_yaw = (yaw_kp1-yaw_k)/dt;
```

```
17
18 M = [0 sin(roll_k) cos(roll_k);0 cos(roll_k)*cos(pitch_k)
       -sin(roll_k)*cos(pitch_k);...
19     cos(pitch_k) sin(roll_k)*sin(pitch_k) cos(roll_k)*sin(pitch_k)];
20 w_b2n_n = cos(pitch_k)*(M)^-1*[dot_roll;dot_pitch;dot_yaw];
21
22 end
```

Compute Gravity Model

```
1  function [ g ] = gravityModel(Lat, P, a, e2);
2
3  GM  = 3.986004418e14; % m^3/s^2 GravitationalConstant
4
5  c20 = -sqrt(5)*4.8416685e-4;
6  Latc = atan((1-e2)*tan(Lat));
7
8  % geocentric
9  P2 = P*P;
10 a2 = a*a;
11 ax = 3*c20*a2/P2;
12 GMoverP2 = GM/P2;
13 sLatc = sin(Latc);
14 cLatc = cos(Latc);
15 g_n = -GMoverP2 * ax*sLatc*cLatc;
16 g_d =  GMoverP2 * (1 + ax/2*(3*sLatc^2-1));
17
18 g_n = -1.0 * g_n; % match Savage's model ?
19
20 % geodetic
21 Alpha = Lat - Latc;
22 cAlpha = cos(Alpha);
23 sAlpha = sin(Alpha);
24 g_N =  g_n*cAlpha + g_d*sAlpha;
25 g_D = -g_n*sAlpha + g_d*cAlpha;
26
27 g = [g_N; 0; g_D];
28
29 %keyboard
30
31 return
32
33 % gravity model
34 Omega   = 7.292115e-5; % rad/s (earth's rate)
```

```
a = 6378137;    % m ( semi-major axis)
b = 6356752.3142;
f = (a-b)/a;
e2 = f*(2-f);
GM = 3.986005e14; % m^3/s^2

P = a;

Latc = (0:100)/100*pi/2;

% ------------------------------
sLatc = sin(Latc);
cLatc = cos(Latc);

b2 = b*b;

% affect of centropital accel.
w_e2i_n = [ Omega*cos(Lat);
            zeros(size(Lat));
           -Omega*sin(Lat)];

ax = 1-e2*sin(Lat).^2;
R_N = a./sqrt(ax);
h = 0;

% --------------------------------
g = GM/a2;

(gD(end)-gD(1))/g * 1e3

close all

figure
plot(Latc*180/pi, gN/g*1e3,'b'); hold
plot(Latc*180/pi, gD/g*1e3,'r');

plot(Latc*180/pi, dg_cross(1,:)/g*1e3,'k--');
plot(Latc*180/pi, dg_cross(2,:)/g*1e3,'g--');
plot(Latc*180/pi, dg_cross(3,:)/g*1e3,'m--');

title(' Geodetic, deltas g(North) & g(Down), [ mg ]')

dgT = dg_cross;
dgT(1,:) = dgT(1,:)+gN;
dgT(3,:) = dgT(3,:)+gD;

( dgT(3,end)- dgT(3,1))/g * 1e3
```

```
82
83 figure
84 plot(Latc*180/pi, dgT(1,:)/g*1e3,'b'); hold
85 plot(Latc*180/pi, dgT(2,:)/g*1e3,'g');
86 plot(Latc*180/pi, dgT(3,:)/g*1e3,'r');
87
88 title('Geodetic, total deltas g(N)[blue],g(E)[green],g(D)[red], [mg]')
```

Integrating Euler Angle

```
1  function [dotPhi,dotTheta,dotPsi] = integrateEuler(euler_k,omega,dt)
2
3  phi = euler_k(1);
4  theta = euler_k(2);
5  psi = euler_k(3);
6  w1 = omega(1);
7  w2 = omega(2);
8  w3= omega(3);
9  cTheta = cos(theta);
10 dotPsi = (sin(phi)*w2+cos(phi)*w3)*dt/cTheta+psi;
11 dotTheta = (cos(phi)*sin(theta)*w2-sin(phi)*cos(theta)*w3)*dt/cTheta + theta;
12 dotPhi = (cos(theta)*w1+sin(phi)*sin(theta)*w2+cos(phi)*sin(theta)*w3)*dt/cTheta
       + phi;
13 end
```

Global Frame to Earth Frame conversion

```
1  function r_wrtE_e = LLA2rWrtEinE(LLA, ...
2                    EarthSemiMajorAxis, EarthEccentricitySq )
3  if (nargin == 1)
4     EarthSemiMajorAxis = 6378137.0;
5     EarthEccentricitySq = 6.694380004260835e-003;
6  end
7
8  cLon  = cos(LLA(1,:));
9  sLon  = sin(LLA(1,:));
10 cLat  = cos(LLA(2,:));
11 sLat  = sin(LLA(2,:));
12
13 R_N  = EarthSemiMajorAxis./sqrt(1-EarthEccentricitySq*sLat.^2);
14 h = LLA(3,:);
15 r_wrtE_e = [(R_N + h).*cLat.*cLon;
```

```
            (R_N + h).*cLat.*sLon;
            (R_N*(1-EarthEccentricitySq) + h).*sLat];
end
```

Skew-symmetric Matrix Construction

```
function skew_mtx = skew(x)
skew_mtx=[0 -x(3) x(2) ; x(3) 0 -x(1) ; -x(2) x(1) 0 ];
end
```

From Reference Trajectory to IMU Readings

```
function [sensor,GTStates] =
    simulateIMUFromPose(GTPose,fs,longitude_ini,lattitude_ini)
% Description: Inversed Inertial Navigation System to obtain IMU measurements
    from
% positionestimates
%
% Input:
% - GTPose: seven attributeTypes
%     - x:column vector, meter
%     - y:column vector, meter
%     - z:column vector, meter
%     - roll:column vector, radian
%     - pitch:column vector, radian
%     - yaw:column vector, radian
%     - name
% - fs: simulated IMU sampling rate
% - longitude_ini: longitude of the starting point
% - lattitude_ini: lattitude of the starting point
% Output:
% - sensor: four attributeType
%   - f_b2i_b: accelerometer measurements, m/s^2, 3xN
%   - w_b2i_b: gyroscope measurements, rand/s, 3xN
%   - dt: sampling period, s, row vector
%   - name
% - GTStates: seven attributeTypes
%     - x:column vector, meter
%     - y:column vector, meter
%     - z:column vector, meter
%     - roll:column vector, radian
%     - pitch:column vector, radian
```

```
%     - yaw:column vector, radian
%     - name

earth_radius = 6378137; % earth radius
earthrate=7.2921150e-5; % rad/s
e2  = 6.694380004260835e-3; % EarthEccentricitySq
g = 9.8198;
dt = 1/fs;
GTStates.pos = [GTPose.x;GTPose.y;GTPose.z];
GTStates.lattitude = GTStates.pos(1,:)/earth_radius + lattitude_ini;
GTStates.longitude = GTStates.pos(2,:)./cos(GTStates.lattitude)/earth_radius +
    longitude_ini;
% extract velocity based on true displacement
GTStates.v_n(1,:) = [0 diff(GTStates.pos(1,:))/dt];
GTStates.v_n(2,:) = [0 diff(GTStates.pos(2,:))/dt];
GTStates.v_n(3,:) = [0 diff(GTStates.pos(3,:))/dt];
GTStates.euler_vect = [GTPose.roll; GTPose.pitch;GTPose.yaw];
GTStates.timestamps = GTPose.timestamps;
GTStates.name = GTPose.name;

%% reconstruct IMU measurements

N = length(GTStates.pos);

GTStates.dot_v_n = [];
GTStates.C_b2n = eulr2dcmCSJ(GTStates.euler_vect(:,1))';
GTStates.w_n2b_b = [];
GTStates.w_e2n_n = [];
GTStates.w_i2e_n = [];

sensor.f_b2i_b = [];
sensor.w_b2i_b = [];
sensor.dt = [];
sensor.name = GTStates.name;
C_n2b = GTStates.C_b2n';
for kk = 1:N-1
    cLat = cos(GTStates.lattitude(kk));
    sLat = sin(GTStates.lattitude(kk));
    h = GTStates.pos(3,kk);
    sensor.dt(kk) = dt;
    w_i2e_n = [earthrate*cLat; 0 ; -earthrate*sLat];
    w_e2n_n =
        [GTStates.v_n(2,kk)/(earth_radius);-GTStates.v_n(1,kk)/(earth_radius);0];
    GTStates.w_e2n_n(:,kk) = w_e2n_n;
    GTStates.w_i2e_n(:,kk) = w_i2e_n;
    GTStates.dot_v_n(:,kk) =
        (GTStates.v_n(:,kk+1)-GTStates.v_n(:,kk))/sensor.dt(kk);
```

```
    % compute local gravity, w/ or w/o centrifugal forces
    ax   = 1-e2*sLat^2;
    R_N  = earth_radius./sqrt(ax);
    r_e_n         = [-R_N*e2*sLat*cLat; 0; -R_N*ax - h];
    r_wrtE_e_l =
        LLA2rWrtEinE([GTStates.longitude(kk);GTStates.lattitude(kk);GTStates.pos(3,kk)]);
    dg_centropital_l = cross(w_i2e_n, cross(w_i2e_n, r_e_n));
    g_vector = gravityModel(GTStates.lattitude(kk), norm(r_e_n), earth_radius,
        e2);
    g_l_n = g_vector - dg_centropital_l;
    GTStates.w_n2b_b(:,kk) =
        gimbalRate2bodyRate(GTStates.euler_vect(:,kk+1),GTStates.euler_vect(:,kk),dt);
    sensor.w_b2i_b(:,kk) = GTStates.w_n2b_b(:,kk)+C_n2b*(w_i2e_n+w_e2n_n);
    C_n2b = eulr2dcmCSJ(GTStates.euler_vect(:,kk+1));
    GTStates.C_b2n(:,:,kk+1) = C_n2b';
    sensor.f_b2i_b(:,kk) =
        C_n2b*(GTStates.dot_v_n(:,kk)+cross(2*w_i2e_n+w_e2n_n,GTStates.v_n(:,kk))-g_l_n);
end
sensor.timestamps = GTStates.timestamps(1:end-1);
```

B.3.2 Navigation Library

Strapdown Inertial Navigation System

```
function est = strapDownINS(sensor,groundTrueStates)

earth_radius = 6378137; % earth radius
earthrate=7.2921150e-5; % eearth rate, rad/s
g = 9.8198; % gravity
e2  = 6.694380004260835e-3; % EarthEccentricitySq

lattitude_ini = groundTrueStates.lattitude(1);
longitude_ini = groundTrueStates.longitude(1);
accel_ini = mean(sensor.f_b2i_b(1:3,1:200)')';

% accel_ini = mean(sensor.f_b2i_b(1:3,2:3)')' - earthrate^2*(earth_radius...
% +groundTrueStates.pos(3,1))/2*[sin(2*groundTrueStates.lattitude(1));0;1...
% +cos(2*groundTrueStates.lattitude(1))];

%initialization
% roll_ini = atan2(accel_ini(2),accel_ini(3)); %
% pitch_ini = atan2(-accel_ini(1),sqrt(accel_ini(2)^2+accel_ini(3)^2));

```

```
roll_ini = atan2(-accel_ini(2),-accel_ini(3)); %
pitch_ini = atan2(accel_ini(1),sqrt(accel_ini(2)^2+accel_ini(3)^2));

% roll_ini = groundTrueStates.euler_vect(1,1); %
% pitch_ini = groundTrueStates.euler_vect(2,1); %
yaw_ini = groundTrueStates.euler_vect(3,1); %
est.euler_vect(:,1) = [roll_ini;pitch_ini;yaw_ini];
est.q_b2n(:,1) = eulr2quaCSJ(est.euler_vect(:,1));
est.vel(:,1) = [0;0;0];
est.pos(:,1) = [0;0;0]; % [N,E,D]
est.dotv_n(:,1) = [0;0;0]; % [N,E,D]
est.w_n2b_b(:,1) = [0;0;0]; % [N,E,D]
est.w_e2n_n(:,1) = [0;0;0]; % [N,E,D]
est.w_i2e_n(:,1) = [0;0;0]; % [N,E,D]
C_b2n = eulr2dcmCSJ(est.euler_vect(:,1))';
est.C_b2n(:,:,1) = C_b2n;
est.LLA(:,1) = [lattitude_ini;longitude_ini;0]; % lattitude, longitude
M = length(sensor.f_b2i_b);
est.timestamps = groundTrueStates.timestamps;

for kk = 1:M
    cLat = cos(est.LLA(1,kk));
    sLat = sin(est.LLA(1,kk));
    h = est.LLA(3,kk);
    ax   = 1-e2*sLat^2;
    R_N  = earth_radius./sqrt(ax);
    r_e_n         = [-R_N*e2*sLat*cLat; 0; -R_N*ax - h];

    w_i2e_n = [earthrate*cos(est.LLA(1,kk)); 0 ; ...
        -earthrate*sin(est.LLA(1,kk))];
    w_e2n_n = [est.vel(2,kk)/(earth_radius);...
        -est.vel(1,kk)/(earth_radius);...
        0];

    est.w_e2n_n(:,kk) = w_e2n_n;
    est.w_i2e_n(:,kk) = w_i2e_n;

    w_n2b_b = sensor.w_b2i_b(:,kk) - C_b2n'*(w_i2e_n+w_e2n_n);
    est.w_n2b_b(:,kk) = w_n2b_b;
    C_b2n = C_b2n*(eye(3) + skew(w_n2b_b)*sensor.dt(kk)); % small angle
    C_b2n = eulr2dcmCSJ(dcm2eulrCSJ(C_b2n))';
    est.C_b2n(:,:,kk+1) = C_b2n;
    est.q_b2n(:,kk+1) = dcm2quaCSJ(C_b2n);
    est.euler_vect(:,kk+1) = dcm2eulrCSJ(C_b2n);

    % compute local gravity, w/ or w/o centrifugal forces
    r_wrtE_e_l = LLA2rWrtEinE(est.LLA([2,1,3],kk));
```

```
    dg_centropital_l = cross(w_i2e_n, cross(w_i2e_n, r_e_n));
    g_vector = gravityModel(est.LLA(1,kk), norm(r_e_n), earth_radius, e2);
    g_l_n = g_vector - dg_centropital_l;
    dot_v_n = C_b2n*sensor.f_b2i_b(:,kk)...
        -cross(2*w_i2e_n+w_e2n_n,est.vel(:,kk))+g_l_n;
    est.dotv_n(:,kk) = dot_v_n;
    est.vel(:,kk+1) = est.vel(:,kk) + dot_v_n*sensor.dt(kk);
    est.pos(:,kk+1) = est.pos(:,kk) + est.vel(:,kk)*sensor.dt(kk) +
        dot_v_n*sensor.dt(kk)^2;

    lattitude_temp = est.pos(1,kk+1)/earth_radius + lattitude_ini;
    longitude_temp = est.pos(2,kk+1)./cos(lattitude_temp)/earth_radius...
        + longitude_ini;
    height_temp = est.pos(3,kk+1);
    est.LLA(:,kk+1) = [lattitude_temp;longitude_temp;height_temp];
end
```

Naive Zero-velocity Reset

```
function est = INSVelocityReset(sensor,groundTrueStates)

% constant defined
earth_radius = 6378137; % earth radius
earthrate=7.2921150e-5; % eearth rate, rad/s
g = 9.8198; % gravity
e2  = 6.694380004260835e-3; % EarthEccentricitySq

% stance phase detection
threshold = exp(-3);
% -3.996 for no noise,
% -3 for noise,
% -1.2 for real,
% -3.5 for exp400
ZUPTdetector = @accelMovVarDetector;
% ZUPTdetector = @accelMoveVarGyroEnergyDetector;
cal_period = 3*round(1/mean(sensor.dt));
% Detector Choice:
% - accelMovVarDetector for simulation
% - accelMoveVarGyroEnergyDetector for experiment

%initialization

lattitude_ini = groundTrueStates.lattitude(1);
longitude_ini = groundTrueStates.longitude(1);
```

```
accel_ini = mean(sensor.f_b2i_b(1:3,1:200)')';
roll_ini = atan2(-accel_ini(2),-accel_ini(3)); %
pitch_ini = atan2(accel_ini(1),sqrt(accel_ini(2)^2+accel_ini(3)^2));

% roll_ini = groundTrueStates.euler_vect(1,1); %
% pitch_ini = groundTrueStates.euler_vect(2,1); %
yaw_ini = groundTrueStates.euler_vect(3,1); %
est.euler_vect(:,1) = [roll_ini;pitch_ini;yaw_ini];
est.q_b2n(:,1) = eulr2quaCSJ(est.euler_vect(:,1));
est.vel(:,1) = [0;0;0];
est.pos(:,1) = [0;0;0]; % [N,E,D]
est.dotv_n(:,1) = [0;0;0]; % [N,E,D]
est.w_n2b_b(:,1) = [0;0;0]; % [N,E,D]
est.w_e2n_n(:,1) = [0;0;0]; % [N,E,D]
est.w_i2e_n(:,1) = [0;0;0]; % [N,E,D]
C_b2n = eulr2dcmCSJ(est.euler_vect(:,1))';
est.C_b2n(:,:,1) = C_b2n;
est.LLA(:,1) = [lattitude_ini;longitude_ini;0]; % lattitude, longitude
M = length(sensor.f_b2i_b);
est.timestamps = groundTrueStates.timestamps;
% est.test_statistics = accelMovVarDetector(sensor);
est.test_statistics = ZUPTdetector(sensor); %
est.threshold = threshold;

est.accRoll = [];
est.accPitch = [];

for kk = 1:M
    cLat = cos(est.LLA(1,kk));
    sLat = sin(est.LLA(1,kk));
    h = est.LLA(3,kk);
    ax   = 1-e2*sLat^2;
    R_N  = earth_radius./sqrt(ax);
    r_e_n         = [-R_N*e2*sLat*cLat; 0; -R_N*ax - h];

    w_i2e_n = [earthrate*cos(est.LLA(1,kk)); 0 ; ...
        -earthrate*sin(est.LLA(1,kk))];
    w_e2n_n = [est.vel(2,kk)/(earth_radius);...
        -est.vel(1,kk)/(earth_radius);...
        0];

    est.w_e2n_n(:,kk) = w_e2n_n;
    est.w_i2e_n(:,kk) = w_i2e_n;

    w_n2b_b = sensor.w_b2i_b(:,kk) - C_b2n'*(w_i2e_n+w_e2n_n);
    est.w_n2b_b(:,kk) = w_n2b_b;
    C_b2n = C_b2n*(eye(3) + skew(w_n2b_b)*sensor.dt(kk)); % small angle
```

```
    C_b2n = eulr2dcmCSJ(dcm2eulrCSJ(C_b2n))';
    est.C_b2n(:,:,kk+1) = C_b2n;
    est.q_b2n(:,kk+1) = dcm2quaCSJ(C_b2n);
    est.euler_vect(:,kk+1) = dcm2eulrCSJ(C_b2n);

    % compute local gravity, w/ or w/o centrifugal forces
    r_wrtE_e_l = LLA2rWrtEinE(est.LLA([2,1,3],kk));
    dg_centropital_l = cross(w_i2e_n, cross(w_i2e_n, r_e_n));
    g_vector = gravityModel(est.LLA(1,kk), norm(r_e_n), earth_radius, e2);
    g_l_n = g_vector - dg_centropital_l;

    dot_v_n = C_b2n*sensor.f_b2i_b(:,kk)...
        -cross(2*w_i2e_n+w_e2n_n,est.vel(:,kk))+g_l_n;
    est.dotv_n(:,kk) = dot_v_n;
    est.vel(:,kk+1) = est.vel(:,kk) + dot_v_n*sensor.dt(kk);
    est.pos(:,kk+1) = est.pos(:,kk) + est.vel(:,kk)*sensor.dt(kk) +
        dot_v_n*sensor.dt(kk)^2;

    lattitude_temp = est.pos(1,kk+1)/earth_radius + lattitude_ini;
    longitude_temp = est.pos(2,kk+1)./cos(lattitude_temp)/earth_radius...
        + longitude_ini;
    height_temp = est.pos(3,kk+1);
    est.LLA(:,kk+1) = [lattitude_temp;longitude_temp;height_temp];

    % calculate roll and picth angle
    est.accRoll(kk) = [atan2(-sensor.f_b2i_b(2,kk),-sensor.f_b2i_b(3,kk))];
    est.accPitch(kk) =
        [atan2(sensor.f_b2i_b(1,kk),sqrt(sensor.f_b2i_b(2,kk)^2+sensor.f_b2i_b(3,kk)^2))];
    C_b2n_update = eulr2dcmCSJ([est.accRoll(kk),...
        est.accPitch(kk),...
        est.euler_vect(3,kk)])';

    % velocity reset if
    if est.test_statistics(kk)<threshold
        est.vel(:,kk+1) = 0;
        C_b2n = C_b2n_update;
    end

end
```

Zero-velocity Update in EKF

```
function est = ZUPTwEKF(sensor,groundTrueStates,ifLocalFrame)

```

```
% navigation constant
earth_radius = 6378137; % earth radius
earthrate=7.2921150e-5; % eearth rate, rad/s
g = 9.8198; % gravity
e2  = 6.694380004260835e-3; % EarthEccentricitySq

% % filter parameter settings
% ramdomly selected
% sensor_arw = 0.0005;
% sensor_vrw = 0.01;
% sensor_rrw = 1e-8;
% sensor_acrw = 1e-5;
% zupt_std = 0.02;

% VN-200
sensor_arw = 6.1087e-04; % 6.1087e-04
sensor_vrw = 0.0137; % 0.0137
sensor_rrw = 4.8481e-05; % 4.8481e-05
sensor_acrw = 3.9240e-04; % 3.9240e-04
zupt_std = 0.05;
zaru_std = 0.02;

threshold_ZUPT = exp(-2.5);
threshold_ZART = exp(-4.1);

ZUPTdetector = @accelMovVarDetector;
% ZUPTdetector = @accelMoveVarGyroEnergyDetector;
cal_period = 5*round(1/mean(sensor.dt));
% Detector Choice:
% - accelMovVarDetector for simulation
% - accelMoveVarGyroEnergyDetector for experiment

%initialization
lattitude_ini = groundTrueStates.lattitude(1);
longitude_ini = groundTrueStates.longitude(1);
accel_ini = mean(sensor.f_b2i_b(1:3,1:cal_period),2);

roll_ini = atan2(-accel_ini(2),-accel_ini(3)); %
pitch_ini = atan2(accel_ini(1),sqrt(accel_ini(2)^2+accel_ini(3)^2));

% roll_ini = groundTrueStates.euler_vect(1,1); %
% pitch_ini = groundTrueStates.euler_vect(2,1); %
% yaw_ini = groundTrueStates.euler_vect(3,1); %
yaw_ini = 0*pi/180; %
% yaw_ini = (-20+15.99)*pi/180; %

est.euler_vect(:,1) = [roll_ini;pitch_ini;yaw_ini];
```

```
est.q_b2n(:,1) = eulr2quaCSJ(est.euler_vect(:,1));
est.vel(:,1) = [0;0;0];
est.pos(:,1) = [0;0;0]; % [N,E,D]
est.dotv_n(:,1) = [0;0;0]; % [N,E,D]
est.w_n2b_b(:,1) = [0;0;0]; % [N,E,D]
est.w_e2n_n(:,1) = [0;0;0]; % [N,E,D]
est.w_i2e_n(:,1) = [0;0;0]; % [N,E,D]
C_b2n = eulr2dcmCSJ(est.euler_vect(:,1))';
est.C_b2n(:,:,1) = C_b2n;
est.LLA(:,1) = [lattitude_ini;longitude_ini;0]; % lattitude, longitude
M = length(sensor.f_b2i_b);
est.timestamps = groundTrueStates.timestamps;
est.test_statistics = ZUPTdetector(sensor); %
est.threshold = threshold_ZUPT;
est.threshold_ZART = threshold_ZART;

est.accRoll = [];
est.accPitch = [];
est.b_g = mean(sensor.w_b2i_b(:,cal_period-400:cal_period),2);
est.b_a = zeros(3,1);

kf.P(:,:,1) = diag([zeros(1,15)]);
kf.Q = diag([sensor_arw*ones(1,3),sensor_vrw*ones(1,3),zeros(1,3),...
    sensor_rrw*ones(1,3),sensor_acrw*ones(1,3).*[1 1 1]].^2)*sensor.dt(1);
kf.R = eye(3)*(zupt_std.^2);
kf.R_ZART = eye(3)*(zaru_std.^2);

kf.H_ZUPT = zeros(3,15);
kf.H_ZUPT(:,4:6) = eye(3);
kf.H_ZART = zeros(3,15);
kf.H_ZART(:,10:12) = eye(3);
kf.dx = zeros(15,1);
O33 = zeros(3,3);
I33 = eye(3);
I1515 = eye(15);

treadmillAngle = 0; % treadmill angle (degree) wrt to IMU,

treadmillVel = rotz(treadmillAngle)*movmean(groundTrueStates.v_n',50)';
for kk = 1:M

    % compensate sensor measurement with bias
    sensor.w_b2i_b(:,kk) = sensor.w_b2i_b(:,kk) - est.b_g(:,kk);
    sensor.f_b2i_b(:,kk) = sensor.f_b2i_b(:,kk) - est.b_a(:,kk);

    % Inertial Naivgation
    cLat = cos(est.LLA(1,kk));
```

```
97      sLat = sin(est.LLA(1,kk));
98      h = est.LLA(3,kk);
99      ax   = 1-e2*sLat^2;
100     R_N  = earth_radius./sqrt(ax);
101     r_e_n         = [-R_N*e2*sLat*cLat; 0; -R_N*ax - h];
102
103     w_i2e_n = [earthrate*cLat; 0 ; ...
104               -earthrate*sLat];
105     w_e2n_n = [est.vel(2,kk)/(earth_radius);...
106               -est.vel(1,kk)/(earth_radius);0];
107
108     est.w_e2n_n(:,kk) = w_e2n_n;
109     est.w_i2e_n(:,kk) = w_i2e_n;
110
111     w_n2b_b = sensor.w_b2i_b(:,kk) - C_b2n'*(w_i2e_n+w_e2n_n);
112     est.w_n2b_b(:,kk) = w_n2b_b;
113     % Integration in discrete time
114     C_b2n = C_b2n*expm(skew(w_n2b_b)*sensor.dt(kk)); % small angle
115     C_b2n = normalizeMTX(C_b2n);
116     est.C_b2n(:,:,kk+1) = C_b2n;
117     est.q_b2n(:,kk+1) = dcm2quaCSJ(C_b2n);
118     est.euler_vect(:,kk+1) = dcm2eulrCSJ(C_b2n);
119
120     % compute local gravity, w/ or w/o centrifugal forces
121     r_wrtE_e_l = LLA2rWrtEinE(est.LLA([2,1,3],kk));
122     dg_centropital_l = cross(w_i2e_n, cross(w_i2e_n, r_e_n));
123     g_vector = gravityModel(est.LLA(1,kk), norm(r_e_n), earth_radius, e2);
124     g_l_n = g_vector - dg_centropital_l;
125
126     dot_v_n = C_b2n*sensor.f_b2i_b(:,kk)...
127         -cross(2*w_i2e_n+w_e2n_n,est.vel(:,kk))+g_l_n;
128     est.dotv_n(:,kk) = dot_v_n;
129     est.vel(:,kk+1) = est.vel(:,kk) + dot_v_n*sensor.dt(kk);
130     est.pos(:,kk+1) = est.pos(:,kk) + est.vel(:,kk)*sensor.dt(kk) +
        dot_v_n*sensor.dt(kk)^2;
131
132     lattitude_temp = est.pos(1,kk+1)/earth_radius + lattitude_ini;
133     longitude_temp = est.pos(2,kk+1)./cos(lattitude_temp)/earth_radius...
134         + longitude_ini;
135     height_temp = est.pos(3,kk+1);
136     est.LLA(:,kk+1) = [lattitude_temp;longitude_temp;height_temp];
137
138     % Error State EKF
139     % Prediction step:
140     A = [-skew(w_i2e_n+w_e2n_n) O33 O33 -C_b2n O33;
141         skew(C_b2n*sensor.f_b2i_b(:,kk)) -skew(2*w_i2e_n+w_e2n_n) O33 O33 C_b2n;
142         O33 I33 O33 O33 O33;
```

```
143         O33 O33 O33 O33 O33;
144         O33 O33 O33 O33 O33];
145     F = expm(A*sensor.dt(kk));
146 
147     kf.P(:,:,kk+1) = F*kf.P(:,:,kk)*F'+kf.Q;
148 
149     % EKF Update step: Zero Velocity Update
150     z = []; % measurement
151     R = []; % measurement noise matrix
152     H = []; % measurement model
153     dx = zeros(15,1); % error state
154     % calculate roll and picth angle (not used here)
155     est.accRoll(kk) = [atan2(-sensor.f_b2i_b(2,kk),-sensor.f_b2i_b(3,kk))];
156     est.accPitch(kk) = [atan2(sensor.f_b2i_b(1,kk),sqrt(sensor.f_b2i_b(2,kk)^2...
            ...
157         +sensor.f_b2i_b(3,kk)^2))];
158 
159     % zero velocity update
160     if est.test_statistics(kk)<est.threshold && 1
161         if ifLocalFrame % if on treadmill
162             z = [z;treadmillVel(:,kk+1)-est.vel(:,kk+1)];
163             H = [H;kf.H_ZUPT];
164             R = diag([diag(R),diag(kf.R).^0.00001]);
165         else
166             z = [z;0-est.vel(:,kk+1)];
167             H = [H; kf.H_ZUPT];
168             R = diag([diag(R);diag(kf.R)]);
169         end
170     end
171 
172     if ~isempty(z)
173         Sk = H*kf.P(:,:,kk+1)*H'+R;
174         Kk = kf.P(:,:,kk+1)*H'*inv(Sk);
175         kf.P(:,:,kk+1) = (I1515-Kk*H)*kf.P(:,:,kk+1);
176         dx = Kk*z;
177     end
178     kf.P(:,:,kk+1) = (kf.P(:,:,kk+1)+kf.P(:,:,kk+1)')/2;
179     kf.dx(:,kk) = dx;
180     % finish estimating errors, update navigation solutions
181     % orientation
182     C_b2n = (I33-skew(dx(1:3)))*C_b2n;
183     est.C_b2n(:,:,kk+1) = C_b2n;
184     est.q_b2n(:,kk+1) = dcm2quaCSJ(C_b2n);
185     est.euler_vect(:,kk+1) = dcm2eulrCSJ(C_b2n);
186     % velocity
187     est.vel(:,kk+1) = est.vel(:,kk+1)+dx(4:6);
188     % position
```

```
    est.pos(:,kk+1) = est.pos(:,kk+1)+dx(7:9);
    lattitude_temp = est.pos(1,kk+1)/earth_radius + lattitude_ini;
    longitude_temp = est.pos(2,kk+1)./cos(lattitude_temp)/earth_radius...
        + longitude_ini;
    height_temp = est.pos(3,kk+1);
    est.LLA(:,kk+1) = [lattitude_temp;longitude_temp;height_temp];
    % IMU biases
    est.b_g(:,kk+1) = est.b_g(:,kk) - dx(10:12);
    est.b_a(:,kk+1) = est.b_a(:,kk) - dx(13:15);
end

disp('Implementation Ends')
```

B.3.3 IMU Noise Library

Deterministic Noise

```
function noise_sensor = addDeterministicErrors(raw_sensor,noiseSettings,ifRandom)

% mode 0: no deterministic noise
% mode 1: w/ turn-on biases
% mode 2: w/ scale factor inconsistency
% mode 3: w/ mis-alignment
% mode 4: w/ g-sensitivity
% mode 5: w/ the point of percussion correction factor

noise_sensor = raw_sensor;
bias_sensor = raw_sensor;

mis_align_angle = 0.03; % deg
accel_bias = 0.01; % g
gyro_bias = 0.3; % deg/s

if ifRandom == 1
    accel_SF_error = sin(mis_align_angle*randn(1,3)*pi/180);
    gyro_SF_error = sin(mis_align_angle*randn(1,3)*pi/180);

    accel_ML_error = sin(mis_align_angle*randn(1,3)*pi/180);
    gyro_ML_error = sin(mis_align_angle*randn(1,3)*pi/180);

    accel_turn_on_bias = accel_bias*9.81*randn(3,1); % m/s^2
    gyro_turn_on_bias = gyro_bias*pi/180*randn(3,1); % rad

```

```
27      accel_SF_error_M = diag(accel_SF_error); % accel scale factor
28      gyro_SF_error_M = diag(gyro_SF_error); % gyro scale factor
29
30      accel_ML_error_M = skew(accel_ML_error); % accel misalignment
31      gyro_ML_error_M = skew(gyro_ML_error); % gyro misalignment
32  else
33      accel_turn_on_bias = accel_bias*9.81*randn(3,1); % m/s^2, bias always ramdom
34      gyro_turn_on_bias = gyro_bias*pi/180*randn(3,1); % rad
35
36      accel_SF_error = sin(mis_align_angle*ones(1,3)*pi/180);
37      gyro_SF_error = sin(mis_align_angle*ones(1,3)*pi/180);
38
39      accel_ML_error = sin(mis_align_angle*ones(1,3)*pi/180);
40      gyro_ML_error = sin(mis_align_angle*ones(1,3)*pi/180);
41
42      accel_SF_error_M = diag(accel_SF_error); % accel scale factor
43      gyro_SF_error_M = diag(gyro_SF_error); % gyro scale factor
44
45      accel_ML_error_M = skew(accel_ML_error); % accel misalignment
46      gyro_ML_error_M = skew(gyro_ML_error); % gyro misalignment
47  end
48
49  if noiseSettings.iniBias == 1
50      disp(['Set Accel Turn-on bias to ' num2str(accel_turn_on_bias') ' [m/s^2]'])
51      disp(['Set Gyro Turn-on bias to ' num2str(gyro_turn_on_bias') ' [rad/s]'])
52      bias_sensor.f_b2i_b = bias_sensor.f_b2i_b + accel_turn_on_bias;
53      bias_sensor.w_b2i_b = bias_sensor.w_b2i_b + gyro_turn_on_bias;
54  end
55
56  if noiseSettings.scaleFactor == 1
57      disp(['Set Accel SF error to ' num2str(accel_SF_error) ' [deg]'])
58      disp(['Set Gyro SF error to ' num2str(gyro_SF_error) ' [deg]'])
59      noise_sensor.f_b2i_b = noise_sensor.f_b2i_b +
            accel_SF_error_M*bias_sensor.f_b2i_b;
60      noise_sensor.w_b2i_b = noise_sensor.w_b2i_b +
            gyro_SF_error_M*bias_sensor.w_b2i_b;
61  end
62
63  if noiseSettings.misalignment == 1
64      disp(['Set Accel ML error to ' num2str(accel_ML_error) ' [deg]'])
65      disp(['Set Gyro ML error to ' num2str(gyro_ML_error) ' [deg]'])
66      noise_sensor.f_b2i_b = noise_sensor.f_b2i_b +
            accel_ML_error_M*bias_sensor.f_b2i_b;
67      noise_sensor.w_b2i_b = noise_sensor.w_b2i_b +
            gyro_ML_error_M*bias_sensor.w_b2i_b;
68  end
```

FSR and bandwidth

```
function noise_sensor = SimSaturateIMU(raw_sensor,noiseSettings, accelFSR,
    gyroFSR, accelBW, gyroBW)

noise_sensor = raw_sensor;
fs = 1/(raw_sensor.timestamps(2));

% try IIR filter LP
if noiseSettings.bandWidth == 1

    disp(['Set Accel BW to ' num2str(accelBW) ' [Hz]']);

    % accel bandwidth
    fc = accelBW;
    filter_order = 6;
    [b,a] = butter(filter_order,fc/(fs/2));

    noise_sensor.f_b2i_b(1,:) = filter(b,a,noise_sensor.f_b2i_b(1,:));
    noise_sensor.f_b2i_b(2,:) = filter(b,a,noise_sensor.f_b2i_b(2,:));
    noise_sensor.f_b2i_b(3,:) = filter(b,a,noise_sensor.f_b2i_b(3,:));

    % gyro bandwidth

    disp(['Set Gyro BW to ' num2str(gyroBW) ' [Hz]']);
    fc = gyroBW;

    [b,a] = butter(filter_order,fc/(fs/2));
    noise_sensor.w_b2i_b(1,:) = filter(b,a,noise_sensor.w_b2i_b(1,:));
    noise_sensor.w_b2i_b(2,:) = filter(b,a,noise_sensor.w_b2i_b(2,:));
    noise_sensor.w_b2i_b(3,:) = filter(b,a,noise_sensor.w_b2i_b(3,:));
end

% FIR LP
if noiseSettings.bandWidth == 1 && 0
    % accel bandwidth
    disp(['Set Accel BW to ' num2str(accelBW) ' [Hz]']);
    fc = accelBW;

    noise_sensor.f_b2i_b(1,:) = lowpass(noise_sensor.f_b2i_b(1,:),fc,fs);
    noise_sensor.f_b2i_b(2,:) = lowpass(noise_sensor.f_b2i_b(2,:),fc,fs);
    noise_sensor.f_b2i_b(3,:) = lowpass(noise_sensor.f_b2i_b(3,:),fc,fs);

    % gyro bandwidth

    fc = gyroBW;
    disp(['Set Gyro BW to ' num2str(gyroBW) ' [Hz]']);
```

```
    noise_sensor.w_b2i_b(1,:) = lowpass(noise_sensor.w_b2i_b(1,:),fc,fs);
    noise_sensor.w_b2i_b(2,:) = lowpass(noise_sensor.w_b2i_b(2,:),fc,fs);
    noise_sensor.w_b2i_b(3,:) = lowpass(noise_sensor.w_b2i_b(3,:),fc,fs);
end

if noiseSettings.FSR ==1
    disp(['Set Accel FSR to ' num2str(accelFSR/9.81) ' [g]']);
    % accel FSR
    noise_sensor.f_b2i_b(3,noise_sensor.f_b2i_b(3,:)>accelFSR) = accelFSR;
    noise_sensor.f_b2i_b(3,noise_sensor.f_b2i_b(3,:)<-accelFSR) = -accelFSR;

    % gyro FSR
    disp(['Set Gyro FSR to ' num2str(gyroFSR*180/pi), '[dps]']);
    noise_sensor.w_b2i_b(1,noise_sensor.w_b2i_b(1,:)>gyroFSR) = gyroFSR;
    noise_sensor.w_b2i_b(2,noise_sensor.w_b2i_b(2,:)>gyroFSR) = gyroFSR;
    noise_sensor.w_b2i_b(3,noise_sensor.w_b2i_b(3,:)>gyroFSR) = gyroFSR;
    noise_sensor.w_b2i_b(1,noise_sensor.w_b2i_b(1,:)<-gyroFSR) = -gyroFSR;
    noise_sensor.w_b2i_b(2,noise_sensor.w_b2i_b(2,:)<-gyroFSR) = -gyroFSR;
    noise_sensor.w_b2i_b(3,noise_sensor.w_b2i_b(3,:)<-gyroFSR) = -gyroFSR;
end
```

B.3.4 Plotting Library

Animate Navigation Solution

```
function animatePath(path)

    plot3(path.x,path.y,path.z,'b','LineWidth',2);grid on;hold on;
    plot3(path.x(end),path.y(end),path.z(end),'r^','LineWidth',2)
    plot3(path.x(1),path.y(1),path.z(1),'ks','LineWidth',2)
    title('Trajectory')
    xlabel('North, m')
    ylabel('East, m')
    zlabel('Down, m')
    %    xlim([-1 5])
    ylim([-1 1])
    zlim([-1 1])
    axis equal

    x_arrow = [1,0,0]';
    y_arrow = [0,1,0]';
    z_arrow = [0,0,1]';
    current_pos = [path.x(1) path.y(1) path.z(1)];
```

```
    current_dcm = eulr2dcmCSJ([path.roll(1), path.pitch(1), path.yaw(1)])';
    h_x =
        mArrow3(current_pos,current_pos+(current_dcm*x_arrow)','color','red','stemWidth',0.03,'
    h_y =
        mArrow3(current_pos,current_pos+(current_dcm*y_arrow)','color','green','stemWidth',0.03
    h_z =
        mArrow3(current_pos,current_pos+(current_dcm*z_arrow)','color','blue','stemWidth',0.03,
    legend('Path','End','Start','b-frame x','b-frame y','b-frame z')
    title('Trajectory')
    %     view(60,15)

    for kk = 1:1000:length(path.x)
        delete(h_x)
        delete(h_y)
        delete(h_z)
        current_pos = [path.x(kk) path.y(kk) path.z(kk)];
        current_dcm = eulr2dcmCSJ([path.roll(kk), path.pitch(kk), path.yaw(kk)])';
        h_x =
            mArrow3(current_pos,current_pos+(current_dcm*x_arrow)','color','red','stemWidth',0.0
        h_y =
            mArrow3(current_pos,current_pos+(current_dcm*y_arrow)','color','green','stemWidth',0
        h_z =
            mArrow3(current_pos,current_pos+(current_dcm*z_arrow)','color','blue','stemWidth',0.
        title(['Trajectory, frame = ',num2str(kk)]);
        legend('Path','End','Start','b-frame x','b-frame y','b-frame z')
        %     xlim([-1 5])
        ylim([-1 1])
        zlim([-1 1])
        %         view(60,15)
        drawnow
    end
end
```

Generate 3D Arrow

```
function h = mArrow3(p1,p2,varargin)
%mArrow3 - plot a 3D arrow as patch object (cylinder+cone)
%
% syntax:  h = mArrow3(p1,p2)
%          h = mArrow3(p1,p2,'propertyName',propertyValue,...)
%
% with:    p1:         starting point
%          p2:         end point
%          properties: 'color':    color according to MATLAB specification
```

```
%                                (see MATLAB help item 'ColorSpec')
%                    'stemWidth': width of the line
%                    'tipWidth':  width of the cone
%
%          Additionally, you can specify any patch object properties. (For
%          example, you can make the arrow semitransparent by using
%          'facealpha'.)
%
% example1: h = mArrow3([0 0 0],[1 1 1])
%          (Draws an arrow from [0 0 0] to [1 1 1] with default properties.)
%
% example2: h = mArrow3([0 0 0],[1 1
    1],'color','red','stemWidth',0.02,'facealpha',0.5)
%          (Draws a red semitransparent arrow with a stem width of 0.02 units.)
%
% hint:    use light to achieve 3D impression
%
propertyNames = {'edgeColor'};
propertyValues = {'none'};
%% evaluate property specifications
for argno = 1:2:nargin-2
    switch varargin{argno}
        case 'color'
            propertyNames = {propertyNames{:},'facecolor'};
            propertyValues = {propertyValues{:},varargin{argno+1}};
        case 'stemWidth'
            if isreal(varargin{argno+1})
                stemWidth = varargin{argno+1};
            else
                warning('mArrow3:stemWidth','stemWidth must be a real number');
            end
        case 'tipWidth'
            if isreal(varargin{argno+1})
                tipWidth = varargin{argno+1};
            else
                warning('mArrow3:tipWidth','tipWidth must be a real number');
            end
        otherwise
            propertyNames = {propertyNames{:},varargin{argno}};
            propertyValues = {propertyValues{:},varargin{argno+1}};
    end
end
%% default parameters
if ~exist('stemWidth','var')
    ax = axis;
    if numel(ax)==4
        stemWidth = norm(ax([2 4])-ax([1 3]))/300;
```

```
    elseif numel(ax)==6
        stemWidth = norm(ax([2 4 6])-ax([1 3 5]))/300;
    end
end
if ~exist('tipWidth','var')
    tipWidth = 3*stemWidth;
end
tipAngle = 22.5/180*pi;
tipLength = tipWidth/tan(tipAngle/2);
ppsc = 50; % (points per small circle)
ppbc = 250; % (points per big circle)
%% ensure column vectors
p1 = p1(:);
p2 = p2(:);
%% basic lengths and vectors
x = (p2-p1)/norm(p2-p1); % (unit vector in arrow direction)
y = cross(x,[0;0;1]);  % (y and z are unit vectors orthogonal to arrow)
if norm(y)<0.1
    y = cross(x,[0;1;0]);
end
y = y/norm(y);
z = cross(x,y);
z = z/norm(z);
%% basic angles
theta = 0:2*pi/ppsc:2*pi; % (list of angles from 0 to 2*pi for small circle)
sintheta = sin(theta);
costheta = cos(theta);
upsilon = 0:2*pi/ppbc:2*pi; % (list of angles from 0 to 2*pi for big circle)
sinupsilon = sin(upsilon);
cosupsilon = cos(upsilon);
%% initialize face matrix
f = NaN([ppsc+ppbc+2 ppbc+1]);
%% normal arrow
if norm(p2-p1)>tipLength
    % vertices of the first stem circle
    for idx = 1:ppsc+1
        v(idx,:) = p1 + stemWidth*(sintheta(idx)*y + costheta(idx)*z);
    end
    % vertices of the second stem circle
    p3 = p2-tipLength*x;
    for idx = 1:ppsc+1
        v(ppsc+1+idx,:) = p3 + stemWidth*(sintheta(idx)*y + costheta(idx)*z);
    end
    % vertices of the tip circle
    for idx = 1:ppbc+1
        v(2*ppsc+2+idx,:) = p3 + tipWidth*(sinupsilon(idx)*y + cosupsilon(idx)*z);
    end
```

```
    % vertex of the tiptip
    v(2*ppsc+ppbc+4,:) = p2;
    % face of the stem circle
    f(1,1:ppsc+1) = 1:ppsc+1;
    % faces of the stem cylinder
    for idx = 1:ppsc
        f(1+idx,1:4) = [idx idx+1 ppsc+1+idx+1 ppsc+1+idx];
    end
    % face of the tip circle
    f(ppsc+2,:) = 2*ppsc+3:(2*ppsc+3)+ppbc;
    % faces of the tip cone
    for idx = 1:ppbc
        f(ppsc+2+idx,1:3) = [2*ppsc+2+idx 2*ppsc+2+idx+1 2*ppsc+ppbc+4];
    end
%% only cone v
else
    tipWidth = 2*sin(tipAngle/2)*norm(p2-p1);
    % vertices of the tip circle
    for idx = 1:ppbc+1
        v(idx,:) = p1 + tipWidth*(sinupsilon(idx)*y + cosupsilon(idx)*z);
    end
    % vertex of the tiptip
    v(ppbc+2,:) = p2;
    % face of the tip circle
    f(1,:) = 1:ppbc+1;
    % faces of the tip cone
    for idx = 1:ppbc
        f(1+idx,1:3) = [idx idx+1 ppbc+2];
    end
end
%% draw
fv.faces = f;
fv.vertices = v;
h = patch(fv);
for propno = 1:numel(propertyNames)
    try
        set(h,propertyNames{propno},propertyValues{propno});
    catch
        disp(lasterr)
    end
end
```

Plot 3D Orientation

```
function plotComparedOrientation(groundTrueStates,est)
d2r = pi/180;
r2d = 180/pi;
subplot(3,1,1)
plot(groundTrueStates.timestamps,groundTrueStates.euler_vect(1,:)*r2d);hold on
plot(est.timestamps,est.euler_vect(1,:)*r2d);
title(['Roll, ', groundTrueStates.name])
legend('Simulated','Estimated')
xlabel('Time, s')
ylabel('Degree')
subplot(3,1,2)
plot(groundTrueStates.timestamps,groundTrueStates.euler_vect(2,:)*r2d);hold on
plot(est.timestamps,est.euler_vect(2,:)*r2d);
title(['Pitch, ', groundTrueStates.name])
legend('Simulated','Estimated')
xlabel('Time, s')
ylabel('Degree')
subplot(3,1,3)
plot(groundTrueStates.timestamps,groundTrueStates.euler_vect(3,:)*r2d);hold on
plot(est.timestamps,est.euler_vect(3,:)*r2d);
title(['Yaw, ', groundTrueStates.name])
legend('Simulated','Estimated')
xlabel('Time, s')
ylabel('Degree')
```

Plot Path

```
function plotComparedPath(groundTrueStates,est)

subplot(3,1,1)
plot(groundTrueStates.timestamps,groundTrueStates.pos(1,:));hold on
plot(est.timestamps,est.pos(1,:));
title(['North, ', groundTrueStates.name])
legend('Simulated','Estimated')
xlabel('Time, s')
ylabel('m')
subplot(3,1,2)
plot(groundTrueStates.timestamps,groundTrueStates.pos(2,:));hold on
plot(est.timestamps,est.pos(2,:));
title(['East, ', groundTrueStates.name])
legend('Simulated','Estimated')
xlabel('Time, s')
ylabel('m')
subplot(3,1,3)
```

```
plot(groundTrueStates.timestamps,groundTrueStates.pos(3,:));hold on
plot(est.timestamps,est.pos(3,:));
title(['Down, ', groundTrueStates.name])
legend('Simulated','Estimated')
xlabel('Time, s')
ylabel('m')
```

Plot Position Comparison

```
function plotComparedPostion(groundTrueStates,est)

plot3(est.pos(1,:),est.pos(2,:),est.pos(3,:),'b','LineWidth',2);grid on;hold on
plot3(est.pos(1,end),est.pos(2,end),est.pos(3,end),'b^','LineWidth',2)
plot3(est.pos(1,1),est.pos(2,1),est.pos(3,1),'bs','LineWidth',2)
plot3(groundTrueStates.pos(1,:),groundTrueStates.pos(2,:),groundTrueStates.pos(3,:),'r','LineWi
    on;hold on;
plot3(groundTrueStates.pos(1,end),groundTrueStates.pos(2,end),groundTrueStates.pos(3,end),'ro',
plot3(groundTrueStates.pos(1,1),groundTrueStates.pos(2,1),groundTrueStates.pos(3,1),'rs','LineW
legend('Estimated Path','Estimated End','Estimated Start','True Path','True
    End','True Start')
title(['True vs Estimated trajectory, ', groundTrueStates.name])
xlabel('North, m')
ylabel('East, m')
zlabel('Down, m')
```

Plot Velocity Comparison

```
function plotComparedVelocity(groundTrue,est)

subplot(3,1,1)
plot(groundTrue.timestamps,groundTrue.v_n(1,:));hold on
plot(est.timestamps,est.vel(1,:));
title(['Simulated Velocity along the North, ', groundTrue.name])
legend('Simulated','Estimated')
xlabel('time, s')
ylabel('m/s')

subplot(3,1,2)
plot(groundTrue.timestamps,groundTrue.v_n(2,:));hold on
plot(est.timestamps,est.vel(2,:));
title(['Simulated Velocity along the East, ', groundTrue.name])
legend('Simulated','Estimated')
```

```
xlabel('time, s')
ylabel('m/s')

subplot(3,1,3)
plot(groundTrue.timestamps,groundTrue.v_n(3,:));hold on
plot(est.timestamps,est.vel(3,:));
title(['Velocity along the Down, ', groundTrue.name])
legend('Simulated','Estimated')
xlabel('time, s')
ylabel('m/s')
```

Plot Ground Truth Path

```
function plotGroundTruePath(groundTrue,name)

if nargin<2
    name = '';
end

plot3(groundTrue.x,groundTrue.y,groundTrue.z,'b','LineWidth',2);grid on;hold on;
plot3(groundTrue.x(end),groundTrue.y(end),groundTrue.z(end),'r^','LineWidth',2)
plot3(groundTrue.x(1),groundTrue.y(1),groundTrue.z(1),'ks','LineWidth',2)
legend('Path','End','Start')
title(['Trajectory: ' name])
xlabel('x, m')
ylabel('y, m')
zlabel('z, m')
axis equal
```

Plot Navigation Error

```
function plotNavError(error)

subplot(3,1,1)
plot(error.timestamps,error.pos);
legend('x','y','z')
title('Position Error')
xlabel('Time, s')
ylabel('m')

subplot(3,1,2)
plot(error.timestamps,error.orientation*180/pi);
```

```
legend('roll','pitch','yaw')
title('Orientation Error')
xlabel('Time, s')
ylabel('Degree')

subplot(3,1,3)
plot(error.timestamps,error.vel);
legend('x','y','z')
title('Velocity Error')
xlabel('Time, s')
ylabel('m/s')
```

Plot Simulated IMU Readings

```
function plotSimulatedIMU(sensor)

subplot(2,1,1)
plot(sensor.timestamps,sensor.f_b2i_b/9.81);
xlabel('Time, s')
ylabel('g')
title(['Simulated Accelerometer readings, ', sensor.name])
legend('x','y','z')
subplot(2,1,2)
plot(sensor.timestamps,sensor.w_b2i_b*180/pi);
xlabel('Time, s')
ylabel('deg/s')
title(['Simulated Gyroscope readings, ' sensor.name])
legend('x','y','z')
```

Plot 3D Position

```
function plotPostion(est)

end_ind = length(est.pos)-round(0.05/(est.timestamps(2)-est.timestamps(1)));

plot3(est.pos(2,1:end_ind),est.pos(1,1:end_ind),-est.pos(3,1:end_ind),'Color',...
[0.00,0.45,0.74],'LineWidth',2);grid on;hold on
plot3(est.pos(2,end_ind),est.pos(1,end_ind),-est.pos(3,end_ind),'^','Color',...
[0.85,0.33,0.10],'LineWidth',2)
plot3(est.pos(2,1),est.pos(1,1),-est.pos(3,1),'ks','LineWidth',2)
legend('Path','End','Start')
title(['Estimated trajectory'])
```

```
ylabel('North, m')
xlabel('East, m')
zlabel('Down, m')
```

Plot Position on XY Plane

```
function plotXYPostion(est)

end_ind = length(est.pos)-round(0.05/(est.timestamps(2)-est.timestamps(1)));

plot(est.pos(2,1:end_ind),est.pos(1,1:end_ind),'Color',[0.00,0.45,0.74],'LineWidth',2);grid
    on;hold on
plot(est.pos(2,end_ind),est.pos(1,end_ind),'^','Color',[0.85,0.33,0.10],'LineWidth',2)
plot(est.pos(2,1),est.pos(1,1),'ks','LineWidth',2)
legend('Path','End','Start')
title(['Horizontal trajectory'])
ylabel('North, m')
xlabel('East, m')
```

Plot Position on XZ Plane

```
function plotXZPostion(est)

end_ind = length(est.pos)-round(0.05/(est.timestamps(2)-est.timestamps(1)));

plot(est.pos(1,1:end_ind),est.pos(3,1:end_ind),'Color',[0.00,0.45,0.74],'LineWidth',2);grid
    on;hold on
plot(est.pos(1,end_ind),est.pos(3,end_ind),'^','Color',[0.85,0.33,0.10],'LineWidth',2)
plot(est.pos(1,1),est.pos(3,1),'ks','LineWidth',2)
legend('Path','End','Start')
title(['Vertical trajectory'])
xlabel('North, m')
ylabel('Down, m')
```

Plot IMU Readings and Stance Phase Status

```
function plotZUPTAndIMU(sensor,est)
figure;
subplot(2,1,1)
```

```
area(est.timestamps(1:end-1), (est.test_statistics<est.threshold)*50);hold on
alpha(0.2)
plot(sensor.timestamps,sensor.f_b2i_b');
title('Stance Phase Test Statistics')
ylabel('Log Likelihood')
xlabel('Time, s')

subplot(2,1,2)
plot(est.timestamps(1:end-1), log(est.test_statistics));hold on
plot([est.timestamps(1) est.timestamps(end-1)],[log(est.threshold)
    log(est.threshold)]);
title('Stance Phase Test Statistics')
ylabel('Log Likelihood')
xlabel('Time, s')
```

B.3.5 Stance Phase Detection Library

Accelerometer Moving Variance Detector

```
function test_statistics = accelMovVarDetector(sensor)
    test_statistics = movvar(sensor.f_b2i_b(3,:),1/sensor.dt(1)/4);
    % for fs = 10000;
end
```

Accelerometer Moving Variance And Gyroscope Energy Detector

```
function test_statistics = accelMoveVarGyroEnergyDetector(sensor)
    window_size = 100;
    test_statistics = sum(movvar(sensor.f_b2i_b(1:3,:),window_size,0,2)) ...
        +
            sum(movmean(abs(sensor.w_b2i_b(1:3,:)-mean(sensor.w_b2i_b(1:3,1:100),2))...
        ,window_size,2));
end
```

SHOE detector

```
function [zupt,total,total_x,total_y,total_z] = SHOE_detector(u,simdata)
```

```
% ------------------------------------------------------------------------
% This function calculated adaptive threshld for ZUPT detector
% Input: u:    IMU readouts at current time step
%        xi:   Uncertainty of estimated velocity
%        dt:   Time differnce beween last time ZUPT is on and current step
%        ZUPT: ZUPT state of previous time step
%        thre1: Threshold of previous time step
%        shock: Maximum shock level of last step
% ------------------------------------------------------------------------

total_x = 1;
total_y = 1;
total_z = 1;

[r, c] = size(u);
W=simdata.Window_size;
sigma2_a = (simdata.sigma_a/simdata.Ts)^2;
sigma2_g = (simdata.sigma_g/simdata.Ts)^2;
g = 9.796;
u_n = mean(u(1:3, :), 2);
u_n = u_n / norm(u_n); % Unit vector along the specific force
for i = 1:3
    u(i, :) = u(i, :) - g*u_n(i);
end
total_x = sum(sum(u(1,:).^2))/sigma2_a+sum(sum(u(4,:).^2))/sigma2_g;
total_y = sum(sum(u(2,:).^2))/sigma2_a+sum(sum(u(5,:).^2))/sigma2_g;
total_z = sum(sum(u(3,:).^2))/sigma2_a+sum(sum(u(6,:).^2))/sigma2_g;

total = sum(sum(u(1:3,:).^2))/sigma2_a+sum(sum(u(4:6,:).^2))/sigma2_g;
total = total/c;

if(total < simdata.factor)
    zupt = ones(1, simdata.Window_size);
else
    zupt = zeros(1, simdata.Window_size);
end
end
```

```
function [zupt,LR,thre2] = adaptive_detector_w_downward_ultrasnoic(u, xi, ...
    dt, ZUPT, thre1, shock, shoe_height)
% ------------------------------------------------------------------------
% This function calculated adaptive threshld for ZUPT detector
% Input: u:    IMU readouts at current time step
%        xi:   Uncertainty of estimated velocity
%        dt:   Time differnce beween last time ZUPT is on and current step
%        ZUPT: ZUPT state of previous time step
%        thre1: Threshold of previous time step
```

```
%       shock: Maximum shock level of last step
%       shoe_height: readouts from downward-facing shoe-mounted ultrasonic
%       sensor
% Output: zupt: Detected ZUPT state. 1 is stance and 0 is swing
%        LR:   Likelihood ratio
%        thre2: Actual adaptive threshold
% -------------------------------------------------------------------------
global simdata;
alpha = simdata.alpha;
theta = simdata.theta;
beta = simdata.beta*0;
p = [0.0364 7.9276]; % Derived from experimental results
p = [0.032 7.9276]; % Adjusted according to differnet subjects
temp = exp(p(1)*shock + p(2));
[r, c] = size(u);
W=simdata.Window_size;
sigma2_a = (simdata.sigma_a/simdata.Ts)^2;
sigma2_g = (simdata.sigma_g/simdata.Ts)^2;
sigma2_shoe = 1;
g = 9.796;
u_n = mean(u(1:3, :), 2);
u_n = u_n / norm(u_n); % Unit vector along the specific force
for i = 1:3
    u(i, :) = u(i, :) - g*u_n(i);
end
total = sum(sum(u(1:3,:).^2))/sigma2_a+sum(sum(u(4:6,:).^2))/sigma2_g ...
    + sum(shoe_height.^2)/sigma2_shoe;
total = total/c;
thre2 = alpha + theta*temp*log(dt/0.0083) - beta*xi;
% Hold the threshold during the stance phase
if (ZUPT == 0)
    thre2 = alpha + theta*temp*log(dt/0.0083) - beta*xi;
elseif ZUPT ==1
    thre2 = thre1;
end
if(thre2 < 1000)
    thre2 = 1000;
end
if(total < thre2)
    zupt = 1;
else
    zupt = 0;
end
LR = total;
end
```

```
function [zupt,LR,thre2] = adaptive_detector_w_DVS(u, xi, dt, ZUPT, thre1, ...
```

```
    shock, firing_rate)
% --------------------------------------------------------------------
% This function calculated adaptive threshld for ZUPT detector
% Input: u:    IMU readouts at current time step
%        xi:   Uncertainty of estimated velocity
%        dt:   Time differnce beween last time ZUPT is on and current step
%        ZUPT: ZUPT state of previous time step
%        thre1: Threshold of previous time step
%        shock: Maximum shock level of last step
%        firing_rate: readouts from side-facing shoe-mounted DVS128
% Output: zupt: Detected ZUPT state. 1 is stance and 0 is swing
%         LR:   Likelihood ratio
%         thre2: Actual adaptive threshold
% --------------------------------------------------------------------
    global simdata;
    alpha = simdata.alpha;
    theta = simdata.theta;
    beta = simdata.beta*0;
    p = [0.0364 7.9276]; % Derived from experimental results
    p = [0.032 7.9276]; % Adjusted according to differnet subjects
    temp = exp(p(1)*shock + p(2));
    [r, c] = size(u);
    W=simdata.Window_size;
    sigma2_a = (simdata.sigma_a/simdata.Ts)^2;
    sigma2_g = (simdata.sigma_g/simdata.Ts)^2;
    sigma2_DVS = 1;
    g = 9.796;
    u_n = mean(u(1:3, :), 2);
    u_n = u_n / norm(u_n); % Unit vector along the specific force
    for i = 1:3
        u(i, :) = u(i, :) - g*u_n(i);
    end
    total = sum(sum(u(1:3,:).^2))/sigma2_a+sum(sum(u(4:6,:).^2))/sigma2_g ...
        + sum(firing_rate.^2)/sigma2_DVS;
    total = total/c;
    thre2 = alpha + theta*temp*log(dt/0.0083) - beta*xi;
    % Hold the threshold during the stance phase
    if (ZUPT == 0)
        thre2 = alpha + theta*temp*log(dt/0.0083) - beta*xi;
    elseif ZUPT ==1
        thre2 = thre1;
    end
    if(thre2 < 1000)
        thre2 = 1000;
    end
    if(total < thre2)
        zupt = 1;
```

```
    else
        zupt = 0;
    end
    LR = total;
end
```

B.3.6 Utility Library

Add Noise to IMU Readings

```
function noise_sensor = addIMUNoise(raw_sensor,IMU_model_name,noiseSettings)

Fs = 1/raw_sensor.dt(2);
L_n = length(raw_sensor.f_b2i_b);
noise_sensor = raw_sensor;
IMU_char = zeros(6,4);

% generate stochastic error
if isfile([pwd, '\lib\IMU_noise_model\' 'noise_model_' IMU_model_name '.mat'])
    load(['noise_model_' IMU_model_name '.mat'])
else
    load(['exp000_' IMU_model_name '.mat'])
    signals = u(1:6,740000:end)'.*[9.8198;9.8198;9.8198;pi/180;pi/180;pi/180]';
    Fs_cal = round(1/mean(u(11,:)));
    IMU_char(1,:) = opt_NBK_search(signals(:,1),Fs_cal,1);
    IMU_char(2,:) = opt_NBK_search(signals(:,2),Fs_cal,1);
    IMU_char(3,:) = opt_NBK_search(signals(:,3),Fs_cal,1);
    IMU_char(4,:) = opt_NBK_search(signals(:,4),Fs_cal,1);
    IMU_char(5,:) = opt_NBK_search(signals(:,5),Fs_cal,1);
    IMU_char(6,:) = opt_NBK_search(signals(:,6),Fs_cal,1);
    save([pwd, '\lib\IMU_noise_model\' 'noise_model_' IMU_model_name '.mat'] ...
        ,'IMU_char');
end

% manually setting noise
IMU_char(1:3,:) = ones(3,4).*[(0.04/1000*9.81)^2, 10, (0.14/1000*9.81)^2,
    1.03373018045034e-08];
IMU_char(4:6,:) = ones(3,4).*[(10/3600*pi/180)^2, 10, (0.01*pi/180)^2,
    2.0084484247909e-8];

[stochasticAccelNoise(:,1),~] = generateStochasticIMUnoise(IMU_char(1,:), ...
    Fs,L_n,noiseSettings,0);
[stochasticAccelNoise(:,2),~] = generateStochasticIMUnoise(IMU_char(2,:), ...
```

```
    Fs,L_n,noiseSettings,0);
[stochasticAccelNoise(:,3),~] = generateStochasticIMUnoise(IMU_char(3,:), ...
    Fs,L_n,noiseSettings,0);
[stochasticGyroNoise(:,1),~] = generateStochasticIMUnoise(IMU_char(4,:), ...
    Fs,L_n,noiseSettings,0);
[stochasticGyroNoise(:,2),~] = generateStochasticIMUnoise(IMU_char(5,:), ...
    Fs,L_n,noiseSettings,0);
[stochasticGyroNoise(:,3),~] = generateStochasticIMUnoise(IMU_char(6,:), ...
    Fs,L_n,noiseSettings,0);

% add both errors to the simulated IMU measurements
disp('Add Stochastic Noise')
noise_sensor.f_b2i_b = noise_sensor.f_b2i_b + stochasticAccelNoise';
noise_sensor.w_b2i_b = noise_sensor.w_b2i_b + stochasticGyroNoise';

% simulate insufficient range and BW
disp('Simulate Reading Constraints')
% raw_sensor,ifFSR, ifBW, accelFSR, gyroFSR, accelBW, gyroBW
noise_sensor = SimSaturateIMU(noise_sensor,noiseSettings, 16*9.81, ...
    2000*pi/180, min(260,floor(Fs-1)), min(260,floor(Fs-1)));

% generate deterministic error
disp('Add Deterministic Noise')
% raw_sensor,ifBias, ifSF, ifML,ifRandom
noise_sensor = addDeterministicErrors(noise_sensor,noiseSettings,1);
```

Calculate Headings of a Set of Paths

```
function angleToNorth = calculateAngleToNorth(est,groundTrueStates,time2align)
    Fs = 1/mean(diff(est.timestamps));
    sample2Align = round(Fs*time2align);

    angleToNorthEst = -atan2(est.pos(2,sample2Align), ...
        est.pos(1,sample2Align))*180/pi;
    angleToNorthGT = -atan2(groundTrueStates.pos(2,sample2Align), ...
        groundTrueStates.pos(1,sample2Align))*180/pi;
    angleToNorth = angleToNorthEst;
end
```

Computer Navigation Error With Reference

```
function error = computeNavErrorWRef(groundTrueStates,est)

error.pos = est.pos - groundTrueStates.pos;
error.orientation = est.euler_vect - groundTrueStates.euler_vect;
error.vel = est.vel - groundTrueStates.v_n;
error.timestamps = groundTrueStates.timestamps;
end
```

Find Time Offset Between Two Solutions

```
function deltaTau = findTimeOffSet(real_sensor,sim_sensor)
% compute initial time delay based on correlation
endTmp = min(length(real_sensor.f_b2i_b(1,:)),length(sim_sensor.f_b2i_b(1,:)));
    deltaTau = finddelay( vecnorm(real_sensor.f_b2i_b), ...
        vecnorm(sim_sensor.f_b2i_b));
end
```

Generate Ground Truth

```
function GT_aligned = groundTruthGeneration(GT_raw,fs_GT,fs)
% Description: interpolate ground truth position and orientation to desired
% sampling rate
%
% Input:
% - GT_raw: seven attributeTypes
%     - x:column vector, meter
%     - y:column vector, meter
%     - z:column vector, meter
%     - roll:column vector, radian
%     - pitch:column vector, radian
%     - yaw:column vector, radian
%     - name
% Output:
% - GT_aligned: seven attributeTypes
%     - x:column vector, meter
%     - y:column vector, meter
%     - z:column vector, meter
%     - roll:column vector, radian
%     - pitch:column vector, radian
%     - yaw:column vector, radian
%     - name

```

```
24
25 N = length(GT_raw); % number of objects
26
27 for kk = 1:N
28     % convert from mm to m
29     GT_aligned(kk).x = (GT_raw(kk).x-GT_raw(kk).x(1));
30     GT_aligned(kk).y = (GT_raw(kk).y-GT_raw(kk).y(1));
31     GT_aligned(kk).z = (GT_raw(kk).z-GT_raw(kk).z(1));
32     GT_aligned(kk).roll = GT_raw(kk).roll;
33     GT_aligned(kk).pitch = GT_raw(kk).pitch;
34     GT_aligned(kk).yaw = GT_raw(kk).yaw;
35     GT_aligned(kk).timestamps = (0:length(GT_aligned(kk).x)-1)/fs_GT;
36     GT_aligned(kk).name = GT_raw(kk).name;
37
38     interpl_method = 'spline'; % 'spline' 'linear'
39     GT_aligned(kk).x =
           interp1(GT_aligned(kk).timestamps(~isnan(GT_aligned(kk).x)),...
40         GT_aligned(kk).x(~isnan(GT_aligned(kk).x)),GT_aligned(kk).timestamps, ...
41         interpl_method,"extrap");
42     GT_aligned(kk).y =
           interp1(GT_aligned(kk).timestamps(~isnan(GT_aligned(kk).y)),...
43         GT_aligned(kk).y(~isnan(GT_aligned(kk).y)),GT_aligned(kk).timestamps, ...
44         interpl_method,"extrap");
45     GT_aligned(kk).z =
           interp1(GT_aligned(kk).timestamps(~isnan(GT_aligned(kk).z)),...
46         GT_aligned(kk).z(~isnan(GT_aligned(kk).z)),GT_aligned(kk).timestamps, ...
47         interpl_method,"extrap");
48     GT_aligned(kk).roll =
           interp1(GT_aligned(kk).timestamps(~isnan(GT_aligned(kk).roll)),...
49         GT_aligned(kk).roll(~isnan(GT_aligned(kk).roll)),GT_aligned(kk).timestamps,
               ...
50         interpl_method,"extrap");
51     GT_aligned(kk).pitch =
           interp1(GT_aligned(kk).timestamps(~isnan(GT_aligned(kk).pitch)),...
52         GT_aligned(kk).pitch(~isnan(GT_aligned(kk).pitch)),GT_aligned(kk).timestamps,
               ...
53         interpl_method,"extrap");
54     GT_aligned(kk).yaw =
           interp1(GT_aligned(kk).timestamps(~isnan(GT_aligned(kk).yaw)),...
55         GT_aligned(kk).yaw(~isnan(GT_aligned(kk).yaw)),GT_aligned(kk).timestamps,
               ...
56         interpl_method,"extrap");
57
58     %% generate smooth IMU data by signal processing
59     if 1
60         timestamps_interp1 =
               (0:(fs_GT/fs):length(GT_aligned(kk).timestamps)-1)/fs_GT;
```

```
        down_sample_window = 1;
        true_temp.x = GT_aligned(kk).x(1:down_sample_window:end);
        true_temp.y = GT_aligned(kk).y(1:down_sample_window:end);
        true_temp.z = GT_aligned(kk).z(1:down_sample_window:end);
        true_temp.roll = GT_aligned(kk).roll(1:down_sample_window:end);
        true_temp.pitch = GT_aligned(kk).pitch(1:down_sample_window:end);
        true_temp.yaw = GT_aligned(kk).yaw(1:down_sample_window:end);
        timestamps_raw = GT_aligned(kk).timestamps(1:down_sample_window:end);

        interpl_method = 'spline';

        GT_aligned(kk).x = interp1(timestamps_raw,true_temp.x,timestamps_interp1,
            ...
            interpl_method,"extrap");
        GT_aligned(kk).y = interp1(timestamps_raw,true_temp.y,timestamps_interp1,
            ...
            interpl_method,"extrap");
        GT_aligned(kk).z = interp1(timestamps_raw,true_temp.z,timestamps_interp1,
            ...
            interpl_method,"extrap");
        GT_aligned(kk).roll =
            interp1(timestamps_raw,true_temp.roll,timestamps_interp1, ...
            interpl_method,"extrap");
        GT_aligned(kk).pitch =
            interp1(timestamps_raw,true_temp.pitch,timestamps_interp1, ...
            interpl_method,"extrap");
        GT_aligned(kk).yaw =
            interp1(timestamps_raw,true_temp.yaw,timestamps_interp1, ...
            interpl_method,"extrap");
        GT_aligned(kk).timestamps = timestamps_interp1;
    end
end
end
```

Align Path to North

```
function est_North = rotateNavigation(est,rotAngle)
    earth_radius = 6378137; % earth radius
    rotMTX = [cos(rotAngle*pi/180) -sin(rotAngle*pi/180) 0;
              sin(rotAngle*pi/180) cos(rotAngle*pi/180) 0;
              0                    0                    1];
    est_North = est;

    % orientation
```

```
    for k = 1:length(est.euler_vect)
        C_b2n_tmp = rotMTX*eulr2dcmCSJ(est.euler_vect(:,k))';
        est_North.euler_vect(:,k) = dcm2eulrCSJ(C_b2n_tmp);
        est_North.q_b2n(:,k) = dcm2quaCSJ(C_b2n_tmp);
        est_North.C_b2n(:,:,k) = C_b2n_tmp;
    end

    est_North.dotv_n = rotMTX*est.dotv_n;
    est_North.vel = rotMTX*est.vel;
    est_North.pos = rotMTX*est.pos;

    est_North.LLA(1,:) = est_North.pos(1,:)/earth_radius + est.LLA(1,1);
    est_North.LLA(2,:) = est_North.pos(2,:)./cos(est_North.LLA(1,:))...
        /earth_radius + est.LLA(2,1);
end
```

B.3.7 Sensor Fusion Library

Downward-Facing Ultrasonic Sensor

```
function flags = elevator_detector(accel_z_array, g)
if sum((accel_z_array-g)>0) == length(accel_z_array)
    flags = 1;
elseif sum((accel_z_array-g)<0) == length(accel_z_array)
    flags = -1;
else
    flags = 0;
end

end
```

```
function elevator_ON = MovingElevatorDetection(accel_z)

elevator_flags = zeros(1,length(accel_z));
window_size = 1*700;
elevator_flags(1:window_size) = 0;
ini_z_axis_accel = mean(accel_z(1:100));
for kk = 1:length(accel_z)
    if kk >= window_size+1
        elevator_flags(kk-window_size:kk) =
            elevator_detector(accel_z(kk-window_size:kk),
            ini_z_axis_accel);%|elevator_flags(kk-window_size:kk);
```

```
    end
end

elevator_ON = zeros(1,length(elevator_flags));

elevator_on = 0;
current_state = 0;
for kk = 1:length(elevator_ON)
    elevator_ON(kk) = elevator_on;
    if elevator_on == 0 && current_state == 0 && elevator_flags(kk) == 1
        elevator_on = 1;
        elevator_ON(kk) = elevator_on;
    elseif elevator_on == 0 && current_state == 0 && elevator_flags(kk) == -1
        elevator_on = -1;
        elevator_ON(kk) = elevator_on;
    end

    if elevator_on == 1 && current_state == -1 && elevator_flags(kk) == 0
        elevator_on = 0;
        elevator_ON(kk) = elevator_on;
    elseif elevator_on == -1 && current_state == 1 && elevator_flags(kk) == 0
        elevator_on = 0;
        elevator_ON(kk) = elevator_on;
    end
    current_state = elevator_flags(kk);
end
```

Barometer

```
function [P_cal,b_cal] = calibratePressure(P,P_ref,t,t_ref)
P_align = interp1(t(~isnan(P)),P(~isnan(P)),t_ref);
fit_ind = ~isnan(P_align)&~isnan(P_ref);
b_cal = [ones(length(P_align(fit_ind)),1) P_align(fit_ind)']\P_ref(fit_ind)';
P_cal = P*b_cal(2) + b_cal(1);
end
```

```
function P = height2pressure(h,T,P0)
P = exp(-h*(0.02896*9.807)./(273.15+T)/8.3143)*P0;
end
```

```
function h = pressure2height(P,T,P0)
h = -log((P)/P0)*8.3143.*(273.15+T)/(0.02896*9.807);
end
```

Adaptive Covariance

```
function [zupt,total,total_x,total_y,total_z,zupt_variance] =
    ZUPTDynamicCovariance(u)
% -------------------------------------------------------------------------
% This function calculated adaptive coravirance for ZUPT measurements
% -------------------------------------------------------------------------
global simdata;

total_x = 1;
total_y = 1;
total_z = 1;

[r, c] = size(u);
W=simdata.Window_size;
sigma2_a = (simdata.sigma_a/simdata.Ts)^2;
sigma2_g = (simdata.sigma_g/simdata.Ts)^2;
g = 9.817269086191379;
u_n = mean(u(1:3, :), 2);
u_n = u_n / norm(u_n); % Unit vector along the specific force
for i = 1:3
    u(i, :) = u(i, :) - g*u_n(i);
end

total_x = sum(sum(u(1,:).^2))/sigma2_a+sum(sum(u(4,:).^2))/sigma2_g;
total_y = sum(sum(u(2,:).^2))/sigma2_a+sum(sum(u(5,:).^2))/sigma2_g;
total_z = sum(sum(u(3,:).^2))/sigma2_a+sum(sum(u(6,:).^2))/sigma2_g;

total = sum(sum(u(1:3,:).^2))/sigma2_a+sum(sum(u(4:6,:).^2))/sigma2_g;
total = total/c;

if(total < simdata.factor)
    zupt = ones(1, simdata.Window_size_for_dynamic_covariance);
else
    zupt = zeros(1, simdata.Window_size_for_dynamic_covariance);
end

zupt_variance =
    (([total,total,total]).*simdata.dynamic_variance_hyper_para_alpha)...
    .^simdata.dynamic_variance_hyper_para_gamma...
    .*simdata.dynamic_variance_hyper_para_beta*1;
end
```

B.3.8 Temperature Compensation Library

B.3.9 Neural Network Library

```
function [u,tem_sigma] = batchTemperautreCompensation(u,type,ifUse)
if ifUse
    if 1
    disp('*Offline temperature compensation using Temperature and differential
        temperature*')
        ind = 1:1:length(u(15,:));
        temp_delta = diff(movmean(u(15,ind),1000));
        temp = movmean(u(15,ind),1000);
        temp = temp(1:end-1);
        rel_temp = temp-u(15,1);
        u = u(:,1:end-1);
        feature = [temp;temp_delta];

        [temp_bias,tem_sigma] = predTempError(feature);

        if type == "dbias"
            temp_bias.a_x = cumsum(temp_bias.a_x);
            temp_bias.a_y = cumsum(temp_bias.a_y);
            temp_bias.a_z = cumsum(temp_bias.a_z);
            temp_bias.g_x = cumsum(temp_bias.g_x);
            temp_bias.g_y = cumsum(temp_bias.g_y);
            temp_bias.g_z = cumsum(temp_bias.g_z);
        end
        % remove bias from IMU
        u(1,:) = u(1,:) - temp_bias.a_x;
        u(2,:) = u(2,:) - temp_bias.a_y;
        u(3,:) = u(3,:) - temp_bias.a_z;
        u(4,:) = u(4,:) - temp_bias.g_x;
        u(5,:) = u(5,:) - temp_bias.g_y;
        u(6,:) = u(6,:) - temp_bias.g_z;

        tem_sigma.a_x = tem_sigma.a_x/tem_sigma.a_x(1);
        tem_sigma.a_y = tem_sigma.a_y/tem_sigma.a_y(1);
        tem_sigma.a_z = tem_sigma.a_z/tem_sigma.a_z(1);
        tem_sigma.g_x = tem_sigma.g_x/tem_sigma.g_x(1);
        tem_sigma.g_y = tem_sigma.g_y/tem_sigma.g_y(1);
        tem_sigma.g_z = tem_sigma.g_z/tem_sigma.g_z(1);
    else
    disp('*Offline temperature compensation using Temperature*')
        rel_temp = u(15,:)-u(15,1);
        feature = rel_temp;

```

```
        accel_x_bias = myNeuralNetworkFunctionForAccelX_bias(feature);
        accel_y_bias = myNeuralNetworkFunctionForAccelY_bias(feature);
        accel_z_bias = myNeuralNetworkFunctionForAccelZ_bias(feature);
        gyro_x_bias = myNeuralNetworkFunctionForGyroX_bias(feature);
        gyro_y_bias = myNeuralNetworkFunctionForGyroY_bias(feature);
        gyro_z_bias = myNeuralNetworkFunctionForGyroZ_bias(feature);

        u(1,:) = u(1,:) - accel_x_bias;
        u(2,:) = u(2,:) - accel_y_bias;
        u(3,:) = u(3,:) - accel_z_bias;
        u(4,:) = u(4,:) - gyro_x_bias;
        u(5,:) = u(5,:) - gyro_y_bias;
        u(6,:) = u(6,:) - gyro_z_bias;

        % predict noise level
        tem_sigma.a_x = myNeuralNetworkFunctionForAccelX_SD(feature);
        tem_sigma.a_y = myNeuralNetworkFunctionForAccelY_SD(feature);
        tem_sigma.a_z = myNeuralNetworkFunctionForAccelZ_SD(feature);
        tem_sigma.g_x = myNeuralNetworkFunctionForGyroX_SD(feature);
        tem_sigma.g_y = myNeuralNetworkFunctionForGyroY_SD(feature);
        tem_sigma.g_z = myNeuralNetworkFunctionForGyroZ_SD(feature);

        tem_sigma.a_x = tem_sigma.a_x/tem_sigma.a_x(1);
        tem_sigma.a_y = tem_sigma.a_y/tem_sigma.a_y(1);
        tem_sigma.a_z = tem_sigma.a_z/tem_sigma.a_z(1);
        tem_sigma.g_x = tem_sigma.g_x/tem_sigma.g_x(1);
        tem_sigma.g_y = tem_sigma.g_y/tem_sigma.g_y(1);
        tem_sigma.g_z = tem_sigma.g_z/tem_sigma.g_z(1);
    end
else
    disp('No temperature compensation')
    tem_sigma.a_x = 0*u(1,:) + 1;
    tem_sigma.a_y = 0*u(2,:) + 1;
    tem_sigma.a_z = 0*u(3,:) + 1;
    tem_sigma.g_x = 0*u(4,:) + 1;
    tem_sigma.g_y = 0*u(5,:) + 1;
    tem_sigma.g_z = 0*u(6,:) + 1;
end
```

```

function performance = train_ANN(x,t,node_num,sel_axis,mode)

% Solve an Input-Output Fitting problem with a Neural Network
% Script generated by Neural Fitting app
% Created 13-Oct-2021 23:47:17
%
% This script assumes these variables are defined:
```

```
9
10 % Choose a Training Function
11 % For a list of all training functions type: help nntrain
12 % 'trainlm' is usually fastest.
13 % 'trainbr' takes longer but may be better for challenging problems.
14 % 'trainscg' uses less memory. Suitable in low memory situations.
15 trainFcn = 'trainlm'; % Levenberg-Marquardt backpropagation.
16
17 % Create a Fitting Network
18 hiddenLayerSize = node_num*ones(1,1);
19
20 % net = cascadeforwardnet(hiddenLayerSize,trainFcn);
21 net = fitnet(hiddenLayerSize,trainFcn);
22
23 % Set training stopping condition
24 net.trainParam.goal = 0;
25 net.trainParam.min_grad = 1.0000e-11;
26 net.trainParam.max_fail = 20;
27 net.trainParam.time = Inf;
28 % net.trainParam.mu = 10e-9;
29 net.trainParam.epochs = 1000;
30
31 % Choose Input and Output Pre/Post-Processing Functions
32 % For a list of all processing functions type: help nnprocess
33 net.input.processFcns = {'removeconstantrows','mapminmax'};
34 net.output.processFcns = {'removeconstantrows','mapminmax'};
35
36 % Setup Division of Data for Training, Validation, Testing
37 % For a list of all data division functions type: help nndivision
38 net.divideFcn = 'dividerand'; % Divide data randomly
39 net.divideMode = 'sample'; % Divide up every sample
40 net.divideParam.trainRatio = 70/100;
41 net.divideParam.valRatio = 15/100;
42 net.divideParam.testRatio = 15/100;
43
44 % Choose a Performance Function
45 % For a list of all performance functions type: help nnperformance
46 net.performFcn = 'mse'; % Mean Squared Error
47
48 % Choose Plot Functions
49 % For a list of all plot functions type: help nnplot
50 net.plotFcns = {'plotperform','plottrainstate','ploterrhist', ...
51     'plotregression', 'plotfit'};
52
53 % Train the Network
54 [net,tr] = train(net,x,t,'useParallel','yes','showResources','no','useGPU','no');
55
```

```
% Test the Network
y = net(x);
e = gsubtract(t,y);
performance = perform(net,t,y);

% Recalculate Training, Validation and Test Performance
trainTargets = t .* tr.trainMask{1};
valTargets = t .* tr.valMask{1};
testTargets = t .* tr.testMask{1};
trainPerformance = perform(net,trainTargets,y);
valPerformance = perform(net,valTargets,y);
testPerformance = perform(net,testTargets,y);

% View the Network
% view(net)

% Plots
% Uncomment these lines to enable various plots.
%figure, plotperform(tr)
%figure, plottrainstate(tr)
%figure, ploterrhist(e)
%figure, plotregression(t,y)
%figure, plotfit(net,x,t)

% Deployment
% Change the (false) values to (true) to enable the following code blocks.
% See the help for each generation function for more information.
if mode == "big"
    % Generate MATLAB function for neural network for application
    % deployment in MATLAB scripts or with MATLAB Compiler and Builder
    % tools, or simply to examine the calculations your trained neural
    % network performs.
    genFunction(net,'TempCompNet','MatrixOnly','yes');
    y = TempCompNet(x);
end
if mode == "medium"
    % Generate MATLAB function for neural network for application
    % deployment in MATLAB scripts or with MATLAB Compiler and Builder
    % tools, or simply to examine the calculations your trained neural
    % network performs.
    if sel_axis == 1
        genFunction(net,'TempCompNet_bias','MatrixOnly','yes');
        y = TempCompNet_bias(x);
    elseif sel_axis == 2
        genFunction(net,'TempCompNet_SD','MatrixOnly','yes');
        y = TempCompNet_SD(x);
    end
```

```
end
if mode == "small"
    % Generate a matrix-only MATLAB function for neural network code
    % generation with MATLAB Coder tools.
    if sel_axis == 1
        genFunction(net,'TempCompNet_accl_bias_x','MatrixOnly','yes');
    elseif sel_axis == 2
        genFunction(net,'TempCompNet_accl_bias_y','MatrixOnly','yes');
    elseif sel_axis == 3
        genFunction(net,'TempCompNet_accl_bias_z','MatrixOnly','yes');
    elseif sel_axis == 4
        genFunction(net,'TempCompNet_gyro_bias_x','MatrixOnly','yes');
    elseif sel_axis == 5
        genFunction(net,'TempCompNet_gyro_bias_y','MatrixOnly','yes');
    elseif sel_axis == 6
        genFunction(net,'TempCompNet_gyro_bias_z','MatrixOnly','yes');
    elseif sel_axis == 7
        genFunction(net,'TempCompNet_accl_SD_x','MatrixOnly','yes');
    elseif sel_axis == 8
        genFunction(net,'TempCompNet_accl_SD_y','MatrixOnly','yes');
    elseif sel_axis == 9
        genFunction(net,'TempCompNet_accl_SD_z','MatrixOnly','yes');
    elseif sel_axis == 10
        genFunction(net,'TempCompNet_gyro_SD_x','MatrixOnly','yes');
    elseif sel_axis == 11
        genFunction(net,'TempCompNet_gyro_SD_y','MatrixOnly','yes');
    elseif sel_axis == 12
        genFunction(net,'TempCompNet_gyro_SD_z','MatrixOnly','yes');
    end
end
if (false)
    % Generate a Simulink diagram for simulation or deployment with.
    % Simulink Coder tools.
    gensim(net);
end
end
```

```
% Solve an Input-Output Time-Series Problem with a Time Delay Neural Network
% Script generated by Neural Time Series app.
% Created 07-Oct-2021 21:28:45
%
% This script assumes these variables are defined:
%
%   temp - input time series.
%   smooth_error - target time series.

X = tonndata(temp,true,false);
```

```
T = tonndata(smooth_error,true,false);

% Choose a Training Function
% For a list of all training functions type: help nntrain
% 'trainlm' is usually fastest.
% 'trainbr' takes longer but may be better for challenging problems.
% 'trainscg' uses less memory. Suitable in low memory situations.
trainFcn = 'trainlm'; % Levenberg-Marquardt backpropagation.

% Create a Time Delay Network
inputDelays = 1:1;
hiddenLayerSize = 10;
net = timedelaynet(inputDelays,hiddenLayerSize,trainFcn);

% Prepare the Data for Training and Simulation
% The function PREPARETS prepares timeseries data for a particular network,
% shifting time by the minimum amount to fill input states and layer
% states. Using PREPARETS allows you to keep your original time series data
% unchanged, while easily customizing it for networks with differing
% numbers of delays, with open loop or closed loop feedback modes.
[x,xi,ai,t] = preparets(net,X,T);

% Setup Division of Data for Training, Validation, Testing
net.divideParam.trainRatio = 70/100;
net.divideParam.valRatio = 15/100;
net.divideParam.testRatio = 15/100;

% Train the Network
[net,tr] = train(net,x,t,xi,ai);

% Test the Network
y = net(x,xi,ai);
e = gsubtract(t,y);
performance = perform(net,t,y)

% View the Network
view(net)

% Plots
% Uncomment these lines to enable various plots.
%figure, plotperform(tr)
%figure, plottrainstate(tr)
%figure, ploterrhist(e)
%figure, plotregression(t,y)
%figure, plotresponse(t,y)
%figure, ploterrcorr(e)
%figure, plotinerrcorr(x,e)
```

```
58
59 % Step-Ahead Prediction Network
60 % For some applications it helps to get the prediction a timestep early.
61 % The original network returns predicted y(t+1) at the same time it is
62 % given x(t+1). For some applications such as decision making, it would
63 % help to have predicted y(t+1) once x(t) is available, but before the
64 % actual y(t+1) occurs. The network can be made to return its output a
65 % timestep early by removing one delay so that its minimal tap delay is now
66 % 0 instead of 1. The new network returns the same outputs as the original
67 % network, but outputs are shifted left one timestep.
68 nets = removedelay(net);
69 nets.name = [net.name ' - Predict One Step Ahead'];
70 view(nets)
71 [xs,xis,ais,ts] = preparets(nets,X,T);
72 ys = nets(xs,xis,ais);
73 stepAheadPerformance = perform(nets,ts,ys)
```

Appendix C

List of Vendors

Adobe Inc.

Address: 345 Park Avenue San Jose, CA 95110−2704

Phone: (408) 537−6000

Email: adobepr@adobe.com

Website: https://www.adobe.com/

Videos demonstrating navigation solutions were edited in Adobe software solution Premiere Pro.

Analog Devices

Address: One Analog Way Wilmington, MA 01887

Phone: (781) 935−5565

Website: https://www.analog.com/

IMUs integrated into the Lab-On-Shoe platforms are available from this vendor.

Arduino

Address: 25 Via Andrea Appiani Monza, Monza, Lombardia, 20900, Italy

Phone: +39 1119116387

Website: https://www.arduino.cc/

Microcontrollers Arduino Uno, Uno WiFi Rev2, Due, Yun Rev 2, and Nano Every used in the Lab-On-Shoe and Sugar-Cube platforms are available from Arduino.

AXON Cable Inc.

Address: 1314 N. Plum Grove Rd Schaumburg IL 60173−4546

Phone: (847) 230−7800

Email: sales@axon-cable.com

Website: https://www.axon-cable.com/

Industrial cables manufactured by AXON Cable were selected to connect shoe PCBs and mother PCB on the Lab-On-Shoe 1 platform.

Boot Barn

Address: 23762 Mercury Rd Ste B, Lake Forest, CA 92630

Phone: (949) 455−0211

Website: https://www.bootbarn.com

Steel-toe boots for the Lab-On-Shoe platform are available from Boot Barn.

Dell Inc.

Address: 1 Dell Way Round Rock, TX 78682

Phone: (800) 289−3355

Website: https://www.dell.com/en-us

Computers, XPS Tower Special Edition, laptops, Latitude 7420, and monitors P2419H, used to analyze, implemented, design, and demonstrate the work presented in this thesis were purchased from Dell.

DFRobot

Address: Rm 602, Bld. Puruan, No. 2 Boyun Rd, Pudong, Shanghai(201203), China

Phone: +86 (21)−61620183

Email: store@dfrobot.com

Website: https://www.dfrobot.com/

Devantech Ultrasonic sensors SRF02 integrated into the Lab-On-Shoe platform are available from this vendor.

Digi-Key Corp

Address: One Technology Way PO BOX 9106, Norwood, MA 02062 Phone: (800) 344−4539

Website: http://www.digikey.com/

Retails electronic components and supplies various development boards.

Electromaker

Location: Poole, Dorset

Email: hello@electromaker.io

Website: https://www.electromaker.io/

Retails electronic components and supplies various development boards.

Garmin Ltd.

Address: 1200 E 151st St, Olathe, KS

Website: https://www.garmin.com/en-US/

Laser range finders Garmin Lite v3 are available from this vendor.

Ideal Aerosmith, Inc.

Address: 2205 W Lone Cactus Dr Suite 7, Phoenix, AZ 85027 Phone: (701) 757−5601

Website: http://www.ideal-aerosmith.com/

Precision two-axis and three-axis positioning and rate table systems for inertial sensor characterization are available from this vendor.

JLCPCB

Address: 2788 San Tomas Expy, Santa Clara, CA 95051

Phone: (800) 262−5643

Website: https://jlcpcb.com/

Email: support@jlcpcb.com

Customized PCBs used on the Lab-On-Shoe and Sugar-Cube platforms were fabricated by JLCPCB.

McMASTER-CARR

Address: 2788 San Tomas Expy, Santa Clara, CA 95051

Phone: (562) 692−5911

Website: https://www.mcmaster.com/

Email: sales@mcmaster.com

Supplier of miscellaneous hardware components used for prototype hardware developed in this thesis.

NVIDIA

Address: 2788 San Tomas Expy, Santa Clara, CA 95051

Phone: (408) 486−2000

Website: https://www.nvidia.com/

Email: info@nvidia.com

NVIDIA Jetson Nano processor and its developer kit used to implement real-time computer vision algorithm for the Lab-On-Shoe 1 platform were purchased from NVIDIA.

Meta Platforms, Inc.

Address: One Hacker Way Menlo Park, CA 94025

Phone: (650) 543−4800

Website: https://about.meta.com/

Oculus Quest 2 was purchased from this vendor.

Polhemus

Address: 40 Hercules Drive Colchester, Vermont 05446

Phone: (800) 357−4777

Email: sales@polhemus.com

Website: https://polhemus.com/

Polhemus specializes in magnetic motion tracking technology. PATRIOT 6DOF motion tracker available from this vendor was used in the characterization of the motion of feet.

RapidTech

Address: 444 Engineering Tower, 4242 Campus Dr, Irvine, CA 92697

Phone: (949) 824−4105

Website: https://manufacturing.uci.edu/index.php/rapidtech/

Email: dolanb@uci.edu (Benjamin Dolan, Technical Director)

Camera fixtures used to mount cameras on the Lab-On-Shoe 1 platform were fabricated by RapidTech.

Robotshop Inc.

Address: 18005 Rue Lapointe, Mirabel, Quebec, J7J 0G2, Canada

Phone: (450) 420−1446

Website: https://www.robotshop.com/

Arducam 8MP Sony IMX219 camera module w/ CS lens 2718, compatible with Raspberry Pi microcontroller and NVIDIA Jetson Nano, is available from this vendor.

Sensory Lab

Address: 707 S. Grand Ave Bozeman MT 59715

Phone: (406) 585−5925

Email: christine@sensorylabs.com (Christine Raymond)

Website: https://sensorylabs.com/

Basler cameras, acA800−200gc, and lenses with 4mm, 2.5mm and 6mm integrated into the Lab-On-Shoe 1 platform were purchased from Sensory Lab.

SparkFun Electronics

Address: 6333 Dry Creek Parkway, Niwot, CO 80503

Phone: (303) 284−0979

Website: https://www.sparkfun.com/

Electronic breakout boards, jumper wires, lithium-ion batteries, voltage regulators, barometers, IMU, UWB, and microcontrollers, used on the Lab-On-Shoe platforms and Sugar-Cube platforms were purchased from SparkFun Electronics.

Stereolabs

Address: 6333 Dry Creek Parkway, Niwot, CO 80503

Email: support@stereolabs.com

Website: https://www.stereolabs.com/

ZED 2 Stereo Camera and ZED Mini were purchased from this vendor to investigate vision SLAM algorithms.

TDK InvenSense, Inc.

Address: 1745 Technology Dr #200, San Jose, CA 95110

Phone: (408) 988−7339

Website: https://invensense.tdk.com/

COTS IMUs ICM−20948, ICM−20649, and ICM−42605 integrated into the Sugar-Cube platforms are available from this vendor.

VectorNav Technologies

Address: 10501 Markison Rd, Dallas, TX 75238

Phone: (512) 772−3615

Website: https://www.vectornav.com/

VectorNav VN−200 IMUs are available from this vendor.