UNIVERSITY OF CALIFORNIA
SANTA CRUZ

GUIDANCE, NAVIGATION, AND CONTROL OF AUTONOMOUS SURFACE VEHICLES FOR OPTIMAL EXPLORATION AND LOW-COST OCEANOGRAPHY

A dissertation submitted in partial satisfaction of the
requirements for the degree of

DOCTOR OF PHILOSOPHY

in

COMPUTER ENGINEERING

by

Pavlo Vlastos

June 2022

The Dissertation of Pavlo Vlastos
is approved:

Gabriel Elkaim, Chair

Ricardo Sanfelice

Renwick Curry

Peter Biehl
Vice Provost and Dean of Graduate Studies

Table of Contents

List of Figures

List of Tables

Abstract

Guidance, Navigation, and Control of Autonomous Surface Vehicles for Optimal Exploration and Low-Cost Oceanography

by

Pavlo Vlastos

Autonomous surface vehicles (ASVs) may be one of the most promising tools to help study, protect, and address problems within complex ocean ecosystems. ASVs can be used to collect ocean data for climate modeling, collect plastics and other human-related pollution, study ocean fronts, and map the ocean floor. These types of vehicles possess sensors that can be used with spatial estimation techniques, such as simultaneous localization and mapping (SLAM), ordinary kriging, or Gaussian process regression (GPR). Such algorithms produce an estimate of the field representing the local environment with an associated level of uncertainty. As the ASV collects more measurements, the estimate is updated. However, it is common for ASVs to follow static paths with linear waypoint-to-waypoint tracking for navigating a field of interest; the path doesn't change as more information of the field is obtained. An alternative strategy is dynamic path planning based on uncertainty suppression. In this thesis an optimal exploration algorithm is presented that maximizes the current field-estimate variance along a path. This algorithm is based on Bellman-Ford graph search and is compared to a myopic, zigzag, and other planners in simulation. These path planners are also paired with different spatial estimation methods, including partitioned ordinary kriging (POK) and partitioned GPR. The results are discussed including the computation time trade-offs. The intent is to determine an efficient path planner and spatial estimator that can be used at run-time onboard an ASV.

Two experimentally implemented ASVs, the Slug 2 and Slug 3 are discussed. This includes discussion of their system design, attitude and heading reference system, system modeling, system identification, control, and experimental results. Both ASVs used GPS for waypoint tracking and data logging. The Slug 3 used an echo sounder depth sensor to measure the depth of a small body of water. For navigation the ASVs used a GPS module and an IMU with tri-axis MEMS accelerometers, gyroscopes, and magnetometers.

To my brother, my Babá, and especially my Mamá. I think she would have really enjoyed this.

Acknowledgments

I would like to thank Professor Gabriel Elkaim for offering me the opportunity to conduct research in the Autonomous Systems Laboratory, going all the way back to my first lab meeting during my freshman year as an undergraduate student. I have yet to find a more challenging, but rewarding work. Gabe convinced me to petition to become a PhD student during my masters, and I'm confident that was an excellent choice to make. He has given me countless advice and ideas. I would like to thank Professor Renwick Curry for making me think more deeply about many of the engineering challenges associated with this research, boiling it down to the fundamentals, and working my way up. Without Ren's key insights, this dissertation and subsequent work would not have been possible. Ren always asked the best questions to further this work and many if not all of his ideas were incredibly helpful and insightful. I think Ren provided the most feedback on all of my writing out of anyone in my life thus far. I must also thank Professor Ricardo Sanfelice for being on my advancement, defense, and reading committees. He provided excellent points of feedback and ideas to help improve the quality of this research. I would also like to thank Max Dunne for his patience in helping me set up the hardware drivers for many of the peripheral devices for this work.

My thanks also goes out to all of the members and my friends in the Autonomous System's Laboratory, including Bryant Mairs, Sharon Rabinovitch, Jordan Liss, Sargis Yonan, Aaron Hunter, Carlos Espinosa, Joseph Adamson, Chris Seruge, Max Lichtenstein, Majid Moghadam, Engin Tekin, James Melvin, Eliana Stefani, and the rest of the ASL members I have had the fortune of knowing and solving problems with.

Additionally, I must thank my friends who I haven't already listed; Conrad Esch, Aviv Elor, Sam Conde, Kyle Cordes, Sam LeBlanc, Eric Jung, Vic-

toria Ly, Ash Robbins, Maryam Tebyani, Michael Suinn, Devesh Gohkalgandhi, Eric Burgstrom, Mike Powell, Siera Catelani, Anca Mihaela, Zach Potter, Megan Boivin, and so many more people that I haven't the space to list here, but were important nonetheless.

Of course, I would also like to thank the readers of this paper and the members of the committee. Your guidance and consideration have been essential.

I absolutely must thank all of the Vlastos family for being ever so patient and understanding. I know I have missed far too many family gatherings while studying out here in California, and y'all are waiting for my excellent cooking. I hope this body of work can help shed light on what in the world took so long. If I'm not mistaken, this is the first computer engineering PhD dissertation from a Vlastos in our family's, dare I say *long* history. Thank you to my brother Lucas, and my cousins Elias, Nikiphoros, Niko, Zoe, Xander, Kali, Kleo, and Mazzi. Thank you to Keith, Kelly, Lisa, George, Manoli, Carrie, Theo Joe, Carol, and of course my Babá. Thank you to Lucas especially for taking care of our Babá (it's never too late to start working on one's health before it works on you) while I was away and listening to my rants on random math stuff and rocket launches. Thank you to my Thea Judy and Theo Steve for visiting, giving excellent financial advice, and inviting me to some of the most memorable Thanksgiving meals before finals.

And finally, and most importantly I would like to thank my Mamá, who inspired me to push myself to work really, really hard. I remember thinking of her throughout the night that I wrote my essays for my application to UC Santa Cruz. She will always be missed, but the memory of her has stayed with me like a fire for knowledge. I suspect that she would have loved to read through this and most likely found more than a dozen typos and grammar edits. In fact, she probably would have thought of some feedback controller designs herself after reading this.

She is missed, but never forgotten.

Chapter 1

Introduction

This thesis describes the Slug Autonomous Surface Vehicle Project. The goal of which, at a high level, was to develop a robust autonomous vehicle to drastically reduce the cost of oceanography. This goal evolved into designing a system capable of autonomous waypoint tracking and optimal exploration. A systems engineering approach was taken to address the framework and design of what could eventually be a high volume and low-cost autonomous surface vehicle able to study and estimate various phenomena in marine environments. This dissertation outlines the designs and implementations of two autonomous surface vehicles (ASV) for this purpose. Path-planning, guidance, navigation, and control are given special attention. This includes control algorithms, their formulation, analysis, and experimental validation. Emphasis is placed on the importance of creating robust ASVs for oceanography and their part in reducing the cost of ocean data acquisition.

1.1 The Overarching Theme: Automated Field Exploration

The overall theme is a vision of the oceanographer using an ASV to collect valuable ocean data; the idea is that a researcher could deploy the ASV and it would autonomously survey the field in an efficient manner, generating a high quality estimate (with bounds) in minimum time. An example use-case is that of an oceanographer attempting to find the concentrations in an area of the ocean of micro-plastics, or other similar pollutants. Ideally, the ASV will travel in a way that quickly provides an accurate estimate of the area or field. The ASV does not explicitly locate the highest concentrations, but rather travels in a way to minimize the uncertainty associated with the field estimate at a given point in time. In this way, the researcher can conduct further studies at specific locations of interest based on their domain-specific skills and objectives.

Note that the oceanographer should need no domain knowledge of robotics, controls, navigation, or autonomy to use the ASV. Rather, they would deploy the ASV and use high level commands on a convenient graphic user interface. The ASV would use the control systems, path planning algorithms, spatial estimator, and attitude and heading reference system that are discussed in this thesis. The idea of generating a quick and accurate field estimate using the ASV is to reduce the time that the researcher spends trying to find domain-specific areas of interest. This idea is revisited throughout this thesis in each chapter to highlight how each subsystem or algorithm contributes to the overarching theme.

1.2 Motivation

Oceanography, the study of physical and biological phenomena of the ocean, is expensive; it can cost many thousands of dollars a day to keep a ship and crew on task or standing by for oceanographic study. Each crewed mission risks the humans onboard, damage to the vessel, and possess intrinsic limitations on mission duration and scheduling. These limitations prevent studying the ocean, especially transient ocean phenomenon such as algal blooms, oil-spills, and waste spills. Aside from transient phenomenon, the high cost of oceanographic limitations effect the study of longer-term problems and solutions to address those problems. For instance, billions of dollars are lost annually due to continuing increases in ocean acidification [56]. Ocean acidification, in particular, may have far reaching effects on many marine organisms, such as phytoplanktonic coccolithophores, pteropods, and other mollusks, [23], [81]. Organisms such as phytoplankton help fix carbon at a rate of up to 50 Gigatons each year [27]. Single-celled and chain-forming diatoms are expected to have declined by 40% since 1950 [54]. That is obviously a concerning metric and further warrants the development of low-cost oceanographic research solutions. The cost of oceanography and the ability to address these issues must be reduced. Another limitation is the time taken to transit from port to the desired research location - time during which the vessel is burning fuel, crew are being fed and housed, and time which the vessel is unavailable for other missions.

Ideally, measurements of the ocean could be made at any time and place at low cost. To that extent another method to study the ocean is to use satellites. Unlike marine vessels satellites are able to view comparably large areas of the Earth in a short amount of time. Satellites, such as the Sea-Viewing Wide Field-of-View Sensor (SeaWiFS) launched specifically to measure chlorophyll of plankton, [59]. At 10% the cost of comparable satellites, the 5-year mission cost 42 million dol-

lars [6]. To help calibrate SeaWiFS, in-situ surface measurements were taken for comparison by the Marine Optical Buoy (MOBY) [35]. This is an instance where both marine surface vessels and satellites may be used together for oceanography. However, some ocean phenomenon may exclusively require in-situ measurements that cannot be measured by satellites.

Autonomous surface vehicles (ASV) can help reduce some of the costs. There are many existing ASV projects for ocean data collection; some are currently commercially available to researchers. There exist short to long-range mission ASV types with various sensors (depending on specific applications and types of research missions). While many ASVs have some level of flexibility in their mission capabilities many do not consider advanced control techniques, or autonomous and intelligent exploration algorithms. This thesis specifically focuses on modeling, control, and exploration algorithms to improve the science return, power usage, and overall system utility. Many ASV designs have underlying hardware and software that are difficult to customize. This can be challenging to researchers that require a versatile ASV for different kinds of missions.

1.3 Background

We review a number of similar projects and research efforts to solve the general problem of collecting oceanographic data using autonomous surface vehicles or remote operable submersible vehicles. These different projects are compared and their advantages and disadvantages to solving the energy constrained exploration and navigation (ECEN) problem is discussed. The general tools for solving the ECEN problem, such as attitude estimation, trajectory tracking, and path planning are subsequently discussed.

1.3.1 Similar Projects and Research

Previous efforts to reduce the cost of, or improve, oceanography using autonomous surface vehicles include notable contributions by [51], [82], [42], [73], [4], [29], and [25]. In 2003, the Monterey Bay Aquarium Research Institute (MBARI), in collaboration with the National Oceanic and Atmospheric Administration Pacific Marine Environmental Laboratory (NOAA/PMEL) developed a drifting buoy that used a state-of-the-art sensor suite called the MAPCO$_2$ with an infra-red analyzer to measure CO_2 concentration of coastal ocean waters. Its design possessed no means of propulsion or navigation, but was able to reliably collect ocean data. There are currently (June 2022) 19 in the Pacific and Atlantic ocean, [82].

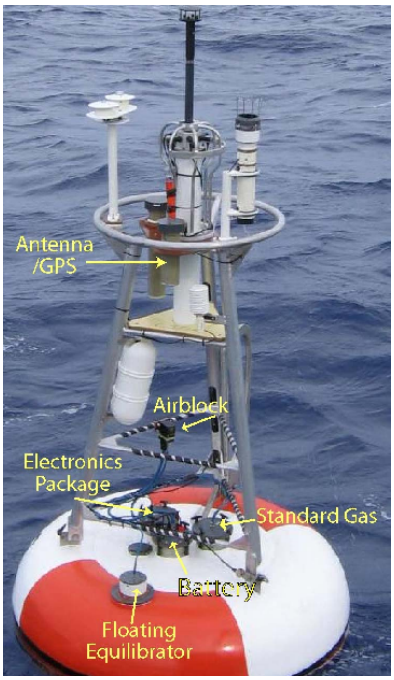


Figure 1.1: A picture the MAPCO$_2$ system by [82], installed on a buoy.

In contrast to limited mobility of the MAPCO$_2$ buoy, the Wave Glider by the Liquid Robotics company uses a similar sensor suite based on the MAPCO$_2$ system. To increase mission duration, it scavenges energy from ocean-waves, using

a submerged glider that is pulled by waves to propel the vehicle, [42]. One of the drawbacks to this system is characterizing control scheme and waypoint tracking along with estimating the maximum mission duration for such a vehicle. However, energy scavenging does extend the overall operating time.

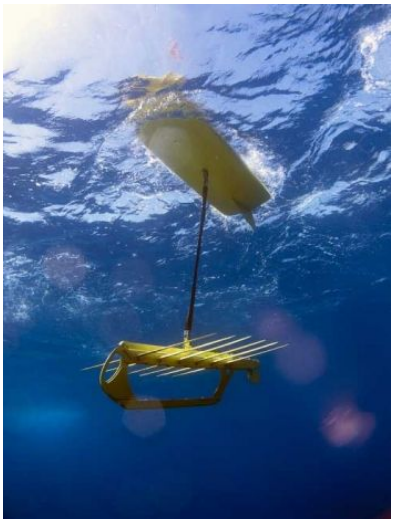


Figure 1.2: A picture the Wave Glider built by Liquid Robotics.

The most similar project to this thesis is the Sea Slug [51], because this work is in some respects a continuation of that work, albeit with different emphases. An autonomous surface vehicle approximately 6.5 meter long was outfitted with a controller area network (CAN) of small microcontrollers (CANodes). Each CANode controlled a specific peripheral; the CAN bus could be easily modified with more nodes depending on the mission. This made it a versatile tool in the hands of an oceanographer. It was capable of autonomous GPS navigation, but did not posses obstacle avoidance capabilities.

In [29] and [4], catamaran-type boats called ROAZ and ROAZ II were used to map coastal features, including the Tua River in three-dimensions using an echo sounder and a camera. It was also capable of acting as a central communications node in multi-vehicle missions. ROAZ II used an RTK GPS for position measure-

Figure 1.3: A picture the Sea Slug from the University of California at Santa Cruz's Autonomous Systems Laboratory [51].

ments, possessed image processing capabilities, an internal world map, a mission management system, and a motion controller. Like [51], it also used a CAN bus, but relied on an embedded system with a Linux operating system instead of a set of microcontrollers. This is a comparatively expensive vehicle with a very specific purpose, and is not able to be re-tasked for a variety of missions. The ROAZ II is capable of radar based collision detection.

A 16-foot long solar powered catamaran ASV is presented in [25]. It was capable of obstacle avoidance while traveling under 1 m/s, with an obstacle detection range of about 30 meters. It used a laser scanner to detect obstacles. Similar to [4], it placed a 15 meter boundary around the obstacle when generating its avoidance trajectory. It has sonars, cameras, GPS, and uses a small computer to interface with these peripheral devices.

One of the more notable takeaways from these projects, with the exception of [51], is a lack of emphasis regarding their guidance systems. Both [4] and [25] do not discuss vehicle modeling, control schemes, or waypoint switching/interpolation. Studying these aspects of autonomous vehicles can improve their utility, safety, and cost over less guided platforms. Fortunately, others have developed detailed, robust ship models, viable controllers, and waypoint maneuvering methods for

ASVs (see [76], [62], [65]).

In [76] the dynamics of ship steering are explored and robust continuous first and second-order models are developed. Details such as controllability and observability of the models are discussed. A Newtonian approach for ship modeling is discussed in [62], which considers everything from hydrodynamic forces that can create moments about a ship's rudder, to modeling the propulsion system and the associated hydrodynamic resistance.

If there is a single, best ASV design in existence now, or one that can be considered the most advanced, then that honor should go to the Saildrone, shown in Fig. 1.4. It is a 7-meter long, narrow hull, 5-meter tall, solid sail autonomous boat with a top speed of 4 meters per second [37]. Saildrone has autonomous waypoint-to-waypoint navigation capabilities. It also possesses a large suite of oceanographic sensors for studying up-welling dynamics, sub-mesoscale variability, air-sea fluxes near ocean fronts, diurnal warming modeling, carbon cycling (related to ocean acidification), and other biophysical interactions. It even has a satellite link for live data capabilities. It is partially funded by a branch of Google. The Saildrone company business model is unique in the oceanography field. In general scientists currently carry the burden of writing grants to purchase sensors, renting an autonomous vehicle or similar service, and paying technicians and engineers for system maintenance and up-keeping. Saildrone owns, operates, maintains, and calibrates all their ASV sensors and oceanographic instruments. They make revenue by instead selling the *data* they collect to scientists. This takes some of the burden off of scientists and may be a lasting and successful business model for ASVs going into the future. Saildrone is also very robust; on September 30th, 2021, it navigated through a category 4 hurricane, recording remarkable video footage inside of the hurricane [31]. Like the Wave Glider, Saildrone is capable of

month long missions or longer as it too scavanges energy, being wind and solar-powered. It's primary goal is to reduce the cost of in situ ocean data collection. It can be argued that this goal is shared broadly with all of these ASV efforts, including this work.

Figure 1.4: Two Saildrone Explorer autonomous surface vehicles docked in Newport R.I.

Some less costly ASV designs, (in which the abbreviation "ASV" may also extend to mean autonomous *sub*-surface vehicle, or autonomous submersible vehicle) that should also be mentioned, include the Light Autonomous Underwater Vehicle or LAUV by OceanScan that was used in some impressive work [32], shown in Fig. 1.5. The LAUV has a partially modular design for extra attachments and sensors. It is roughly 1.2 meters long (depending on the configuration), 15 centimeters wide, and is rated to 100 meters in depth [1].

And last, but not least, is the BlueROV2 by Blue Robotics. Of all the aforementioned ASVs, the BlueROV2 is the least expensive, starting at about $3,490.00,[2] shown in Fig. 1.6. It is a highly modular design, with extra attachments including extra propellers, sensors, and even small robotic arms to grab and move small objects underwater. It is largely meant for use with a communication

[1]More information at https://www.oceanscan-mst.com/light-autonomous-underwater-vehicle/

[2]More information at https://bluerobotics.com/store/rov/bluerov2/

Figure 1.5: Picture of the Light Autonomous Underwater Vehicle LAUV by OceanScan

tether and for remote operation (hence not autonomous).

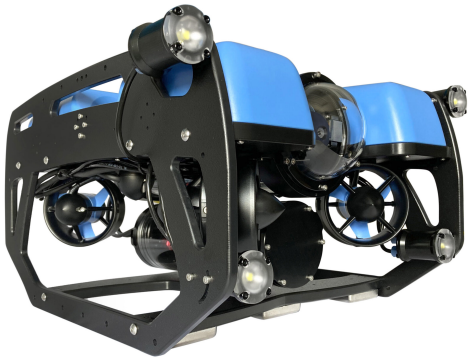


Figure 1.6: Picture the BlueROV2 ASV by Blue Robotics.

Caveats and Considerations

With all of these aforementioned ASVs already in existence, or currently being rolled out, what are the areas of differentiation needed? The previously mentioned projects cover a wide variety of use cases, physical scale, computational

limitations, energy constraints, and varying budget constraints. However, with the exception of the work done by [32] they are largely used for either waypoint-to-waypoint navigation, in which the control system is performing straight line tracking, or simply being remote controlled as is largely the use case of vehicles like the BlueROV2. Maintaining course while subject to disturbances can be challenging, especially when there are disturbances. The missing component is intelligent exploration.

For example, Saildrone, a scientist usually has to manually specify waypoints for the vehicle to navigate for taking measurements. This means that the number of ASVs doing useful scientific missions is limited by the number of personnel that can readily direct them to where they should go to next. One of the main ideas and contributions of this work is to try to automate how to choose those waypoints onboard the ASV during an autonomous mission. Part of the reasoning behind this is due to the fact that the number of autonomous exploration systems will likely (and should ideally) exceed the number of ocean scientists. Put another way; the exploration capabilities and rate of data collection by a fleet of ASVs should not be constrained by the number of people available to direct them. Essentially, this work introduces some methods and modifications of using spatial estimation to inform which global waypoints, with potential scientific value, should be chosen and explored next during a mission.

There are many useful open source software packages for vehicles like the BlueROV2, LAUV, and the Sea Slug, including Ardupilot, and Ardusub. There are also a few different open source mission planning applications that work with these software, which include Q Ground Control, Mission Planner, and others. These are excellent for setting waypoints, and providing a good foundation for autonomous vehicles, but at a high level the user still has to specify each waypoint.

Further more, there remains a lack of readily available software and firmware for hardware peripherals such as inertial measurement units (IMUs), rotary encoders, specific microcontrollers, and other useful hardware. Additionally, there are no options in the way of open source intelligent exploration algorithms for ASVs (or other autonomous vehicles). There is open source software for machine learning, computer vision, and object detection specifically, but many proposed methods of optimal *exploration* are not readily available and have computational complexity that scales poorly with the number of points measured ([74] and [83]).

This thesis highlights the advantages and performance gains of using light-weight, low-cost microcontrollers with similarly low-cost embedded compute platforms, such as the Raspberry Pi. This work also highlights the importance of considering embedded computation design choices for truly cost-effective ASVs[3] System modeling, system identification, guidance, navigation, and control are all discussed in detail; many ASVs discussed in this section don't provide *any* code base. This is unfortunate because that reduces the rate at which ASVs are adopted and are able to be modified to myriad missions.

This work demonstrates that most of the same capabilities of the previously discussed ASVs can be replicated with a far more cost-effective system architecture, using open source and costume-written embedded code. For example, it shown in this work that a simple 32-bit PIC32 microcontroller, with a single GPS module, a servo, and a propeller can execute waypoint-to-waypoint navigation similar to the Saildrone. That setup costs roughly under $200. Later is is also shown how a PIC32 microcontroller working with a Raspberry Pi can be used to do spatial estimation and conduct intelligent waypoint or sample-point generation.

All of the previously mentioned ASVs could be improved by adopting the

[3]It is worth noting that a sizable part of this research was built on top of an embedded C code library written by the author. All of the code used in this work is either already available to the public or is in the process of being made open source.

intelligent exploration algorithms presented, an efficient open source embedded code base, and the light-weight low-cost system architecture later discussed.

1.3.2 Attitude Estimation

Attitude estimation is required for precise ASV guidance; it allows for determining the orientation (and body-fixed angular rates) of the vehicle. For instance, the heading angle of a vehicle is necessary information for a trajectory-tracking control algorithm (see [65]). There is exists a large body of work done on solving the problem of attitude estimation. Popular approaches to attitude estimation include complementary filters (CF) by [50, 26], TRIAD method, Whaba's problem, extended Kalman filters (EKF) by [53, 45], and even machine learning techniques [2].

In this work a CF-based AHRS is chosen for it's ease of implementation and low computational cost. One of the key benefits of the CF-based AHRS is that it's computationally efficient enough to be run on a microcontroller that is simultaneously controlling other peripherals (such as motors, and servos).

1.3.3 Path Planning

Another key aspect in the background research for this work is path planning. This involves the selection of a sequence of points for an autonomous vehicle, or robot, to move through the environment. Notable work in this area includes, [9], [8], and [22].

[9] develops a two-step path planning algorithm to generate optimal UAV flight paths. In [8], a probabilistic path planning algorithm that uses queries to check for collisions is developed. A map is composed of a number of nodes, and is used to compute the shortest path from a starting point to an end point.

The algorithm minimizes the number of collision checks by retaining information of previous queries. Graph searching approaches for autonomous car navigation are discussed in [22], in which algorithms such as A^* are used to plan paths around obstacles detected while moving. More recent work includes [83], which demonstrates using the Kriging Method for field exploration with an autonomous vehicle using multiple different path planners.

Work in [32] introduces mapping coastal ocean phenomena based on temperature and salinity measurements. They use spatial statistics to inform the decision making of a marine robot. Specifically, they use Gaussian Random Fields (GRFs) and the expected integrated Bernoulli variance reduction along with co-Kriging to inform and prioritize sampling locations for the ASV (though they refer to it as an AUV in their work).

Similarly, work by [74] introduces optimal planning based on Gaussian Process Regression (GPR), which shares much of the underlying math and has many parallels to Kriging. Their research explores both single and multi-vehicle planning to reduce the spatial estimation error of a region of space over time.

Caveats and Considerations

A key problem in [83], [32], and [74] is that they all share the need to invert a spatial covariance matrix. This has computational complexity of $O(n^3)$, where n is the number of field measurements. Work by [60] introduces some methods to recursively partition the initial problem space to reduce the computational complexity (this is also known as local Kriging). This dissertation highlights new methods (some contributed by the author in [80]) to also recursively partition the initial problem space to reduce the computational complexity. A key difference is that the new partition method is an *informed* partition scheme because it bases

the field partition sizes *dynamically* on the auto-correlation of the field as it is measured. This is in contrast to other methods which are either static, not able to be used in real-time, or are informed by other methods. Statistical spatial estimation techniques such as Ordinary Kriging are applied dynamically based on the partition update.

It should also be noted that all of these approaches essentially discretize a field, or form a graph, breaking up the problem of path planning into a more computationally tractable form. This means that a number of discrete points on a field with some utility value are considered. Each point may have a gradient of values or a set of distinct possible values. These approaches are applicable, since the path planners for this thesis are implemented on an embedded system.

1.3.4 Trajectory Generation

Trajectory generation relates the chosen path to time constraints and other state variables of a vehicle. This means that when a trajectory is formed, the vehicle has specific state variable values planned for each time step along or between a set of points. It also deals with forming the path based on the the points selected during path planning. In [16], Bézier curve splines are used to form the trajectories, where as in [22] and [65], cubic splines are considered and simulated. Splines are also used in [43] based on Voronoi diagrams.

These works all tend to require knowledge of the closest point to the path and the angle of a tangent vector on the closest point on the path with regard to some reference frame. Path curvature is also a key factor in trajectory generation. If a path is jagged or non-differentiable at a point, this can induce a higher control effort from the vehicles trajectory tracking controller when it passes the point. This is where path smoothing becomes useful, which can be done using splines

and other methods.

1.4 Thesis Contributions

The contributions of this thesis include:

- Autonomous exploration algorithms for static and dynamic path planning based on Ordinary Kriging, or Gaussian Process Regression. This includes online trajectory generation and optimized sample-point generation for maximal uncertainty suppression.
- Experimental demonstration of waypoint line-tracking to sub-meter mean cross-tracker error with existing hardware.
- Simulation and experimental validation of embedded attitude and heading reference system (AHRS) algorithms, including results using a custom AHRS validation apparatus.
- Two novel hardware ASV implementations including low-level drivers and high-level code.
- A full software stack for simulating and collecting real-time data collected by the experimental ASVs.

1.5 Organization

This section outlines the general approach to realize the goal of creating a cost-effective method to conduct oceanography and similar exploration missions. The task of designing the GNC for this purpose was divided into: 1) guidance; answering the question of where should the vehicle go? 2) navigation; how to

plan a path in relation to the vehicle's *state* to successfully get the vehicle to where it needs to go, and 3) control; ensuring the vehicle follows the path. These three parts each involved conducting background research on similar projects, simulating different approaches, comparing performance between approaches, and finally coming up with new methods to do GNC.

Chapter 1 discusses past work on similar projects and introduces some key concepts that we return to throughout this writing. Ch. 2 details the specific systems' design choices. Ch. 3 outlines the Slug 2 and Slug 3 sensor and actuator hardware specifications. Ch. 4 investigates attitude and heading reference systems and compares different methods in simulation. Experimental results for a complementary filter-based attitude estimation algorithm are shared and discussed. Ch. 5 deals with various vehicle models, and how they are derived. Ch 6 discusses how to find the parameters of vehicle models (through system identification) using both ARX and an extended Kalman filter (EKF). Ch. 7 delves into autonomous navigation with an emphasis on trajectory generation, tracking, control, B-Spline generation embedded systems, and how to estimate vehicle-system state variables and model parameters. Comparisons of different ways to track and structure trajectories is given, along with computational and accuracy results for attitude estimators. Ch. 8 covers intelligent exploration based on spatial estimation techniques such as ordinary Kriging and Gaussian Process Regression (GPR). New techniques for speeding up the spatial estimation are introduced with results. Optimal exploration is discussed as well, with a focus on graph search algorithms. Ch. 9 goes over the experimental results of the ASV implementations. We conclude with Ch. 10 and discuss the research as a whole, and future work. Other work and further results are found in Appendix A. Generally, additional background is provided at the beginning of each chapter, and the relevant contributions are listed

towards the end of each chapter.

Chapter 2

System Implementations

This chapter examines the hardware and software architecture of the Slug 2 and Slug 3 ASVs outlined throughout this thesis, how it evolved over time, and the reasons for specific design choices. Both ASVs are shown in Fig. 2.1 and Fig. 2.2. A system block diagram is provided in Fig. 2.3. Subsequent chapters provide more detail on each block.

2.0.1 Connection to the Overarching Theme

The specific implementation of an ASV is not directly necessary for an oceanographer to know. However, it is important to outline the system architecture. This chapter shows what communication protocols were used for each sensor and actuator and how they work together. Additionally, the computation platforms are discussed along with how they are connected and what information is shared between different components. This architecture is easy to modify for different types of missions, extending its use-cases to various kinds of researchers that might deploy it in different environments. Fig. 2.1 and Fig. 2.2 show the Slug 2 and Slug 3 respectively. They represent two instances of using the architecture discussed

in this chapter. The Slug 3 system block diagram is shown in Fig. 2.3.

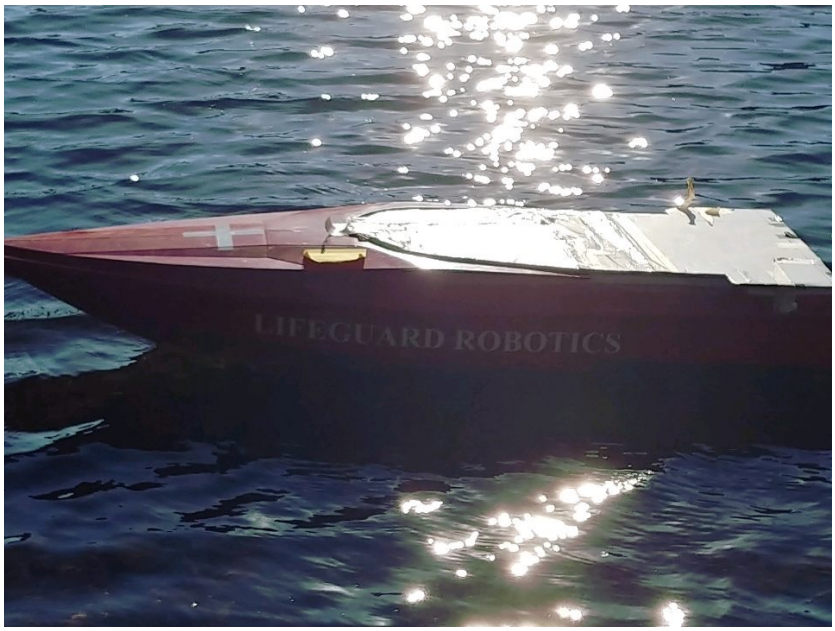


Figure 2.1: The Slug 2 autonomous surface vehicle.

Figure 2.2: The Slug 3 autonomous surface vehicle.

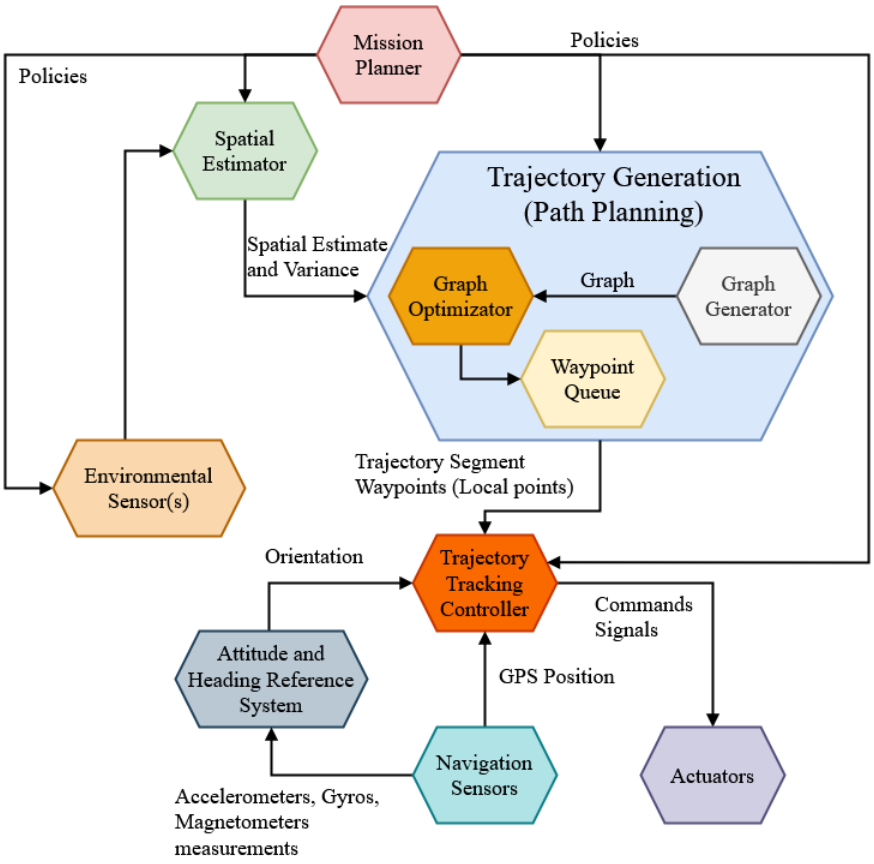


Figure 2.3: The top-level system block diagram.

2.1 Hardware Design

This section goes over the hardware design implementation for both the Slug 2 and Slug 3 ASVs. Initially, the Slug 2 was the primary test vehicle. It used a single PIC32 microcontroller that interfaced with a GPS, an IMU, an ESC for the propeller motor, a servo for the rudder, a radio transceiver, and an RC receiver.

Both the Slug 2 and Slug 3 use multiple different kinds of sensors, actuators, transmitters, and receivers. Highlighting the necessary protocols for the ASV leads to a deeper understanding of how the system works with all the peripheral devices. The ASVs uses the Max32 microcontroller board which is based on a PIC32MX79F128H microprocessor; the overall system architecture and underly-

ing code is meant to be extensible to other microcontroller with only small changes to the low-level drivers. The microcontroller is capable many communication protocols: UART, I2C, SPI, and CAN. Some protocols, such as UART have multiple channels such that multiple different devices can communicate to the microcontroller. All of the drivers for these protocols were written from scratch, tested, and successfully employed in the validation procedure discussed in [79].

2.2 Software: Communication Protocols and Peripherals

The software discussion centers around the communication protocols necessary for the different hardware peripherals to communicate with the central microcontroller and Raspberry Pi4B. Some sensors can be directly connected to the Raspberry Pi, and may even include pre-written and pre-tested firmware drivers that are part of a commercially available product (such is the case for the depth sensor). However, a majority of the hardware peripherals connected to the Max32 microcontroller rather than the Raspberry Pi. They required custom C code for communication protocols including UART, I2C, and SPI. Additionally, the MAVLink and NMEA protocols were integrated with two separate serial drivers. The MAVLink integration allowed the Max32 and Rasperry Pi could send MAVLink packets between each other over USB serial or a similar connection. The NMEA protocol was used for the GPS module connected to the Max32.

UART

There are four UART channels on the Max32. The first of these channels is dedicated to serial output for debugging and code development with a standard

computer. The second channel is used for radio telemetry. Anything from individual ASCII characters to more advanced packet structures with a checksum may be transmitted and received over this channel. The third channel is to interface with the GPS unit. The standard NMEA packet structure is used and a custom NMEA library was written to obtain COG, speed, position, and HDOP (the quality of the horizontal degradation of the position measurements). The fourth channel is used for receiving commands from a remote controller for manual control of the ASV. This is critical in case the GNC algorithm fails while testing and manual control is necessary, or for testing other aspects of the ASV.

I2C

The inter-integrated circuit (I2C) communication protocol is used primarily for communication with the sensor-head (MPU9250) at 100Hz. This is used to measure body-fixed angular rates (p, q, r) using tri-axis rate gyros, specific force using the tri-axis accelerometers, and the magnetic field using the tri-axis magnetometer. Note that the sensor-head frame is not necessarily coincident with the vehicle body frame.

SPI

The serial peripheral interface (SPI) communication protocol is used to read the magnetic quadrature phase encoder (AS5047D). This is the encoder used in the validation procedures for the Complementary Filter based Attitude and Heading Reference System (AHRS), but is also intended to be used to measure directly measure the rudder angle.

2.2.1 MAVLink

For efficient data collection from the onboard sensors, the MAVLink[1] packet structure is used. This allows use of convenient programs such as QGroundControl, an open-source program to track vehicles' locations on virtual maps while recording vehicle telemetry. Hardware-in-the-loop (HIL) simulations have already been conducted that use MAVLink to transmit data to the ground station for validation of the embedded GNC algorithm (see Ch. 7).

2.2.2 Kill Switch

One of the biggest problems encountered during system integration testing is implementing a safety precautions enabling user take-over or manual control of the ASV, including stopping the vehicle's primary propulsion motor. Fig 2.4 shows how the kill switch circuit (RXc) connects to the microcontroller, and transistor connecting the electronic speed controller (ESC) for the primary motor to ground. The motor speed, rudder servo angle, and operation mode signals are received by the RXc and passed to the microcontroller. Then it sends the motor speed and rudder servo angle signals to the ESC and and servo respectively. The mode signal is processed in software on the microcontroller to switch between manual and autonomous mode. There is a fourth signal that does not get sent to the microcontroller, but instead is sent directly to a transistor to connecting the ESC to ground. If this signal is high the transistor disconnects the ESC from ground, effectively cutting power to the motor, but not the microcontroller. This is important because it is a hardware fail-safe, meaning no software error will prevent the signal from reaching the transistor. Thus power to the motor can be cut, while leaving power for both the microcontroller and servo. The servo doesn't

[1]See https://mavlink.io/ for more information on the MAVLink protocol

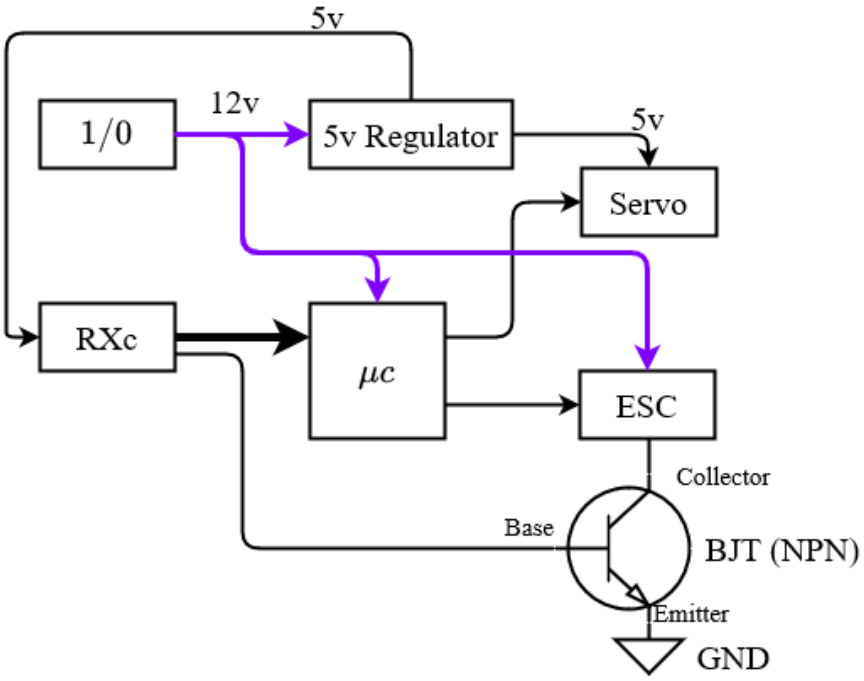


Figure 2.4: Actuator system block diagram with RXc, or kill switch circuit block.

have power cut, because it might be convenient or necessary to move the servo in the event of user override. For example, the servo can be moved to a maximum angle such that the rudder drag in the water increases and slows down the ASV.

The Slug 3 built also used most of the existing drivers, and used the same micrcontroller, but it had three major design changes: 1) it used a different catamaran-type hull, 2) it used an additional custom-made PCB board ontop for easier interfacing with peripheral devices (including a new IMU), and 3) it interfaced with a Raspberry Pi 4B over USB UART. Figure 2.5 shows the system block diagram for the Slug 3.

The addition of the Raspberry Pi was driven by the need for high-level computation, additional memory capacity, and file system use for black-box data logging. As was mentioned earlier, the radio telemetry would occasionally fail to send receiver a message at the ground station. To solve this, the microcontroller

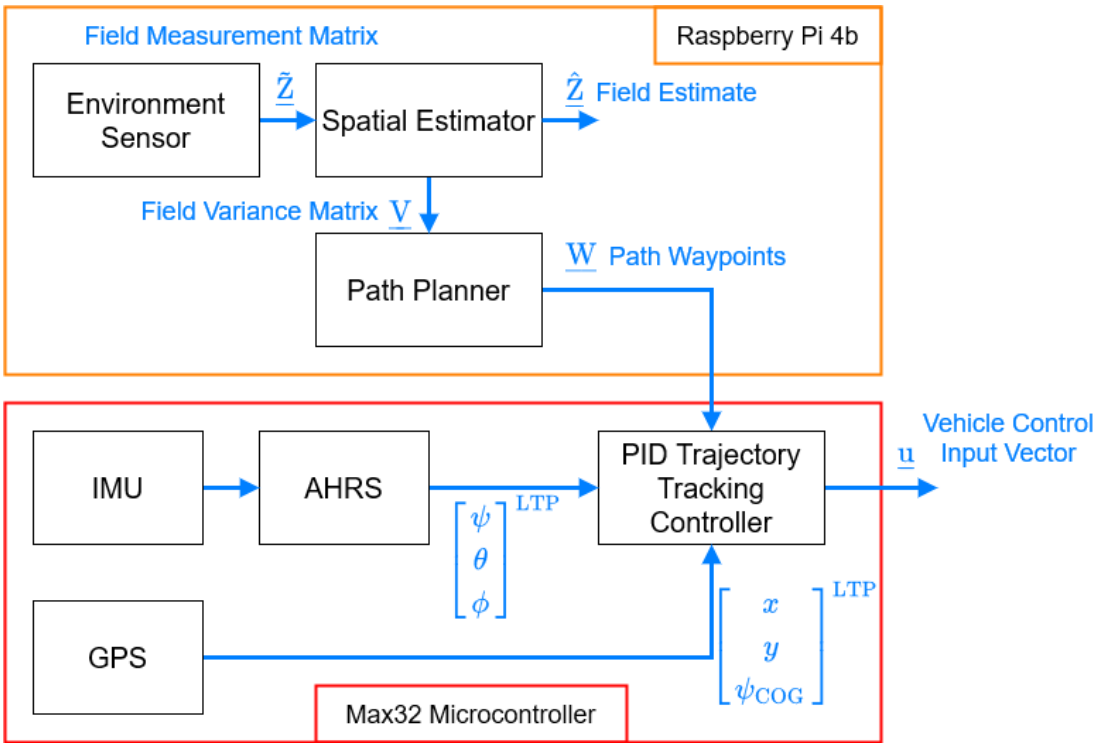


Figure 2.5: The system block diagram for the Slug 3 ASV.

sent MAVLink messages over USB UART to the Raspberry Pi, where they were recorded onto a USB drive. This eliminated the packet loss issue due to radio transmission, because system data could be recovered after deployment.

Furthermore, the Raspberry Pi could compute things like the spatial estimation quickly, while the lower-level hardware peripherals could be dealt with by the microcontroller simultaneously.

2.3 Conclusion and Caveats

This chapter outlined the hardware and software design of the Slug 2 and Slug 3. The design choices were made with regard to affordability and time for system integration. Special attention was given to the various software drivers and how they were used to interact with the hardware peripherals. The Max32

microcontroller was shown to be a capable computation platform for vehicles of this type. One of the aspects of the hardware design that could be further developed is the mechanical design. Specifically, a custom hull could be designed and fabricated; if the hull modular, then it could potentially increase the number of use-cases for such an ASV. A mission specific ASV could be customized to a certain size for payloads.

Chapter 3

Sensors and Actuators

A system such as an ASV requires sensors to perceive it's surroundings, aid in localization, and to determine it's attitude (3D orientation) relative to it's surroundings. The problem of determining orientation or attitude of the vehicle is complicated and discussed further in Ch. 4. The main sensors on the ASV include: 1) an inertial measurement unit (IMU) sensor-head with tri-axis accelerometers, gyroscopes, and magnetometers, 2) a temperature sensor, 3) an echo sounder depth sensor, and 4) a GPS receiver. A rotary encoder was also used, but not onboard the vehicles. It was as part of an attitude validation apparatus that is discussed in Ch. 4. The main actuators include: 1) the rudder servo, and 2) the brush-less direct current (BLDC) motors for the propeller(s). Note that the Slug 2 had one main propeller and the Slug 3 had two, but both ASVs used a single rudder servo.

3.0.1 Connection to the Overarching Theme

The sensors and actuators used by an ASV are critical to its ability to navigate, sense, and interact with the local environment. This chapter outlines some

of the necessary sensors and actuators shared by many autonomous vehicles. One of the sensors of special note is the echo sounding depth sensor; This sensor represents the environmental field attribute sensor that is mission specific to this thesis. It will be used later in measuring the depth field of a small body of water. It is not a particularly interesting field attribute sensor, but it is *representative* of similar sensors that are location-specific, and measures phenomena not easily measured at large distances. For example a more ideal sensor would be micro-plastic-concentration sensors; also a location specific sensor and would take the place of the echo sounding depth sensor. This would mean that the field being estimated would be one of micro-plastic density rather than a depth field. Fortunately the estimation techniques discussed in Ch. 8 can be used for many different kinds of stationary field attributes.

3.1 Sensors

This section outlines the sensors used for both the Slug 2 and Slug 3 autonomous surface vehicles. A brief discussion of each sensor implementation is provided. This includes software drivers that were necessary to interface with the microcontroller, record data provided by each sensor, and the update rate for each sensor. Most of the sensors described here used custom C code for interfacing the Max32 microcontroller and Raspberry Pi 4b.

3.1.1 IMU

Two different IMUs were employed. The Slug 2 used the MPU-9250 Sparkfun breakout board. It connected to the Max32 microcontroller using a I2C wire connection. The I2C drivers were written to be non-blocking. This allowed for

reading bytes corresponding to the tri-axis accelerometers, gyroscopes, and magnetometers while other software tasks were carried out. It should be noted that the MPU-9250 used a separate integrated circuit for the magnetometer with a different I2C address than the MPU-9250. This IMU was read with an update of 100Hz, but was limited to 8Hz for the magnetometer. It was powered directly from the Max32 at 3.3 volts. As of this writing this IMU is no longer available. The Slug 3 used the ICM 20948 IMU; it is very similar to the MPU-9250. This used an SPI driver for faster measurement updates (up to 7MHz), though that speed was not necessary.

The calibration and implementation of both of these IMUs in gereneral is discussed in Ch. 4.

3.1.2 NTC Thermistor

An off-the-shelf NTC thermistor was briefly used in an experiment with the Slug 2. It was used to measure the temperature of the water at different locations. However, the temperature of a small body of water tends to be the almost the same value regardless of the location, unless there is a local heat source. Even then, the temperature gradient is so small as to not be very interesting for the purposes of field attribute estimation. The sensor connected to a power pin, a ground pin, and an analog to digital (ADC) pin on the Max32. The ADC module on the Max32 was configured to be non-blocking and interrupt driven.

3.1.3 Echo Sounder Depth-Sensor

The echo sounder works by emitting a directed sound wave in the water. When the sound wave bounces off of an object (such as a fish or the ocean floor) the time of flight is used to calculate the distance to the object. This sensor was

used on the Slug 3 to take depth measurements of small bodies of water, such as ponds and lakes. Fig. 3.1 shows the depth sensor used onboard the Slug 3. It connects directly to the Raspberry Pi4B via USB serial connection. No custom code was necessary to write for this sensor as Blue Robotics provides an open source repository for using their python module.

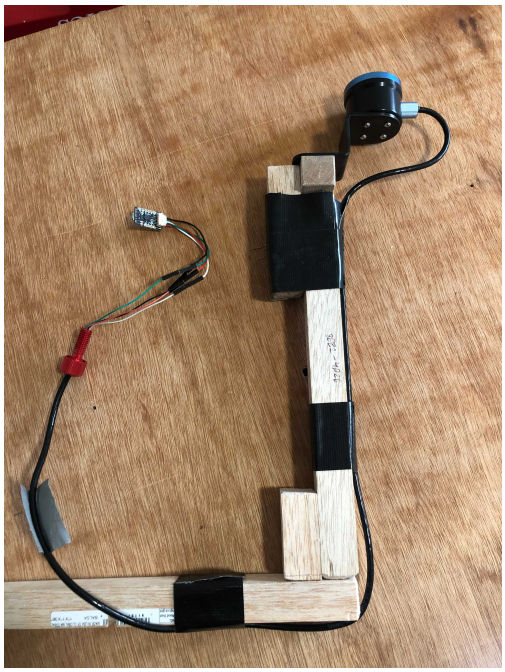


Figure 3.1: The Blue Robotics echo sounder depth sensor attached to a wooden arm. The arm connected to the top back of the Slug 3 to point downward.

Calibration

The depth sensor was used to measure known "true" distances. Specifically, a large/thin plastic sheet was set at a series of known distances away from the depth sensor. The depth sensor measurements were compared to the true distance values. They agreed with the true to within approximately $\sim \pm 0.5$in. Fig. 3.2 shows the distance measured by the depth sensor and the known distance to the sheet. Note that the transitions between different sets of known distances was

interpolated with a simple ramp. This is an approximation and in reality the sheet was not perfectly moved at a constant velocity.

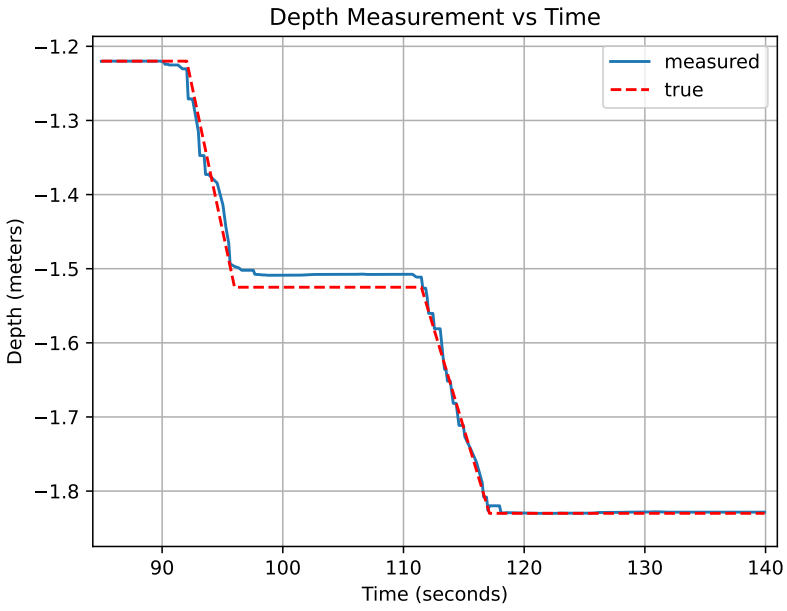


Figure 3.2: The measured distance from the depth sensor and the true depth signals over time. The transition between set distances were changed to ramps based on the time stamp when the sheet was moved.

Fig. 3.3 shows the depth sensor measurements versus the true reference distance. Fig. 3.4 shows the distance error over time. Fig. 3.5 shows the distance error histogram. Note that these figures are before any calibration.

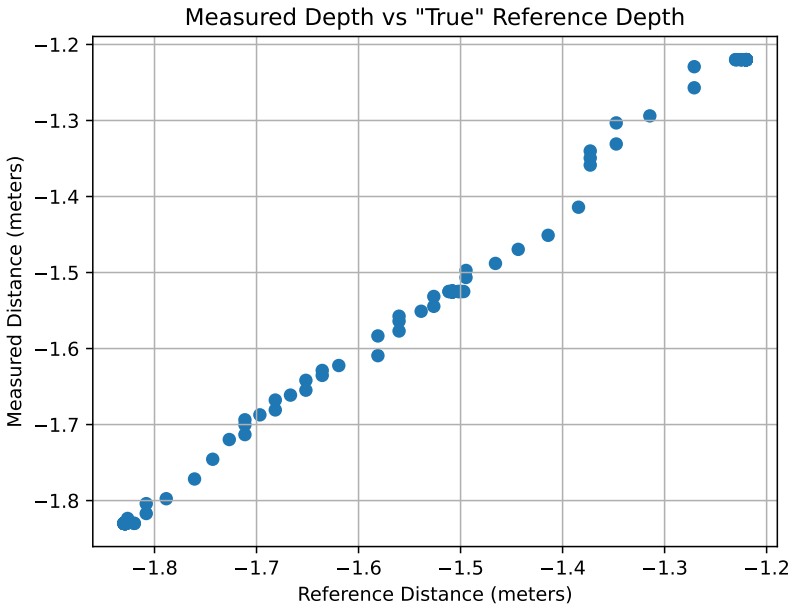


Figure 3.3: The depth sensor measurements versus the true reference distance.

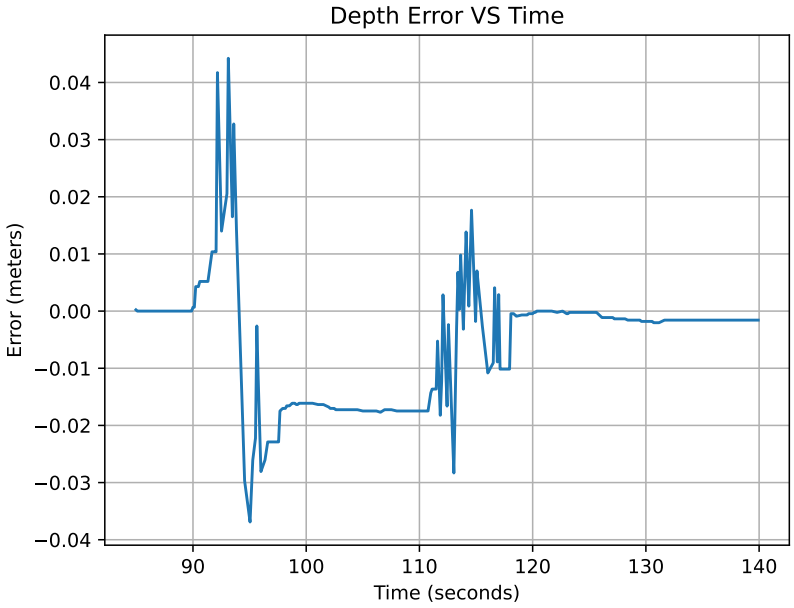


Figure 3.4: The distance error between the depth sensor measurements and the true reference distance over time.

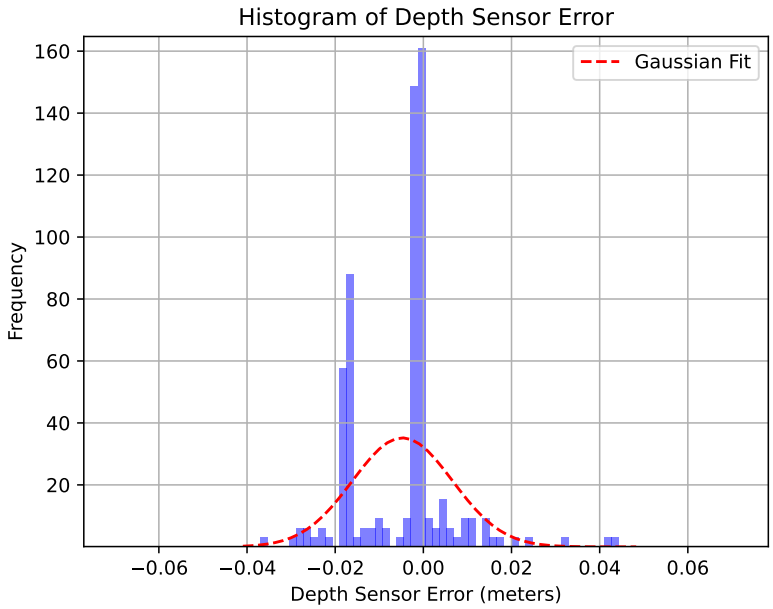


Figure 3.5: A histogram of the depth sensor error.

Before calibration the depth sensor possessed a mean error of approximately -0.00474 meters with a standard deviation of 0.01134 meters (0.44646 inches). The calibration procedure was done using least-squares. The sensor measurements are represented by the following equation:

$$y = mx + b \tag{3.1}$$

where y is a measured distance, m is a scale factor, x is a "true" distance and b is a scalar offset. Note that at this point m and b are not yet known, but can be estimated as we will show below. First, let us re-write Eq. (3.1) as

$$x = \frac{y - b}{m} \tag{3.2}$$

Now let

$$\mathbf{q} = \mathbf{S}\mathbf{r} \tag{3.3}$$

and

$$\overbrace{\begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_{n-1} \end{bmatrix}}^{\mathbf{q}} = \overbrace{\begin{bmatrix} y_0 & 1 \\ y_1 & 1 \\ \vdots & \vdots \\ y_{n-1} & 1 \end{bmatrix}}^{\mathbf{S}} \overbrace{\begin{bmatrix} \frac{1}{m} \\ \frac{b}{m} \end{bmatrix}}^{\mathbf{r}} \tag{3.4}$$

where $\mathbf{q} \in \mathbb{R}^{n\times 1}$ is the vector of true distance values, $\mathbf{S} \in \mathbb{R}^{n\times 2}$ is a matrix of measured values and a vector of ones, and $\mathbf{r}$ is a vector of coefficients. We may now estimate the coefficient vector $\mathbf{r}$ with least-squares,

$$\mathbf{r} = (\mathbf{S}^{\mathrm{T}}\mathbf{S})^{-1}\mathbf{S}^{\mathrm{T}}\mathbf{q} \tag{3.5}$$

It was found that $1/m = 0.999932$ and $b/m = 0.004629$. With two equations and two unknowns we may solve and find that $m =\sim 1.000070$ and $b =\sim 0.004630$. The coefficients can then be applied to the original data using Eq. (3.1). Due to the sensors initial level of accuracy the calibration improvement is small; the mean error becomes -0.000218 meters and the standard deviation becomes 0.011340 meters (0.446444 inches). Based on these metrics, the depth sensor was deemed to possess a sufficient level of accuracy for experimental use.

3.1.4 GPS Receiver

The GPS receiver used was the SparkFun GPS breakout board chip antenna SAM-M8Q. It connected to the Max32 directly with a UART wired connection. A custom NMEA0183 UART parsing module was written in C for the Max32 to

read the GPS data. This included latitude, longitude, course over gound (COG), horizontal dilution of precision (HDOP), vertical dilution of precision (VDOP), and other standard GPS data. The NMEA0183 parsing module was written to be non-blocking to allow for other software tasks to be carried out "simultaneously."

3.1.5 Encoder

The encoder used with the attitude and heading reference system (AHRS) validation apparatus, discussed in Ch. 4 was the AS5047D high speed position sensor. It uses a hall-effect sensor and magnet to detect the angular orientation of rotating axle connected to the magnet. A non-blocking interrupt driven SPI driver was written in C to allow it to interface with the Max32 microcontroller.

3.2 Actuators

The Slug 2 and Slug 3 both used essentially the same hobby servos for the rudders and electronic speed controllers (ESCs) for the propellers. Custom C code was written to configure the output compare pins of the Max32 microcontroller to drive both actuators.

3.2.1 Servo

A standard hobby servo was used to move the rudders for both the Slug 2 and Slug 3. They connect to the Max32 microcontroller with a power (5 volt), ground, and signal connection. However, the Max32 used a custom daughter-board that allowed for a 5-volt rail with a higher amperage rating, so as not to draw potentially damaging excessive current through the Max32 5-volt regulator. A servo pulse was generated using custom pulse-width-modulation (PWM) module

written in C code for the Max32. It was non-blocking and interrupt driven. The servo pulse width could change between 1.0 and 2.0 milliseconds, corresponding to a rudder angle of approximately ±45°. Fig. 3.6 shows a conceptual depiction (not to scale) of the PWM signal for the servo.

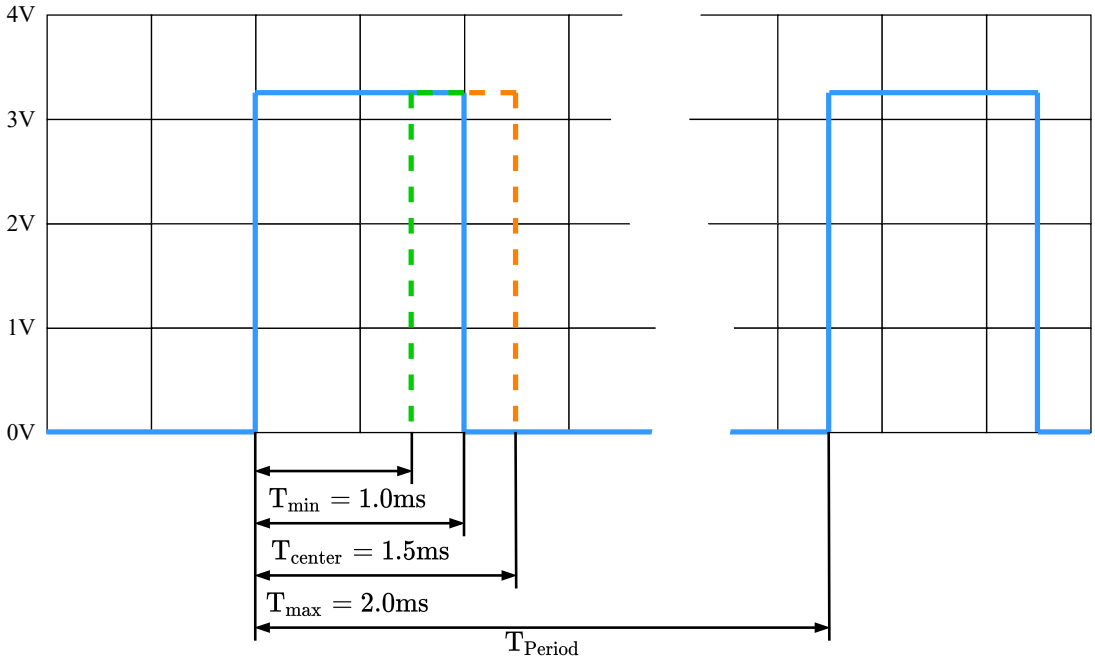


Figure 3.6: A drawing of the PWM signal (blue) for the rudder servo. The duty cycle changes such that the high time T falls within a range of $[1.0, 2.0]$ms with the center at 1.5ms.

The servo is driven using a PWM signal, shown in Fig. 3.6. By adjusting the high-time of the signal (the duty cycle), the change in the effective voltage to the servo causes the servo to rotate. The minimum servo angle corresponds to a high-time of T_{min} and the maximum angle corresponds to T_{max}.

3.2.2 BLDC Motor

The same output compare pins associated with the PWM module for the Max32 were used to drive a signal similar, but different signal than the servos.

The signal was sent into the Turnigy Plush 40A ESCs that would in-turn drive a Radient Reaktor BLDC motor. This motor was connected to the main propeller on the Slug 2. The Slug 3 used similar motors with similar ESCs, using the same PWM signal as the Slug 2.

3.3 Conclusion and Caveats

This chapter outlined the sensors and actuators used by the Slug 2 and Slug 3 autonomous surface vehicles. It discussed how the hardware was integrated and articulates the capability of a single microcontroller to interface with all of these peripherals "simultaneously" using efficient custom embedded C code. It should be noted that all of these sensors and actuators are relatively inexpensive with the exception of the Blue Robotics echo sounder depth sensor. These actuators and sensors are representative of those found on more expensive ASVs such as the SailDrone, but are not as precise or accurate. These components were chosen with regard to a constrained budget and to minimize the amount of work needed to integrate them with the Slug 2 and Slug 3 ASVs.

Chapter 4

Attitude and Heading Reference System

An attitude and heading reference system (AHRS) is commonly used on autonomous vehicles. An AHRS is used to provide roll, pitch, and yaw estimates for aircraft, satellites, and other vehicles that require accurate, onboard measurements of their orientation in three-dimensional space. An ARHS generally consists of a set of three-axes gyroscopes and some aiding sensors (e.g.: three-axis magnetometers and/or accelerometers) that are used together in a sensor fusion algorithm to estimate vehicle orientation (or attitude). Gyroscopes provide angular rates about the three orthogonal axes in the body-frame of a vehicle. Ideally, the orientation of the vehicle can be found by integrating these body-fixed angular rates, though the process is non-linear and has some particularities depending on the specific attitude parameterization. Due to inherent integration errors and time-varying drift of the rate biases of the gyros, aiding sensors are required to estimate the true gyro drift rates and improve the final attitude estimate. Note that the rate gyros are incapable of resolving the initial attitude, and that the aiding sensors are required to converge on the initial attitude estimate as well. The ASV uses a

sensor-head with a three-axis MEMs accelerometer, gyroscope, and magnetometer. The measurements from the sensor-head can be combined using sensor fusion techniques such as Extended Kalman Filters (EKF), TRIAD, or complementary filters (CF) as part of an algorithmic design of an AHRS. In this section, both simulation and experimental implementations for different ARHS algorithms are explored. First, a simulation is used to show the advantages and disadvantages of three different AHRS options: 1) EKF, 2) TRIAD, and 3) CF. We examine the performance in terms of computation time and accuracy.

Depending on the size, weight, and power (SWaP) of the AHRS, and the budget available, differing validation tests can be used to ensure that the AHRS is correctly estimating the attitude and within the required specifications for the mission. This can be achieved with rate tables, or similarly expensive apparatus, where external sensors provide a measurement of the orientation of the vehicle. The external truth reference (external measurement) is compared to the attitude generated by the AHRS (internal measurements). This comparison between true and estimated orientation is used to validate the system. If the internal and external measurements agree (e.g.: low mean difference and standard deviation), then the AHRS can be used with high confidence.

Part of the navigation component of GNC is discussed in this chapter and is largely focused on attitude estimation through using an AHRS. The use of GPS and course over ground (COG) is also discussed to address how to guide a vehicle. However, Ch. 7 is dedicated to the other key aspects of GNC, and a more comprehensive discussion as the system is implemented in this thesis is found here.

4.0.1 Connection to the Overarching Theme

ASV attitude is a hard requirement for autonomous vehicle guidance - it simply cannot be accomplished without a decent estimate. In addition to the vehicle trajectory control the ability of an ASV to determine its full 3D orientation can be very useful to an oceanographer or similar researcher. For instance, a researcher receiving telemetry from an ASV indicating that the vehicle is pitching with a high amplitude oscillation can indicate to the researcher what kinds of ocean conditions and weather are present in an area. Furthermore, orientation could be useful if the field attribute sensor takes measurements that require a specific orientation. This could include taking underwater images that require the ASV to assume a specific orientation within some margin of error at different locations.

4.1 ARHS Comparison

This section discusses AHRS algorithms including the EKF, TRIAD, CF, along with simulations comparing the three. Additionally, a validation apparatus is presented that was created as part of the research for this thesis. It was used in validating a custom CF AHRS implemented on a micocontroller in C code. The AHRS was used onboard a small ASV with a GPS receiver and the course over ground (COG) angle was compared to the CF yaw angle for further experimental validation.

4.1.1 EKF AHRS

The EKF is a computationally intensive AHRS algorithm compared to CF and TRIAD. This is shown quantitatively subsection 4.1.4. A version of an EKF is discussed here briefly for completeness and to substantiate the claim of its

comparatively significant computational demand.

There are two main steps: 1) prediction and 2) correction (sometimes called the update). The prediction step is comprised of the following two equations:

$$\hat{\mathbf{x}}_{k+1}^{(-)} = \mathbf{f}(\mathbf{x}_k^{(+)}, \mathbf{u}_k) \tag{4.1}$$

$$\mathbf{P}_k^{(-)} = \Phi(\mathbf{x}_{k-1}, \mathbf{u}_k)\mathbf{P}_{k-1}^{(+)}\Phi^{\mathrm{T}}(\mathbf{x}_{k-1}, \mathbf{u}_k) + \mathbf{Q}_{k-1} \tag{4.2}$$

$$\Phi_k = \frac{\partial \mathbf{f}_k(\mathbf{x}_k, 0)}{\partial \mathbf{x}_k} \tag{4.3}$$

where $\mathbf{u}_k$ is the control input to the system at the k-th time step, Φ is the discrete state space transition matrix, $\mathbf{Q}$ is the process noise covariance matrix, $\mathbf{P}$ is the estimation covariance matrix, and $\mathbf{x}$ is the state vector. The correction update step is defined by Eqs. (4.4)-(4.8)

$$\mathbf{v}_k = \mathbf{z}_k - \mathbf{h}(\mathbf{x}_k) \tag{4.4}$$

$$\mathbf{S}_k = \mathbf{H}(\mathbf{x}_k)\mathbf{P}_k^{(-)}\mathbf{H}^{\mathrm{T}}(\mathbf{x}_k) + \mathbf{R}_k \tag{4.5}$$

where $\mathbf{R}$ is the sensor noise covariance matrix. The Kalman gain is then define as

$$\mathbf{K}_k = \mathbf{P}_k^{(-)}\mathbf{H}^{\mathrm{T}}(\mathbf{x}_k)\mathbf{S}_k^{-1} \tag{4.6}$$

$$\mathbf{x}_k^{(+)} = \hat{\mathbf{x}}_k^{(-)} + \mathbf{K}_k\mathbf{v}_k \tag{4.7}$$

$$\mathbf{P}_k^{(+)} = (\mathbf{I} - \mathbf{K}_k\mathbf{H}(\mathbf{x}_k))\mathbf{P}_k^{(-)} \tag{4.8}$$

where $\mathbf{P}_k^{(-)}$ is the error covariance extrapolation and $\mathbf{P}_k^{(+)}$ is the error covariance update at the k-th time step. Specifically, in order to match our TRIAD and CF forms we use the attitude prorogation of quaternions.

Now that we have reviewed the EKF equations, we examine how they relate to the rotation kinematics for attitude estimation. To that extent the EKF AHRS implementation from [68] is repeated here for completeness. The same notation is used so that the reader can more easily reference the original text. The Earth-fixed frame is represented by $\mathbf{E} = \begin{bmatrix} \mathrm{e}_1 & \mathrm{e}_2 & \mathrm{e}_3 \end{bmatrix}$. The body-fixed frame is represented by $\mathbf{B} = \begin{bmatrix} e'_1 & e'_2 & e'_3 \end{bmatrix}$. As noted in [68], a vector $\mathbf{x} \in \mathbb{R}^3$ can be translated from the body-fixed frame to the Earth-fixed frame, $\mathbf{x}_\mathcal{B} = {}^\mathcal{B}_\mathcal{E}\mathbf{C}\mathbf{x}_\mathcal{E}$.

Quaternions are used to represent orientation in [68], where $\bar{\mathbf{q}} = \begin{bmatrix} q_1 & q_2 & q_3 & q_4 \end{bmatrix}^{\mathrm{T}} = \begin{bmatrix} \mathbf{q}^{\mathrm{T}} & q_4 \end{bmatrix}^{\mathrm{T}}$. The first three elements of $\mathbf{q}$ are complex and q_4 is real. Let

$$\begin{bmatrix} \mathbf{q}\times \end{bmatrix} = \begin{bmatrix} 0 & -q_3 & q_2 \\ q_3 & 0 & -q_1 \\ -q_2 & q_1 & 0 \end{bmatrix} \tag{4.9}$$

$$\mathbf{q} = \begin{bmatrix} q_1 & q_2 & q_3 \end{bmatrix}^{\mathrm{T}} = \sin(\theta/2)\mathbf{n}^{\mathrm{T}} \tag{4.10}$$

$$q_4 = \cos(\theta/2) \tag{4.11}$$

where $\mathbf{n}^{\mathrm{T}} \in \mathbb{R}^3$ is the vector representing the axis of rotation and θ the angle.

Multiplication between quaternions is defined as

$$\bar{\mathbf{q}}^a \otimes \bar{\mathbf{q}}^b = \begin{bmatrix} \left(q_4^b \mathbf{q}^a + q_4^a \mathbf{q}^b + \left[\mathbf{q}^a \times \right] \mathbf{q}^b \right)^{\mathrm{T}} & q_4^a q_4^b - \mathbf{q}^a \cdot \mathbf{q}^b \end{bmatrix} \tag{4.12}$$

where $\bar{\mathbf{q}}^a$ and $\bar{\mathbf{q}}^b$ are arbitrary quaternions. Let the pure vector quaternion $\bar{\mathbf{x}}_{\mathcal{E}} = \begin{bmatrix} \mathbf{x}_{\mathcal{E}}^{\mathrm{T}} & 0 \end{bmatrix}^{\mathrm{T}}$.

The kinematics equations are,

$$\frac{d}{dt}\bar{\mathbf{q}} = \frac{1}{2}\bar{\mathbf{q}} \otimes \begin{bmatrix} \omega_{\mathcal{B}}^{\mathrm{T}} & 0 \end{bmatrix}^{\mathrm{T}} = \frac{1}{2} \begin{bmatrix} -\left[\omega_{\mathcal{B}} \times \right] & \omega_{\mathcal{B}} \\ -\omega_{\mathcal{B}}^{\mathrm{T}} & 0 \end{bmatrix} \bar{\mathbf{q}} = \Omega(\omega_{\mathcal{B}}) \mathbf{q} \tag{4.13}$$

where $\Omega(\omega_{\mathcal{B}}) \in \mathbb{R}^{4\times 4}$ is a skew symmetric matrix. The discrete-time equivalent of Eq. (4.13) is

$$\bar{\mathbf{q}}(t_k) = \Phi \bar{\mathbf{q}}(t_{k-1}) \tag{4.14}$$

such that

$$\Phi = \cos(|\mathbf{u}_{\mathcal{B}}|/2)\mathbf{I}_{4\times 4} + \frac{\sin(|\mathbf{u}_{\mathcal{B}}|/2)}{|\mathbf{u}_{\mathcal{B}}|/2} \Omega(\omega_{\mathcal{B}}(t_{k-1})) \tag{4.15}$$

and corresponds to Eq. (4.3) of the EKF equations outlined earlier. A key assumption here is that the angular velocity is assumed to be constant within the sample time $\Delta T = t_k - t_{k-1}$, such that

$$\mathbf{u}_{\mathcal{B}} = \int_{t_{k-1}}^{t_k} \omega_{\mathcal{B}}(\tau) d\tau \approx \omega_{\mathcal{B}}(t_{k-1}) \Delta T \tag{4.16}$$

The sensor model for the tri-axis body-fixed gyroscope model ω_m, accelerometer model $\mathbf{a}_m$, and magnetometer model $\mathbf{h}_b$ from [68] is show below:

$$\omega_m = {}^g\mathbf{K}\omega_{\text{body}} + {}^g\mathbf{b} + {}^g\mathbf{v} \tag{4.17}$$

$$\mathbf{a}_m = {}^a\mathbf{K}_{\mathcal{E}}^{\mathcal{B}}\mathbf{C}(\mathbf{a}_{\text{body}} - \mathbf{g}) + {}^a\mathbf{b} + {}^a\mathbf{v} \tag{4.18}$$

$$\mathbf{h}_m = {}^m\mathbf{K}_{\mathcal{E}}^{\mathcal{B}}\mathbf{C}(\mathbf{h} + {}^h\mathbf{b}) + {}^m\mathbf{b} + {}^m\mathbf{v} \tag{4.19}$$

where $\mathbf{C} \in \mathbb{R}^{3\times3}$ is the rotation matrix between different reference frames, ${}^g\mathbf{K}, {}^a\mathbf{K}, {}^m\mathbf{K}$ are scale factor matrices, ${}^g\mathbf{b}, {}^a\mathbf{b}, {}^m\mathbf{b}$ are the bias vectors, and ${}^g\mathbf{v}, {}^a\mathbf{v}, {}^m\mathbf{v}$ are Gaussian noise on the measurements for the tri-axis gyroscopes, accelerometers, and magnetometers respectively. The noise is assumed to be zero-mean. The constant gravity vector in the body-fixed frame is $\mathbf{g}$. The time-varying accelerometer measurements in the body-fixed frame are represented by $\mathbf{a}_{\text{body}}$. The constant magnetic field vector in the body-fixed frame are represented by $\mathbf{h}$ and the time-varying magnetometer variations in the body-fixed frame are represented by ${}^h\mathbf{b}$. Note that the super-scripts g, a, m indicate the gyroscope, accelerometer, and magnetometer respectively. A simplification stated in [68] is that $\mathbf{a}_{\text{body}} \approx \mathbf{0}$.

The gyro-bias is modeled as a random-walk vector,

$$\frac{d}{dt}{}^g\mathbf{b} = \mathbf{w}_g \tag{4.20}$$

where $\mathbf{w}_g$ is Gaussian noise. It possesses a zero-mean, and a covariance matrix, ${}^b\Sigma_g = {}^b\,\sigma_g\mathbf{I}_{3\times3}$

In [68] the magnetic distortions are represented by ${}^h\mathbf{b}$ and can be modeled using a first-order Gauss-Markov vector random process,

$$\frac{d}{dt}{}^h\mathbf{b} = -\alpha^{\mathbf{b}} + \mathbf{w}_h \tag{4.21}$$

where $\mathbf{w}_h$ is zero-mean Gaussian noise and α is a positive constant. The corre-

sponding covariance matrix for the noise is ${}^{b}\Sigma_h = {}^{b}\sigma_h^2\mathbf{I}_{3\times3}$.

The continuous-time system model is as follows

$$\frac{d}{dt}\bar{\mathbf{q}} = \Omega(\omega)\bar{\mathbf{q}} \tag{4.22}$$

$$\frac{d}{dt}{}^{h}\mathbf{b} = -\alpha^{h}\mathbf{b} + \mathbf{w}_h \tag{4.23}$$

$$\frac{d}{dt}{}^{g}\mathbf{b} = \mathbf{w}_g \tag{4.24}$$

$$\tag{4.25}$$

The state vector is

$$\mathbf{x} = \begin{bmatrix} \bar{\mathbf{q}}^{\mathrm{T}} & {}^{h}\mathbf{b}^{\mathrm{T}} & {}^{g}\mathbf{b}^{\mathrm{T}} \end{bmatrix}^{\mathrm{T}} \tag{4.26}$$

The state propagation developed in [68] is given by

$$\begin{bmatrix} \frac{d}{dt}\hat{\bar{\mathbf{q}}} \\ \frac{d}{dt}\hat{\mathbf{b}}_h \\ \frac{d}{dt}\hat{\mathbf{b}}_g \end{bmatrix} = \underbrace{\begin{bmatrix} -\frac{1}{2}\Xi(\hat{\bar{\mathbf{q}}})\hat{\mathbf{b}}_g \\ -\alpha\bar{\mathbf{b}}_h \\ \mathbf{0}_{3\times1} \end{bmatrix}}_{\mathbf{f}_0} + \underbrace{\begin{bmatrix} \frac{1}{2}\Xi(\hat{\bar{\mathbf{q}}}) \\ \mathbf{0}_{3\times1} \\ \mathbf{0}_{3\times1} \end{bmatrix}}_{\mathbf{f}_1} \omega_m \tag{4.27}$$

where $\Xi(\hat{\bar{\mathbf{q}}})$ is defined as

$$\Xi(\hat{\bar{\mathbf{q}}}) = \begin{bmatrix} q_4\mathbf{I}_{3\times3} + \begin{bmatrix}\mathbf{q}\times\end{bmatrix} \\ -\mathbf{q}^{\mathrm{T}} \end{bmatrix} = \begin{bmatrix} q_4 & -q_3 & q_2 \\ q_3 & q_4 & -q_1 \\ -q_2 & q_1 & q_4 \\ -q_1 & -q_2 & -q_3 \end{bmatrix} = \begin{bmatrix} \bar{\mathbf{s}}_1 & \bar{\mathbf{s}}_2 & \bar{\mathbf{s}}_3 \end{bmatrix} \tag{4.28}$$

and where the angular velocity $\hat{\omega} = \omega_m - {}^{g}\mathbf{b}$ and acts as a control input. The

state propagation is

$$\overbrace{\begin{bmatrix} \hat{\bar{\mathbf{q}}}(k) \\ \begin{bmatrix} {}^{h}\hat{\mathbf{b}}(k) \\ {}^{g}\hat{\mathbf{b}}(k) \end{bmatrix} \end{bmatrix}}^{\mathbf{x}(k)} = \overbrace{\begin{bmatrix} \Phi(k-1) & \\ & \begin{bmatrix} e^{-\alpha\Delta T}\mathbf{I}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{I}_{3\times3} \end{bmatrix} \end{bmatrix}}^{\mathbf{f}(k-1)} \overbrace{\begin{bmatrix} \hat{\bar{\mathbf{q}}}(k-1) \\ \begin{bmatrix} {}^{h}\hat{\mathbf{b}}(k-1) \\ {}^{g}\hat{\mathbf{b}}(k-1) \end{bmatrix} \end{bmatrix}}^{\mathbf{x}(k-1)} + \overbrace{\begin{bmatrix} {}^{q}\mathbf{w}(k-1) \\ \begin{bmatrix} {}^{h}\hat{\mathbf{w}}(k-1) \\ {}^{g}\hat{\mathbf{w}}(k-1) \end{bmatrix} \end{bmatrix}}^{\mathbf{w}(k-1)} \tag{4.29}$$

Let

$$\Phi(k-1) = \cos(|\tilde{\omega}(k-1)|\Delta T/2)\mathbf{I}_{4\times4} + \frac{\sin(|\tilde{\omega}(k-1)|\Delta T/2)}{|\tilde{\omega}(k-1)|\Delta T/2}\Omega(\tilde{\omega}(k-1)) \tag{4.30}$$

and

$$\tilde{\omega}(k-1) = \omega_m(k-1) - {}^{g}\mathbf{b}(k-1) \tag{4.31}$$

The gyro measurement noise is incorporated into the state model via the process noise component ${}^{q}\mathbf{w}(k-1)$,

$${}^{q}\mathbf{w}(k-1) = \frac{\Delta T}{2}\Xi(\hat{\bar{\mathbf{q}}}(k-1)){}^{g}\mathbf{v}(k-1) \tag{4.32}$$

As stated in [68], the process noise covariance matrix $\mathbf{Q}(k-1)$ is

$$\mathbf{Q}(k-1) = \begin{bmatrix} \sigma_g^2(\Delta T/2)^2(\text{trace}(\mathbf{M}_0)\mathbf{I}_{4\times4} - \mathbf{M}_0) & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \sigma_h^2\frac{1-e^{-2\alpha\Delta T}}{3\alpha}\mathbf{I}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & {}^{b}\sigma_g^2\mathbf{I}_{3\times3} \end{bmatrix} \tag{4.33}$$

$$\mathbf{M}_0 = \bar{\mathbf{q}}(k-1)\bar{\mathbf{q}}(k-1)^{\mathrm{T}} + \mathbf{P}^{g}(k-1) \tag{4.34}$$

where $\bar{\mathbf{q}}(k-1)$ is the expectation of the quaternion component within the state vector, and $\bar{\mathbf{q}}(k-1)$ is the covariance matrix of the quaternion component.

The following equations are given for each reference vector components that is required for vector matching as noted in [68]:

$$\begin{bmatrix} \mathbf{a}_m(k) \\ \mathbf{h}_m(k) \end{bmatrix} = \begin{bmatrix} {}^{\mathcal{B}}_{\mathbf{E}}\mathbf{C}(\bar{\mathbf{q}}(k)) & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & {}^{\mathcal{B}}_{\mathbf{E}}\mathbf{C}(\bar{\mathbf{q}}(k)) \end{bmatrix} \begin{bmatrix} \mathbf{g} \\ \mathbf{h} + {}^{h}\mathbf{b}(k) \end{bmatrix} + \begin{bmatrix} {}^{a}\mathbf{v}(k) \\ {}^{m}\mathbf{v}(k) \end{bmatrix} \tag{4.35}$$

where again the sub-script m and a indicate which equations refer to the magnetometer and accelerometer respectively. The measurement noise covariance matrix referenced in the EKF Eq. (4.5) is

$$\mathbf{R}(k) = \begin{bmatrix} \sigma_a^2\mathbf{I}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \sigma_m^2\mathbf{I}_{3\times3} \end{bmatrix} \tag{4.36}$$

This subsection briefly repeated the work for an EKF AHRS primarily developed in [68]. In this subsection we introduced the general equations for the EKF, outlined how quaternions can be used to describe orientation, introduced the kinematic equations for rotation, and showed how a discrete state space model could be propagated.

4.1.2 TRIAD AHRS

The TRIAD AHRS algorithm is robust and avoids the issue of gyro drift entirely simply by not using gyros. Note that this makes TRIAD more suitable as an aiding or low-bandwidth AHRS choice. It is a method that uses two aiding reference vectors to compute a rotation direction cosine matrix (DCM) [72] [21]. The TRIAD algorithm as presented in [21] is outlined here for completeness.

First, we consider two reference vectors, $\mathbf{v}_1$ and $\mathbf{v}_2$ in the local tangent plane frame of reference, such that their elements correspond to the x, y, and z axes. In the case of gravity this would imply that $\mathbf{v}_1 = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}$. Let the observation unit vectors be $\mathbf{w}_1$ and $\mathbf{w}_2$ in the body-fixed reference frame. For the ASV, these are measured from a tri-axis accelerometer and magnetometer respectively. Note that some other examples include sun or lunar-tracking sensors or cameras. Now assume that there is a DCM $\mathbf{A}$ such that

$$\mathbf{A}\mathbf{v}_1 = \mathbf{w}_1, \quad \mathbf{A}\mathbf{v}_2 = \mathbf{w}_2 \tag{4.37}$$

Let

$$\mathbf{M}_o = \begin{bmatrix} o_1 & o_2 & o_3 \end{bmatrix} \tag{4.38}$$

and

$$\mathbf{M}_r = \begin{bmatrix} r_1 & r_2 & r_3 \end{bmatrix} \tag{4.39}$$

where

$$o_1 = \mathbf{w}_1 \tag{4.40}$$

$$o_2 = \frac{\mathbf{w}_1 \times \mathbf{w}_2}{\|\mathbf{w}_1 \times \mathbf{w}_2\|} \tag{4.41}$$

$$o_3 = \frac{\mathbf{w}_1 \times (\mathbf{w}_1 \times \mathbf{w}_2)}{\|\mathbf{w}_1 \times \mathbf{w}_2\|} \tag{4.42}$$

$$r_1 = \mathbf{v}_1 \tag{4.43}$$

$$r_2 = \frac{\mathbf{v}_1 \times \mathbf{v}_2}{\|\mathbf{v}_1 \times \mathbf{v}_2\|} \tag{4.44}$$

$$r_3 = \frac{\mathbf{v}_1 \times (\mathbf{v}_1 \times \mathbf{v}_2)}{\|\mathbf{v}_1 \times \mathbf{v}_2\|} \tag{4.45}$$

$$\tag{4.46}$$

Now the estimate of the DCM, $\mathbf{A}$ is

$$\hat{\mathbf{A}} = \mathbf{M}_o \mathbf{M}_r^{\mathrm{T}} \tag{4.47}$$

Finally, the Euler angles can be extracted with

$$\psi = \text{atan2}(A_{23,33}) \tag{4.48}$$

$$\theta = \text{asin}(-A_{13}) \tag{4.49}$$

$$\phi = \text{atan2}(A_{12,11}) \tag{4.50}$$

$$\tag{4.51}$$

Caveats of TRIAD

One of the key drawbacks it possess is that it is highly dependent on the variance σ_w associate with the measured aiding reference vectors. On a different

note, many micro-electrical-mechanical (MEMS) tri-axis accelerometer, magnetometers, and gyros sensor-heads (also variably called inertial measurement units or IMUs) have different update rates for each specific sensor. Typically a gyro has a faster update rate (or bandwidth) than both the accelerometers and magnetometers. This is not a problem by itself, but depending on the application and if the vehicle is rotating fast enough, the update rate from a TRIAD AHRS may be to slow. If the speed requirements are very high, then an AHRS that uses gyros with a fast update rate would be necessary; TRIAD may not be the best option. Note that there are also methods to propagate the attitude as a state between TRIAD updates, such as in [21], which may also be useful for high-dynamics vehicles.

4.1.3 CF AHRS

Complementary filters have certain advantages and disadvantages compared to other AHRS algorithms. They are generally less computationally intensive, but at the cost of accuracy and slower convergence rates. Fig. 4.1 shows a passive discrete complementary filter, introduced in [50]. It uses a skew anti-symmetric projection operator $\pi_a()$ to project the rotation matrix error $\tilde{R}$ onto the skew matrix $(\hat{R}\Omega_y)_\times$ formed from the rotation matrix estimate $\hat{R}$ and the gyro angular rate vector Ω_y. The rotation matrix error is $\tilde{R} = \hat{R}^{\mathrm{T}} R_y$, where R_y is the rotation matrix between the inertial aiding vector and the aiding vector as measured in the body-fixed reference frame.

Assume there exists some unit norm vector that represents a normalized sensor measurement, such as an accelerometer or magnetometer vector measured in the body-fixed frame:

$$\boldsymbol{v}^b = \begin{bmatrix} v_1 & v_2 & v_3 \end{bmatrix}^{\mathrm{T}} \tag{4.52}$$

R_y may found using the following equation:

$$R_y \triangleq \begin{bmatrix} v_1^2 + (v_2^2 + v_3^2)\mathrm{c}\phi & v_1 v_2(1-\mathrm{c}\phi) - v_3\mathrm{s}\phi & v_1 v_3(1-\mathrm{c}\phi) + v_2\mathrm{s}\phi \\ v_1 v_2(1-\mathrm{c}\phi) + v_3\mathrm{s}\phi & v_2^2 + (v_3^2 + v_1^2)\mathrm{c}\phi & v_2 v_3(1-\mathrm{c}\phi) - v_1\mathrm{s}\phi \\ v_3 v_1(1-\mathrm{c}\phi) - v_2\mathrm{s}\phi & v_2 v_3(1-\mathrm{c}\phi) + v_1\mathrm{s}\phi & v_3^2 + (v_1^2 + v_2^2)\mathrm{c}\phi \end{bmatrix} \tag{4.53}$$

where ϕ is the angle between the inertial and body-fixed reference vector, and can be found using the definition of the dot product:

$$\phi = \frac{\cos^{-1}(\boldsymbol{m}_i \cdot \boldsymbol{m}_b)}{||\boldsymbol{m}_i||||\boldsymbol{m}_b||} \tag{4.54}$$

where m_i and m_b are analogous to v_1 and v_2 respectively.

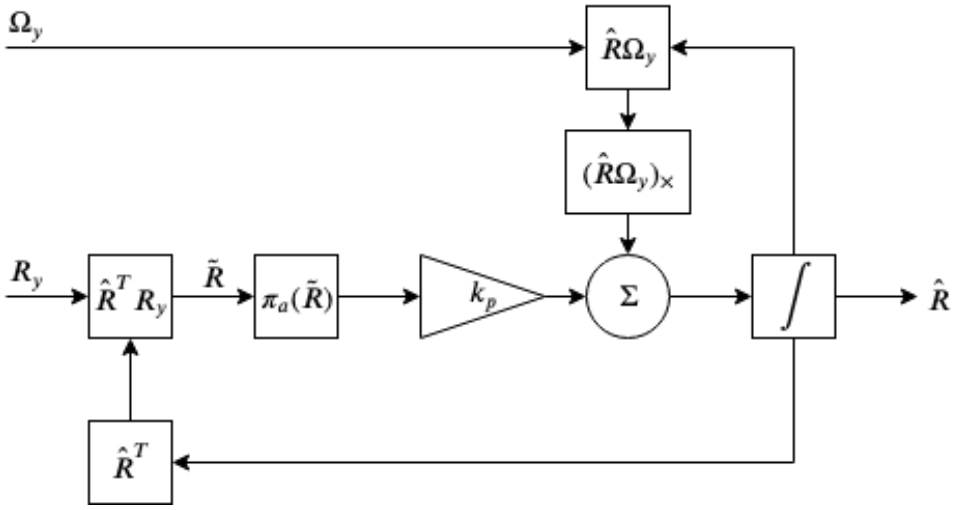


Figure 4.1: A passive discrete CF, as introduced in [50]

Fig. 4.2 shows the full CF with accelerometer and magnetometer feedback. Note that there are now two gains $k_{p,a}$ and $k_{p,m}$ that can be tuned depending on the level of noise from the aiding vector sensors. Generally, the magnetometer gain is low because there can be substantial noise for vehicles with electric motors, batteries, actuators, and other electronic components.

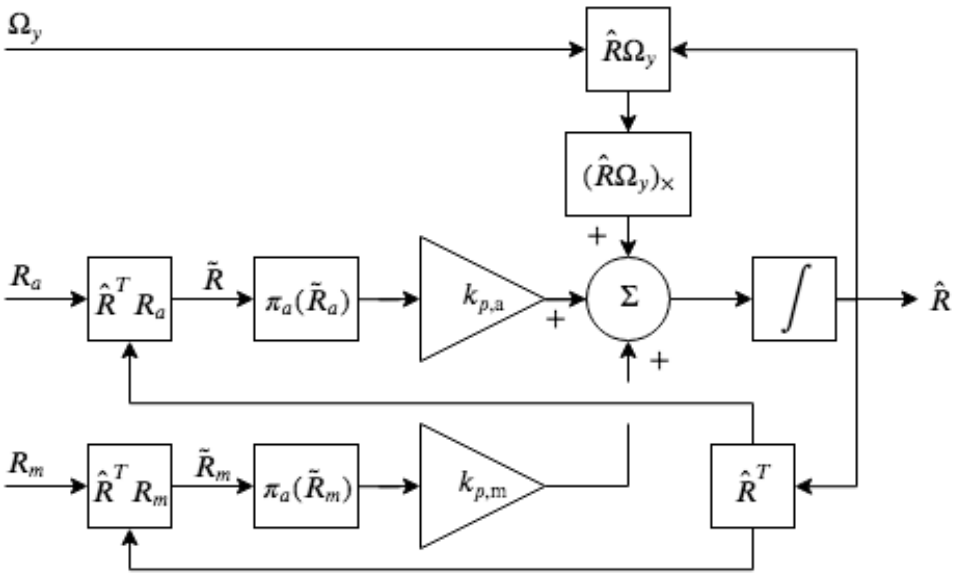


Figure 4.2: A full CF with both accelerometer and magnetometer feedback.

It should be noted that there are many other versions of the complementary filter. While the above versions were implemented and tested, the final version of the CF used in this work used quaternions. It is detailed later in Section 4.1.4.

4.1.4 Comparison of EKF, TRIAD, and CF

A simulation was created to compare the EKF, TRIAD, and CF ARHS algorithms. This subsection briefly discusses how that simulation was created and then the results of that simulation. Fig. 4.3 shows the simulated true and measured angular rates of a generic vehicle or object. The attitude orientation Euler angles and the true inertial aiding reference vectors were generated by integrating the true angular rates and applying rotations to the inertial aiding vectors. The rates were integrated using quaternions, and the aiding vectors were rotated into the body frame based on the resulting Euler angles. Quaternions were chosen specifically because they provide an elegant way to ensure that the body angular rates

can be integrated while avoiding rotational singularities. Additionally, they are well-suited for embedded applications because of their relatively low computation cost.

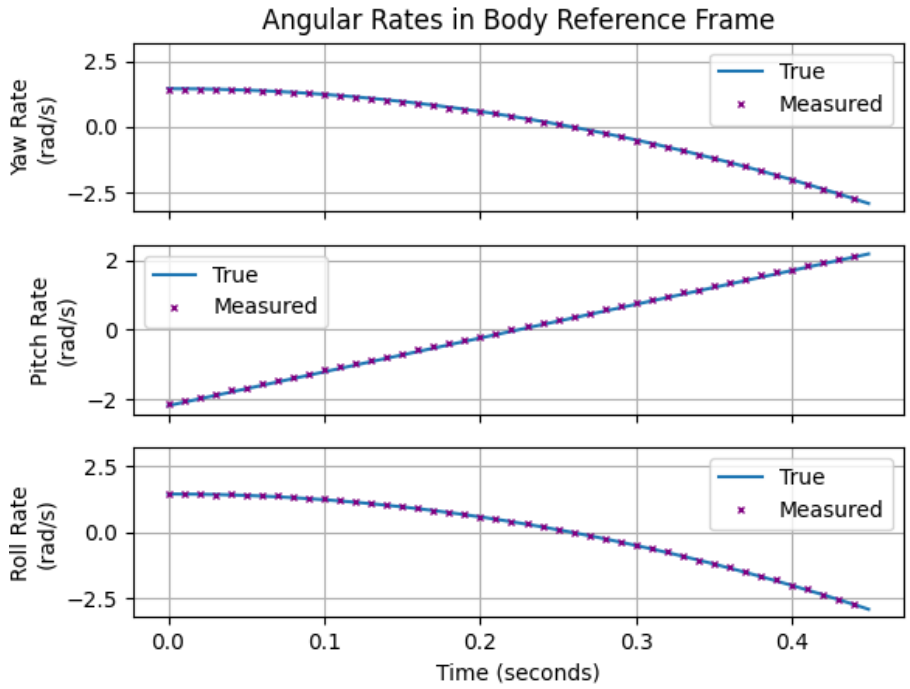


Figure 4.3: True and measured rotational rates in the body reference frame, with the standard deviation of noise from a normal distribution of $\sigma_{\text{gyro}} = 0.01$

Since we are dealing with a tri-axis accelerometer, gyroscope, and magnetometer it is helpful to visualize the orthogonal attitude basis vectors $\hat{e}_0$, $\hat{e}_1$, and $\hat{e}_2$, such that the matrix

$$\begin{bmatrix} \hat{e}_0 & \hat{e}_1 & \hat{e}_2 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \mathbf{I}_{3\times 3} \tag{4.55}$$

Fig. 4.4 shows the integration of the true body angular rates, previously shown in

Fig. 4.3.

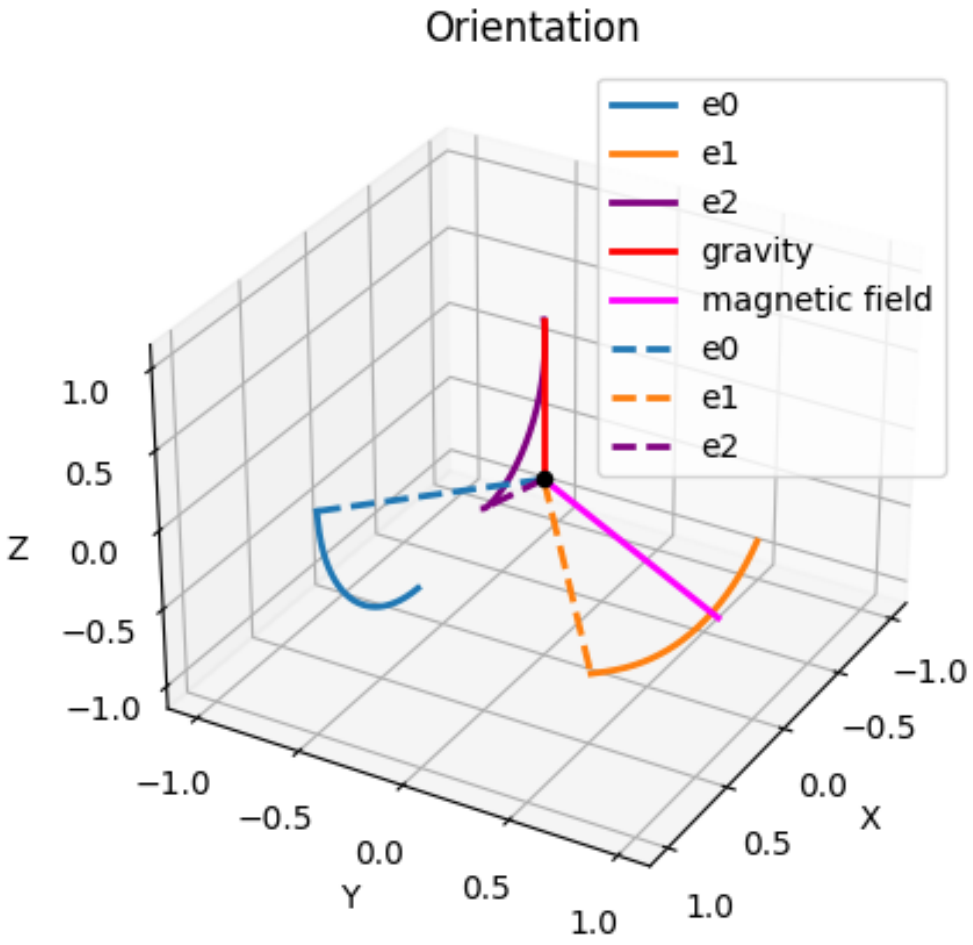


Figure 4.4: Visualization of the rotation of the orthogonal basis vectors. Also shown are the inertial aiding vectors gravity (red), and the magnetic field vector (magenta).

Rate Quaternion Integration

A popular and computationally efficient method for angular rate integration is quaternion integration. This is partly due to the fact that quaternions avoid the transcendental functions that compromise the integration of Euler angular rates. There have been many use-cases of quaternion integration. For instance, an

adaptive unscented Kalman filter with quaternion-based orientation estimation is presented in [64], and similar approaches for quaternion integration are developed in [67] and [5]. The integration method is repeated here for completeness; define a vector of angular rates in the body-frame as

$$\omega^{\mathrm{b}} = \begin{bmatrix} p & q & r \end{bmatrix}^{\mathrm{T}} \tag{4.56}$$

where p, q and r represent the roll, pitch, and yaw body angular rates about the x, y, and z body-axes respectively. The body angular rates can be expressed as part of a quaternion-based differential equation:

$$\dot{\mathbf{q}} = \frac{1}{2}\Omega(\omega^{\mathrm{b}})\mathbf{q} \tag{4.57}$$

with $\mathbf{q}$ representing the attitude quaternion, and where

$$\Omega(\omega^{\mathrm{b}}) = \begin{bmatrix} \overbrace{\begin{bmatrix} 0 & r & -q \\ -r & 0 & p \\ q & -p & 0 \end{bmatrix}}^{\left[\omega^{b}_{\times}\right]} & p \\ q \\ r \\ -p & -q & -r & 0 \end{bmatrix} = \begin{bmatrix} -[\omega^{\mathrm{b}}_{\times}] & \omega^{\mathrm{b}} \\ -\omega^{\mathrm{b}^{\mathrm{T}}} & 0 \end{bmatrix} \tag{4.58}$$

For constant ω (see [5]) we see that solution for Eq. (4.57) is

$$\mathbf{q}(t) = \exp\Big(t\frac{1}{2}\Omega(\omega^{b})\Big)\mathbf{q}(0) \tag{4.59}$$

Note that the skew operator can also assume the following notation,

$$\lceil \omega^b \rceil = \left[\omega^b_\times\right] = \begin{bmatrix} 0 & r & -q \\ -r & 0 & p \\ q & -p & 0 \end{bmatrix} \tag{4.60}$$

The change in angle for a given time-step, Δt may be expressed as

$$\Delta\theta = [\omega^b]\Delta t \tag{4.61}$$

It should be noted that a key assumption here is that for small Δt the angular velocity is assumed to be constant. In reality that is not true, but it is generally close to a constant rate and is useful to assume as will be shown later. The assumption that the angular rate is constant for small time-steps allows for the discretization of Eq. (4.57), see [21]. The solution for the discrete-time model is then

$$\mathbf{q}_{k+1} = M(\Delta\theta)\mathbf{q}_k \tag{4.62}$$

where (from [64]) we have,

$$M(\Delta\theta) = \cos\left(\left|\left|\frac{\Delta\theta}{2}\right|\right|\right)\mathbf{I}_{4\times 4} + \frac{\sin(||\Delta\theta/2||)}{||\Delta\theta||}\Omega(\Delta\theta) \tag{4.63}$$

Note the similarity to Eq. (4.15). It is the same matrix exponential formula. The Euler angles are extracted using the following method by [46] as

$$\phi = \tan^{-1}\left(\frac{2q_1q_3 + 2q_0q_1}{2q_0^2 + 2q_3^2 - 1}\right) \tag{4.64}$$

$$\theta = \sin^{-1}(2q_1q_3 + 2q_0q_2) \tag{4.65}$$

$$\psi = \tan^{-1}\left(\frac{2q_1q_2 + 2q_0q_3}{2q_0^2 + q_1^2 - 1}\right) \tag{4.66}$$

Next, the aiding reference vectors in the body reference frame are generated. To do this, a rotation direction cosine (DCM) rotation matrix can be formed from the true attitude quaternion q, using another method from [46].

$$\mathbf{R}(\mathbf{q}) = \begin{bmatrix} 2q_0^2 - 1 + 2q_1^2 & 2q_1q_2 + 2q_0q_3 & 2q_1q_3 - 2q_0q_2 \\ 2q_1q_2 - 2q_0q_3 & 2q_0^2 - 1 + 2q_2^2 & 2q_2q_3 + 2q_0q_1 \\ 2q_1q_3 + 2q_0q_2 & 2q_2q_3 - 2q_0q_1 & 2q_0^2 - 1 + 2q_3^2 \end{bmatrix} \tag{4.67}$$

The inertial gravity vector $\mathbf{v}_g^i$ and inertial magnetic field vectors $\mathbf{v}_m^i$ can be rotated in to the body reference frame of the vehicle. They are both 3×1 normalized vectors.

$$\mathbf{v}_g^b = \mathbf{R}^{\mathrm{T}}\mathbf{v}_g^i \tag{4.68}$$

In summary, a sequence of Euler angles ψ, θ, ϕ is calculated from a sequence of rotation rates $\mathbf{r}, \mathbf{q}, \mathbf{p}$ using Eq. (4.62), and the aiding vectors in the inertial reference frame of the vehicle are calculated using Eqs. (4.64) - (4.68). One caveat is that using quaternions for the Complementary Filter with the method outlined above doesn't provided a tremendous advantage over using DCMs, since we still use DCMs in Eq. (4.67) to rotate the inertial-frame accelerometer and magnetometer vectors into the body-frame. However, if the vectors are rotated using

the quaternion rotation operator then DCMs can be avoided until Euler angles are required to be extracted from the estimated attitude quaternion. Another option instead of using quaternions is to propogate the attitude rotation DCM $\hat{R}$ using the matrix exponential operator $\exp(\omega)$. This can be approximated by a Taylor expansion, or similar approximations using the Rodrigues' formula [61] and Eq. (4.60).

$$\exp(\omega) = I + \frac{\sin(||\omega||)}{||\omega||}\lceil\omega\rceil + \frac{\sin^2(\frac{||\omega||}{2})}{\frac{||\omega||^2}{2}}\lceil\omega\rceil^2 \tag{4.69}$$

$$\hat{R}_{k+1} = \exp(\omega)\hat{R}_k \tag{4.70}$$

Simulation Noise

Noise was added to both the true angular rates and the true aiding vectors to better simulate a measurement from a tri-axial gyros, accelerometers, and magnetometers. A Gauss-Markov process was used along with a scalar offset to simulate gyro drift; for example, the noisy angular yaw rate $\tilde{\mathbf{r}}$ is based on the variance of the gyro σ^2_{gyro}, a time constant β, noise from a normal distribution $\epsilon(t_k) \sim \mathcal{N}(0, \sigma_{\text{gyro}})$, and a scalar offset a_0.

$$\tilde{\mathbf{r}}(t_k) = \sigma^2_{\text{gyro}} e^{-\beta|t_k|}\tilde{\mathbf{r}}(t_{k-1}) + \epsilon(t_k) + \mathbf{r}(t_k) + a_0 \tag{4.71}$$

Where $\mathbf{r}(t_k)$ is the true angular rate and ϵ is an additional noise term. Similarly, noise from a normal distribution was added to the aiding vectors

$$\tilde{\mathbf{v}}^b_g(t_k) = \mathbf{v}^b_g(t_k) + w(t_k) \tag{4.72}$$

$$\tilde{\mathbf{v}}_m^b(t_k) = \mathbf{v}_m^b(t_k) + w(t_k) \tag{4.73}$$

where

$$w(t_k) \sim \mathcal{N}(0, \sigma_w) \tag{4.74}$$

Simulation Results

Figure 4.5 shows the true aiding reference vectors in the body reference frame, based on the quaternion integration of Eq. (4.62) of the true angular rates in the body frame. The reason for this choice is based on the computation speed up due to the approximations of the integration technique saving computation cycles. Moreover, it is a common implementation on embedded systems for open source autopilots. Again, the resulting rotation matrix transpose in Eq. (4.68) is used to rotate the inertial reference vectors in to the body frame. Upon adding noise, and offsets to the rates and aiding vectors, a realistic tri-axis gyroscope, accelerometer, and magnetometer set of signals is created.

Figs. 4.6 and 4.7 show the attitude estimates of the EKF, TRIAD, and CF ARHS algorithms. Each AHRS outputs the estimated Euler angles Ψ, Θ, Φ, that represent the orientation of the vehicle or object in the inertial reference frame. The EKF output is shown in green, the TRIAD output is shown in orange, and the CF output is shown in purple. The true orientation is shown by the solid blue line.

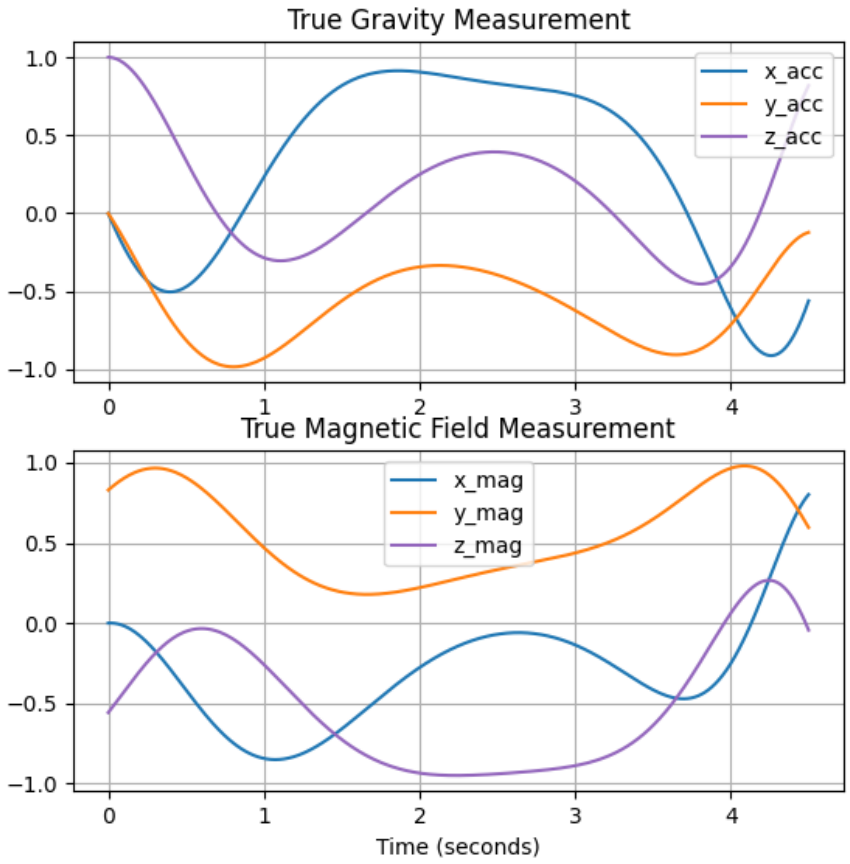


Figure 4.5: The true aiding reference vectors in the body reference frame

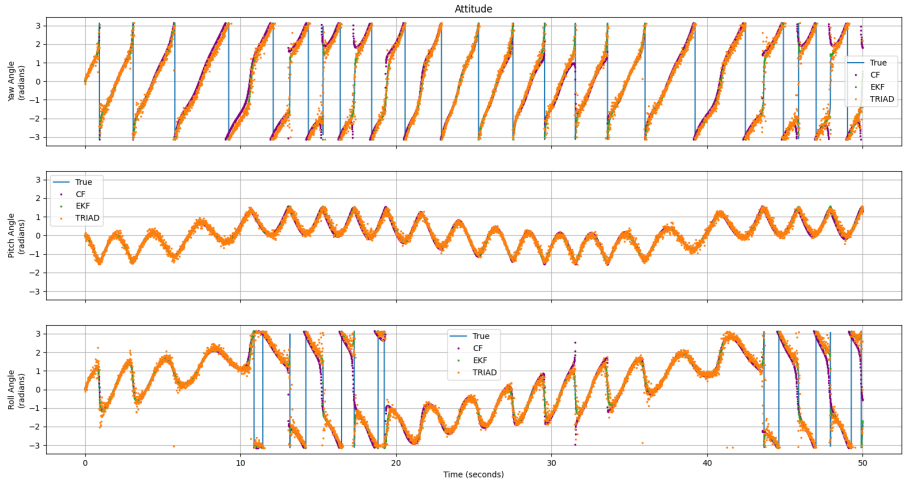


Figure 4.6: A 50 second simulation showing attitude estimates of EKF, TRIAD, and CF for the orientation Euler angles.

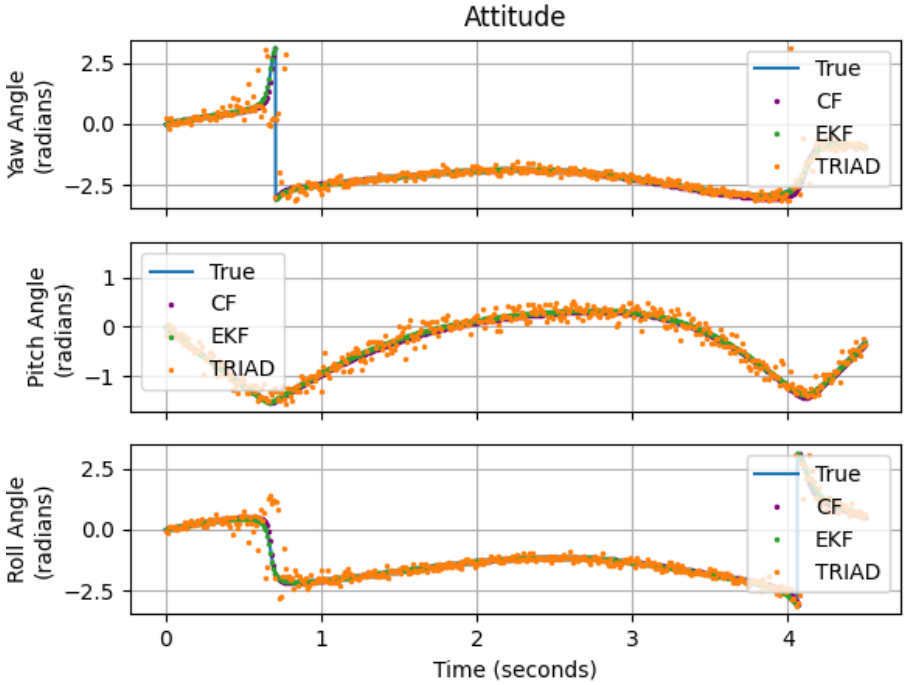


Figure 4.7: A 5 second simulation of the attitude estimates of EKF, TRIAD, and CF for the orientation Euler angles.

Performance

Two key metrics when comparing AHRS algorithms are error and the computation time. The term error is a little general, but can be expressed in a few ways, including mean absolute error (MAE), mean, and standard deviation. Figures 4.8 and 4.9 show the error histograms and errors signals respectively, for each axis of rotation from each AHRS algorithm. The mean computation times for the CF, EKF, and TRIAD were approximately 0.168ms, 0.488ms, and 0.202ms respectively. Table 4.1 shows the mean and standard deviation corresponding to the results in Fig. 4.8.

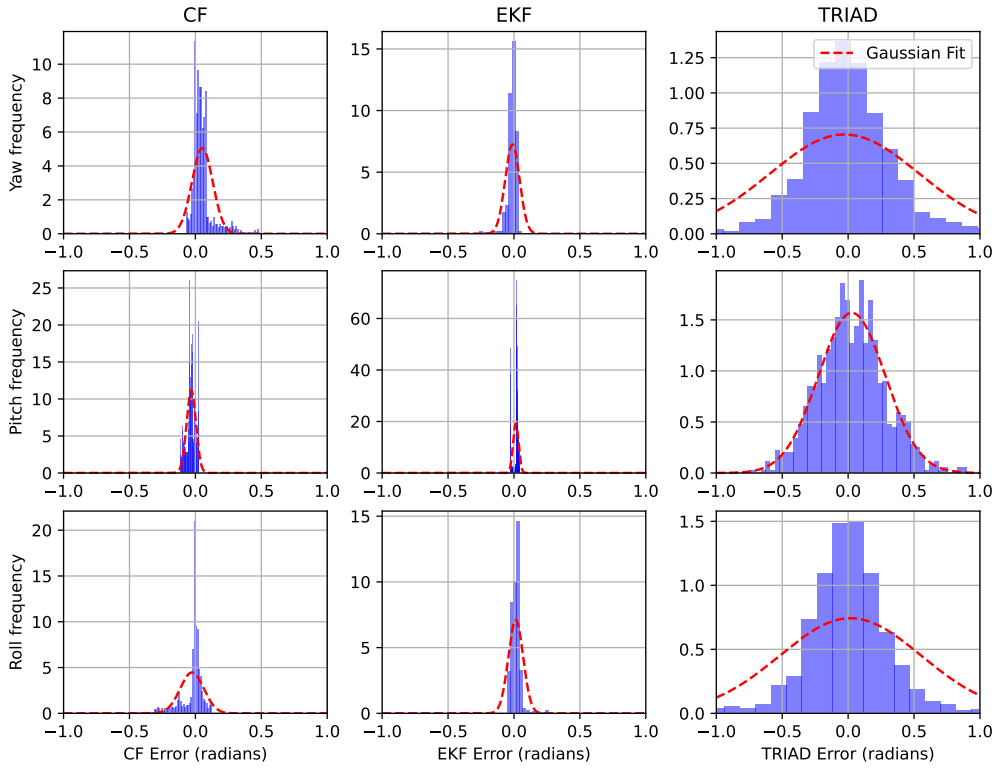


Figure 4.8: The error histograms for each axis of rotation from the different AHRS algorithms. Note that the CF did not incorporate gyro bias compensation.

Table 4.1: Attitude estimation error comparison with mean and standard deviation

	μ (radians)	σ (radians)
CF Yaw	0.052	0.079
CF Pitch	-0.032	0.036
CF Roll	-0.026	0.089
EKF Yaw	0.055	0.055
EKF Pitch	0.014	0.021
EKF Roll	0.016	0.056
TRIAD Yaw	-0.027	0.566
TRIAD Pitch	0.028	0.254
TRIAD Roll	0.018	0.537

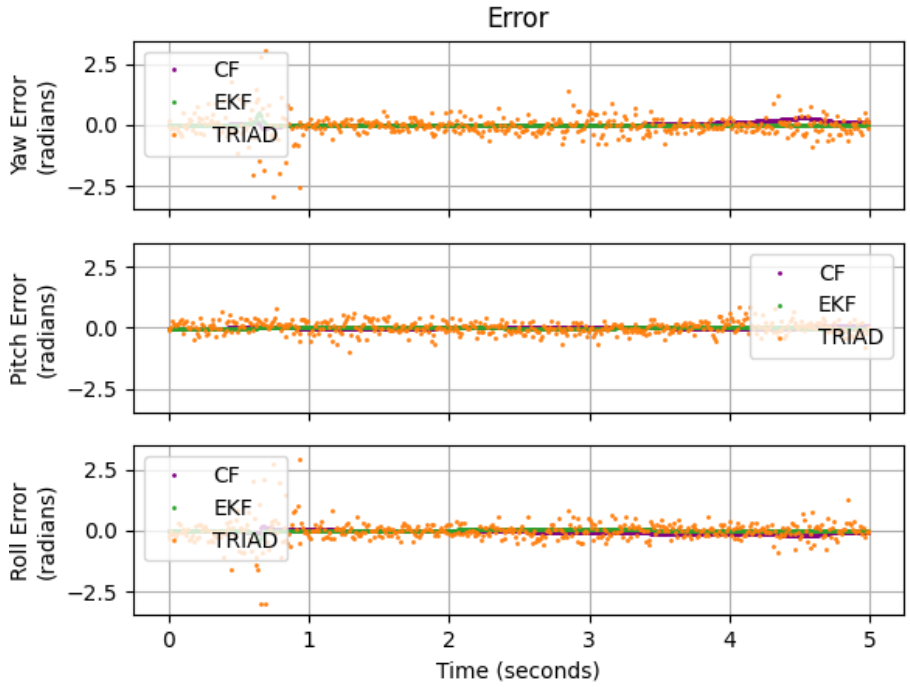


Figure 4.9: A 5 second window of the error signals for each axis of rotation from the different AHRS algorithms

Computation time (flop count) is calculated based on the time it takes for an ARHS algorithm to execute per measurement set. This can vary depending on the computer or embedded system that is running the ARHS. For example,

a Raspberry Pi or similar computer running a non-real-time operating system might vary on the time it takes to execute an AHRS attitude estimate, where as an embedded system with hard-timing, based directly on a crystal oscillator with a phase-locked loop (PLL), will be almost, if not exactly consistent in time based on the number of cycles each operation takes. A non-real-time computer is used to run the simulation. Fig. 4.10 shows the computation time of the AHRS algorithm at each time step of the simulation.

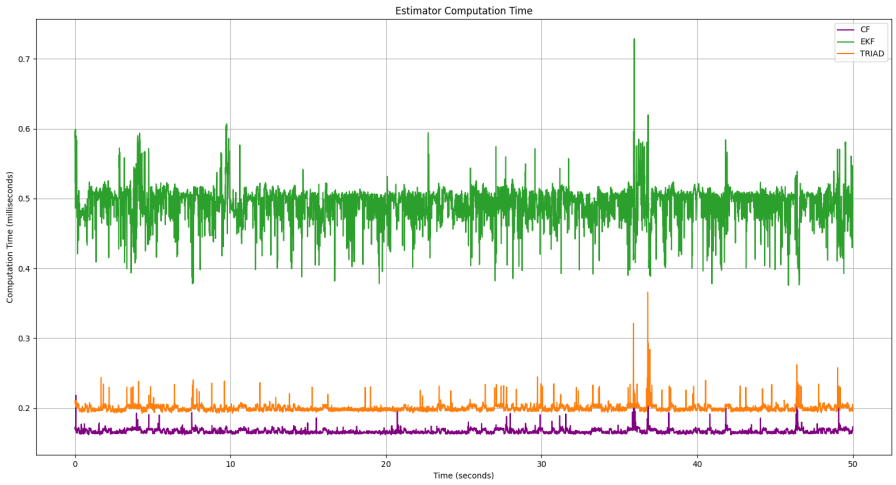


Figure 4.10: Computation time for each AHRS algorithm at each time step of the simulation.

As mentioned earlier, the MAE can help in the comparison between ARHS algorithms. For example, the MAE for an AHRS algorithm's roll angle estimate signal $\hat{\phi}$ can be defined as follows.

$$e_\phi = \sum_{i=0}^{m-1} |\hat{\phi}(i) - \phi(i)| \tag{4.75}$$

The total MAE for an AHRS can then be defined as the sum of each MAE value for all axes of rotation, $e_\Sigma = e_\psi + e_\theta + e_\phi$. Figure 4.11 shows the MAE versus

mean computation time

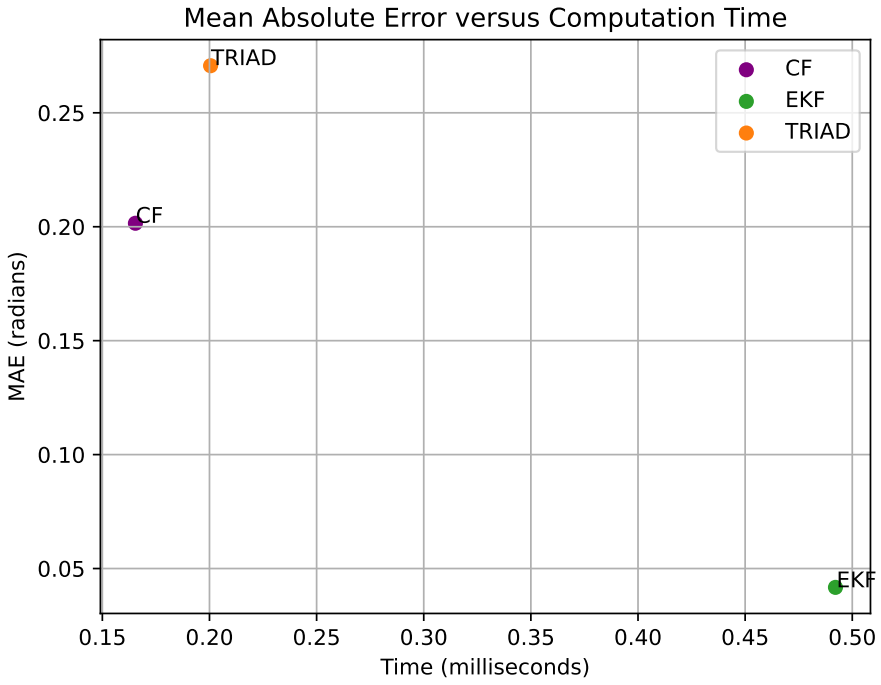


Figure 4.11: MAE versus mean computation time for each AHRS algorithm.

Discussion

In general, it was found that the CF was the fastest to compute, and produced an intermediate MAE, followed by the TRIAD with its additional computation time cost. However, the TRIAD algorithm specifically scales along the vertical MAE axis proportional to the accelerometer and magnetic field noise variance $\sigma^2_{w,\text{acc}}$ and $\sigma^2_{w,\text{mag}}$. Realistic variance values were chosen based on real sensor data sheets. The magnetic field can be significantly noisy in proximity to electric motors and similar noise sources. This is a key drawback to using TRIAD in practice. In ideal and potentially unrealistic environments, where the accelerometer and magnetometer have very little noise, the TRIAD method can outperform both

the CF and EKF in terms of MAE. That would require very small standard deviations of noise that may not be physically feasible. Unsurprisingly, the EKF outperforms the other AHRS algorithms in MAE, but at the cost of being the slowest in terms of computational time. It does possess other drawbacks however, including the fact that the process noise must be known *a priori*. This can be challenging, because even if data is collected and a process noise is calculated, it may not be ergodic and thus using it to provide future process noise is problematic. In which case, the EKF may change in performance over time and can possibly diverge (though in practice, EKFs work quite).

4.1.5 Validation of Complementary Filter-Based AHRS

In this work, an AHRS is designed using a CF based on work by [50] and [20] to provide the orientation estimates. The ASV's orientation estimates can be calculated faster than every 0.05-seconds, or 20Hz. This is useful information to have while path following. For instance, the path angle error can be calculated by knowing the vehicles' heading, or yaw angle (which is part of the orientation estimate) with respect to the local tangent plane and the path angle associated with the closest point on the path to the vehicle. Eq 7.18 in sec 7.2.2 is meant to use the heading angle estimate from the CF-based ARHS. The other option is to use the course-over-ground (COG) measurements from the GPS unit as the heading angle of the vehicle. This is susceptible to error if the ASV drifts in the current, where they direction of the true vehicle heading is not necessarily the direction of movement (which the COG measurement is based on). The potential to fuse the COG angle measurement from the vehicle's GPS unit with the heading angle estimate may be explored, but for now the yaw angle from the CF-based AHRS is used.

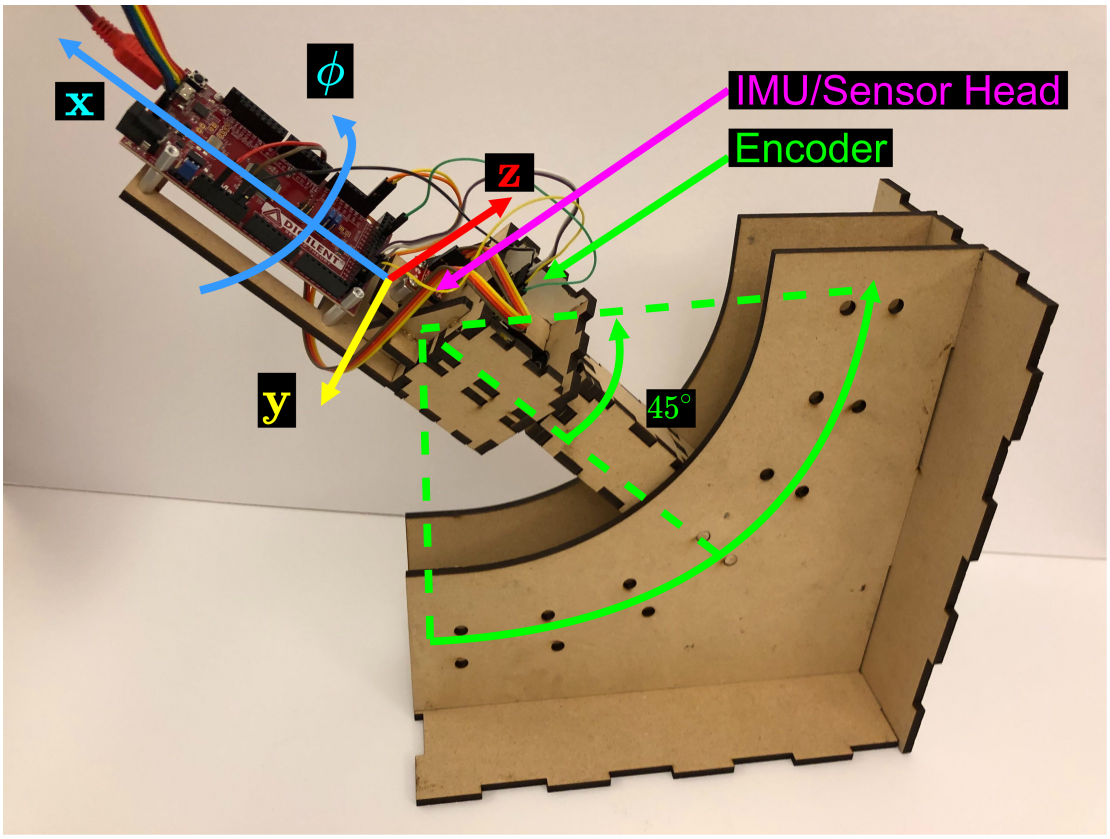


Figure 4.12: Experimental low-cost AHRS testing and validation apparatus with the body-fixed axes labeled.

4.1.6 Validation Apparatus

The CF-based AHRS is validated using a custom validation apparatus outlined in the author's previous work, [79]. The validation apparatus used an encoder for comparison with the orientation estimates from the CF in each axis, respectively. The validation apparatus was designed to be flexible, low-cost, and easy to construct using readily available materials. While a full three-axis truth measurement would be ideal, it was decided that free rotation along a single axis would be much easier to construct and would allow more flexibility. Future work will determine if a full three-axis validation apparatus can be cost-effectively constructed. The high accuracy external truth measurement is an encoder (the Austria Microsys-

tems AS5047D) which has an absolute resolution of approximately 0.022°. In order to increase flexibility, the apparatus can be set at various angles from 0 to 90 such that the aiding sensors (magnetometers and accelerometers) can have varying influence on the sensor fusion. Orientation estimates for each axes are compared with the true rotation angle, as measured by the encoder. Rather then compare angular rates, as one would with a rate-table, the angular position is compared. The apparatus top holds the microcontroller (Microchip PIC32), sensor-head (Invensense MPU9250), and the encoder (Austria Microsystems AS5047D) relative to the column, and spins freely. This allows the apparatus top to make multiple revolutions without wrapping wires around the main column.

The sensor-head can be mounted in three different positions on the apparatus top, such that each cardinal body axis of rotation can be concentric with the axis of rotation of the apparatus independently. The sensor-head measures angular rates using gyroscopes, specific force ($\mathbf{a} - \mathbf{g}$) using accelerometers, and the intensity of Earth's magnetic field using magnetometers in each of the three body-fixed axes. When the top of the apparatus rotates, the encoder measures the angle of rotation. The top can be rotated by hand or, with a simple attachment, using a servo. As the true rotation is measured by the encoder, the source of rotation is arbitrary, so long as the rotation rate does not exceed the gyro speed and the upper limit of the encoder. The results in the paper show rotations generated by hand. The design choice to exclude a servo to generate rotations was made to further reduce the cost of the apparatus. The amount of rotation of the top is limited only by the length of wires to the micro-controller. A USB connection is required to record measurements from the on-board sensors. All electronic components are placed in the top of the apparatus for less restricted rotations; the only connection back from the apparatus top is the USB cable itself.

Development of AHRS for smaller vehicles with low SWaP cannot always be validated with expensive external truth measurement equipment, often due to cost restrictions. Depending on the application, SWaP requirements, and budget requirements, commercial-off-the-shelf (COTS) AHRS may not be available. This section outlines a low-cost solution for validating an AHRS that uses a CF for sensor fusion, and a low-cost sensor-head consisting of a three-axis MEMs accelerometer, gyroscope, and magnetometer. The proposed method is based on a low-cost prototype validation apparatus, shown in Fig. 4.12, that uses a simple encoder as a truth measurement of a single axis of rotation to compare with the integrated angular rates estimated by the CF. The total cost of the apparatus is less than $100.

The arc of mounting holes permits the apparatus to be configured to test varying magnitudes of the force due to gravity and Earth's magnetic field. This allows for the isolation of the inertial vectors to observe the behavior of the AHRS with one aiding vector, the other, or both with varying magnitudes. For example, if the axis of rotation is configured such that it is perpendicular to the the horizontal plane, there is no contribution of yaw feedback from the gravity vector measurement, because the axis of rotation is co-linear with the gravity vector. The arc of the apparatus and the sensor-head in the top is situated specifically so that the center of the sensor-head circuit remains in the same space to mitigate the possible change in magnetic field during reconfiguration of apparatus angle. This is important as it permits the apparatus to be used in electrically noisy environments (e.g. indoors near a computer) yet maintain a consistent magnetic vector measurement. This alleviates the need to use a specially designed testing environment, which can add to development costs. The arc design uses more material intentionally to add more weight to the base of the structure and increase stability.

The larger flat surfaces of the of the apparatus allows for simple mounting to various surfaces. The material used for construction is medium-density fiberboard (MDF). This material is chosen for its relative strength, low-cost, and light weight. The MDF was cut using a laser cutter, and although owning such a machine is expensive, there are myriad locations that allow individuals to send CAD files to have cut. The CAD files for the apparatus allow for easy material width adjustment using building-equation relationships that automatically resize the rest of the structure.

The validation apparatus is comprised of only twenty-eight parts, most of which are duplicates to simplify assembly. The top of the apparatus can be easily configured to mount different kinds of micrcontrollers, sensor-heads, and encoders. Varying sensor-heads and other components means that other types of sensor fusion algorithms that rely on different kinds of measurements can be tested. For example, using sun-sensors such as in [1], instead of gyros.

In addition to simple mounting, the large surfaces and extra space of the apparatus may be used to hold batteries if logging data outside, or in locations with limited power.

4.1.7 Apparatus Results

One of the first steps before using the apparatus for testing was to calibrate the acclerometers and magnetometers. There are many different ways to do this, but here the iterative calibration method described by [24] is used. Fig. 4.13 shows the pre and post calibration of the normalized measurement vectors of the tri-axis magnetometer measurements. It is useful to normalize these vectors by dividing each element by the magnitude of the vector. Ideally, such a sensor would provide this by default, but due to the constraints of manufacturing precision for low cost

sensors, a calibration is required and subsequently performed. This is discussed in further detail in [79].

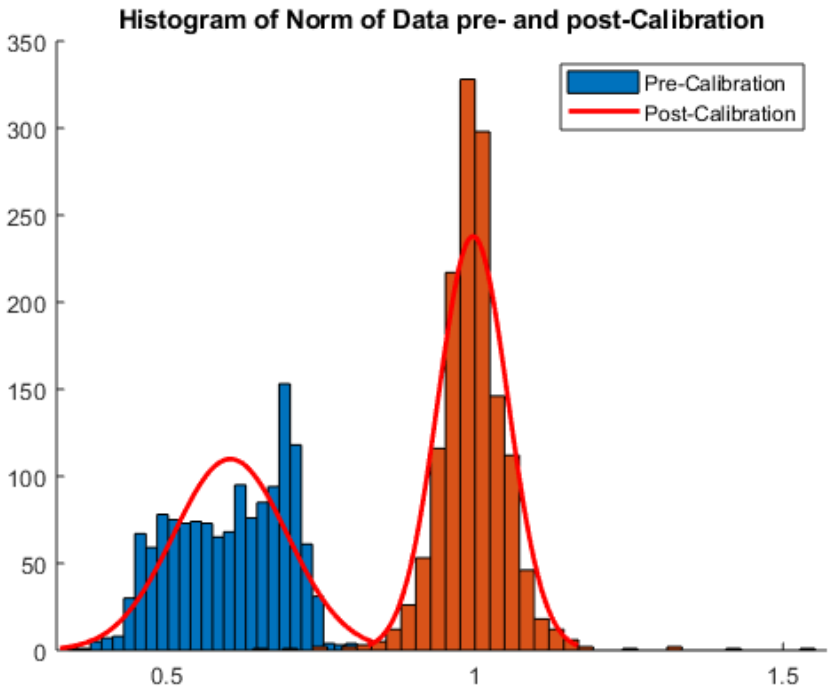


Figure 4.13: A historgram of the magnetic field measurements before and after calibration

The CF-based AHRS described in subsection 4.1.3 was implemented in MATLAB and then later in C code. Fig. 4.14 shows the roll angle estimated by the CF compared to the encoder angle measured about the same axis. Clearly there is very good agreement.

Further tests at different apparatus angles allowed for isolation of the aiding vectors as can be seen in Fig. 4.15. It shows the apparatus positioned at 90°, such that the axis of rotation was normal to the horizontal plane and parallel with the gravity vector. As expected, the roll angle signal from the accelerometer alone was zero. This ability to isolate aiding vectors was useful for tuning the CF and further debugging.

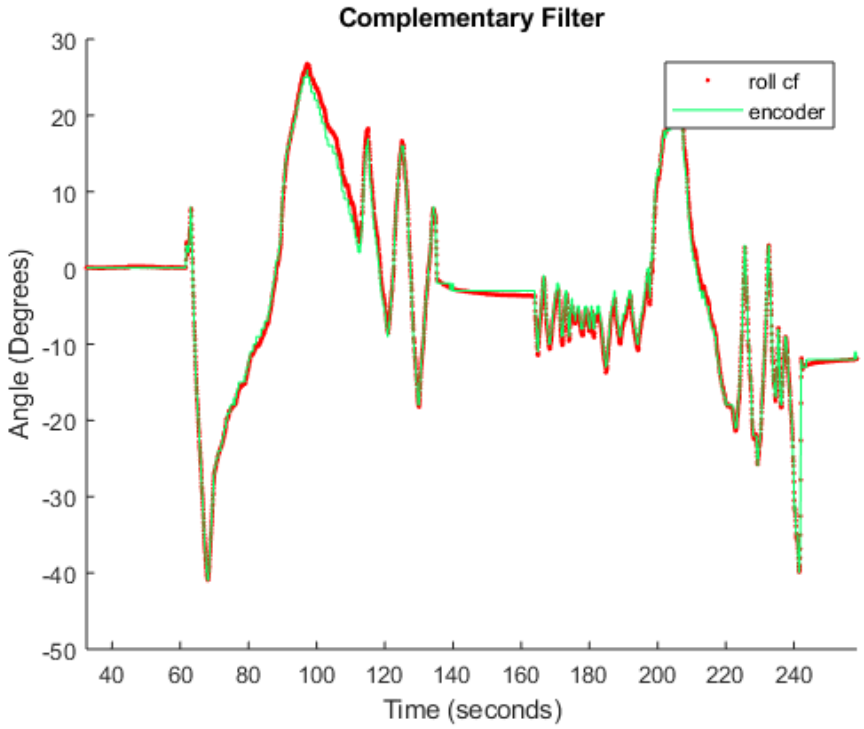


Figure 4.14: CF Roll angle compared to the encoder angle on the validation apparatus.

It should be noted that the magnetometer is particularly sensitive to noise. This is visible in the roll angle signal in Fig. 4.15. The accelerometer is also noisy, but less so. Possible sources for noise include the various signal and power wires for the validation apparatus, the wall power outlets, and other active electrical components near by.

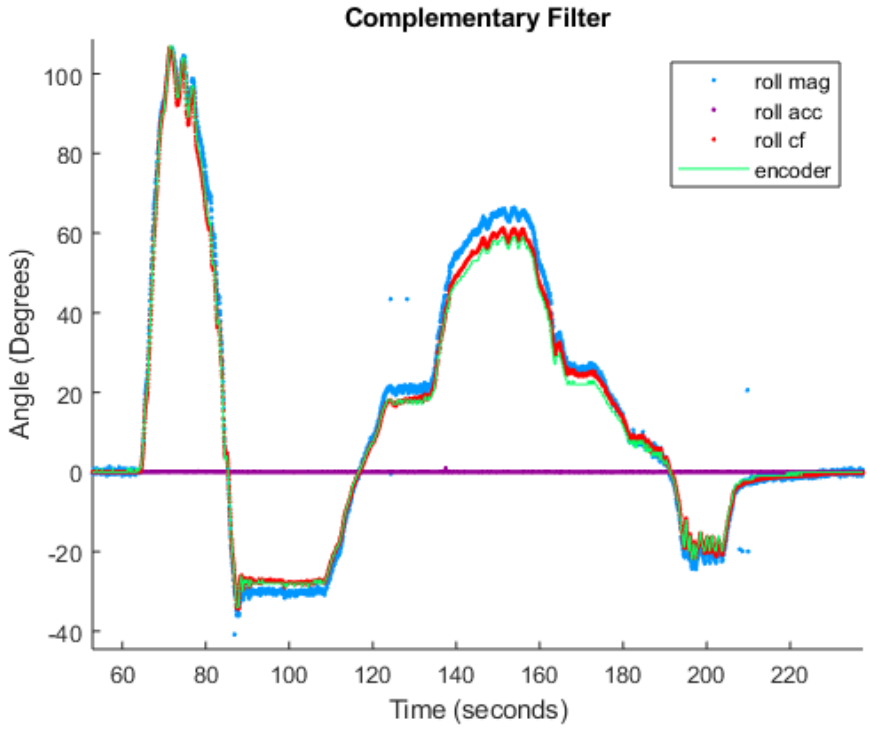


Figure 4.15: Roll signal comparison with the validation apparatus positioned at 90°, normal to the horizontal plane.

Table 4.2 shows the mean, standard deviation, and apparatus orientation for testing the AHRS with varying magnitude in aiding vector contribution.

The apparatus proved to be invaluable in both debugging and designing of the final complementary filter used for navigation.

4.1.8 Field Experiment Results: COG vs CF Yaw

After conducting experiments with the validation apparatus, the CF was tested onboard the Slug 2 ASV. The vehicle was controlled remotely to some arbitrary path, transmitting position and sensor measurements back to a ground station. Fig. 4.16 shows the position of the vehicle.

The COG angle and the CF yaw angle were compared over time. Fig. 4.17 shows the two angles.

Table 4.2: Mean error, standard deviation of error, and validation apparatus orientation.

	μ (Degrees)	σ (Degrees)	θ_{va} (Degrees)
CF	-0.2443	1.0563	90°
Magnetometer	-0.9838	2.9558	90°
Accelerometer	9.7167	27.8414	90°
CF	-0.2708	1.3275	45°
Magnetometer	-0.0129	2.5085	45°
Accelerometer	2.0780	5.2003	45°
CF	-0.1089	0.5906	0°
Magnetometer	3.5137	5.4626	0°
Accelerometer	-2.8029	4.3261	0°

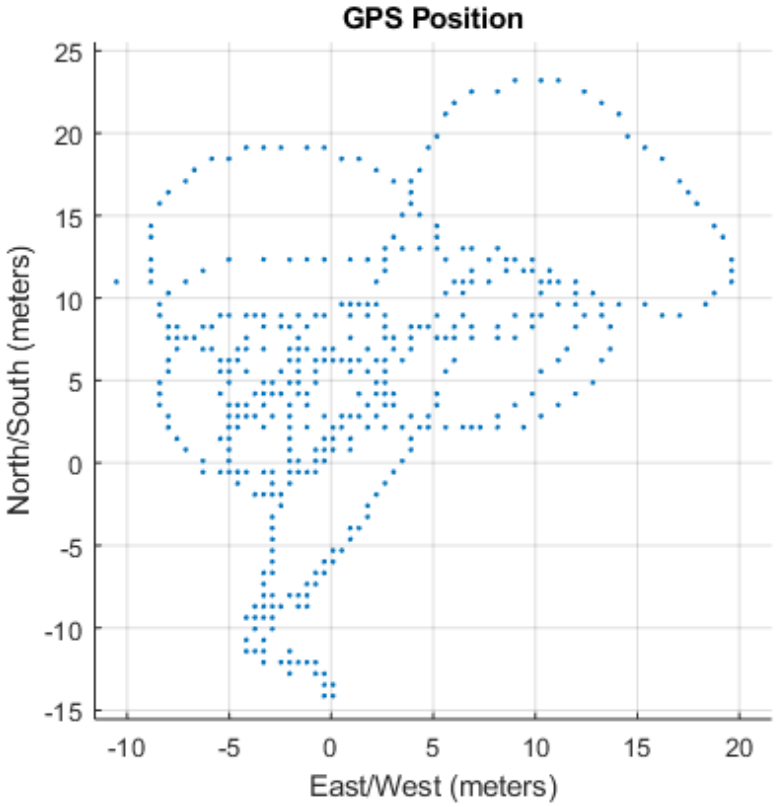


Figure 4.16: GPS position of the Slug 2 ASV, being remotely controlled to collect measurements for attitude estimates, and other data.

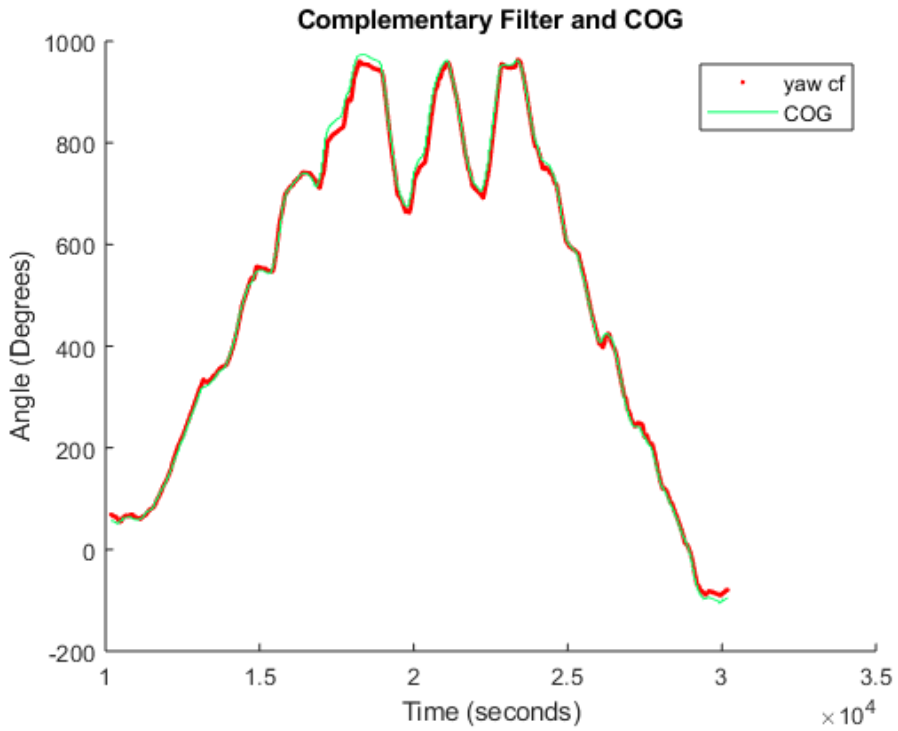


Figure 4.17: A comparison of the COG angle and CF estimated yaw angle over time. There is large agreement between the two. This highlights the difference in COG and CF Yaw.

A histogram of the error, shown in Fig. 4.18 was created to better understand and characterize the performance of the CF. The majority of estimates were near zero.

This experiment was repeated a number of times to further validate that; 1) the COG angle could be used in feedback for navigation, because it followed the CF yaw heading angle over time within a permissible error margin, and 2) the CF yaw could also be used as a higher-frequency heading angle in between or in place of the COG angles. Fig. 4.19 shows another example of the GPS position recorded by another ASV, called the Slug 3. Fig. 4.20 shows the recorded COG angle and the CF yaw angle. Fig. 4.21 shows the histogram of the angle error between the two angle signals over time.

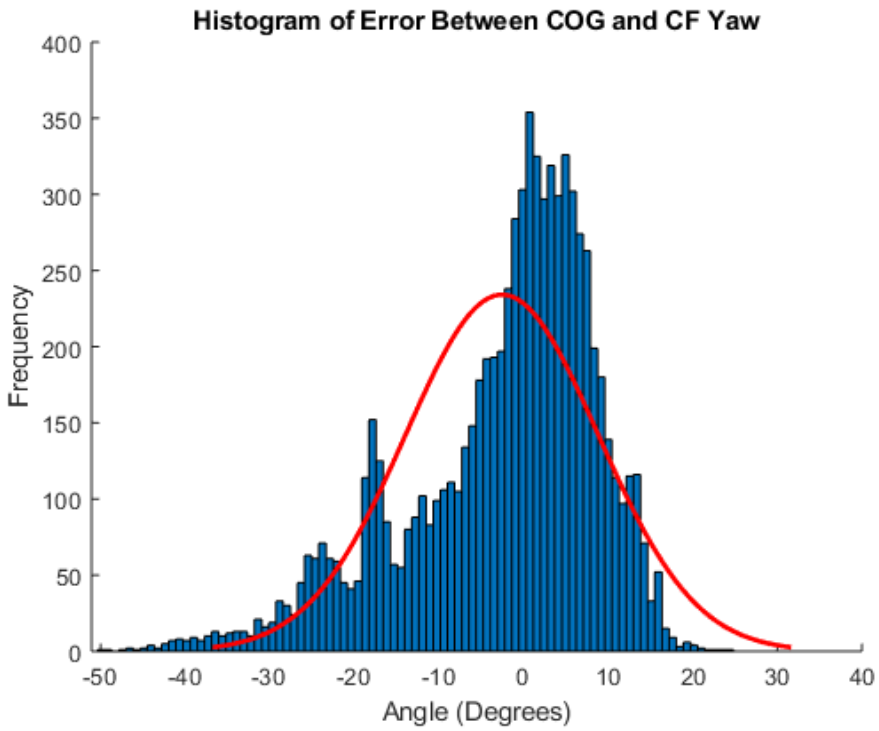


Figure 4.18: A histogram of the frequency (vertical axis) of angle error (horizontal axis).

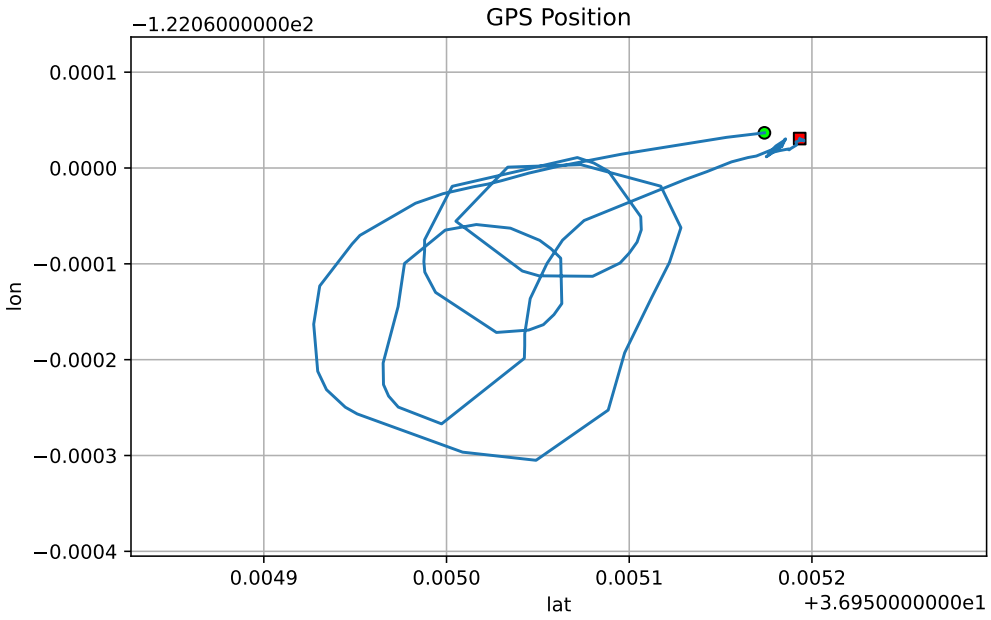


Figure 4.19: GPS position of the Slug 3 ASV, being remotely controlled to collect measurements for attitude estimates, and other data.

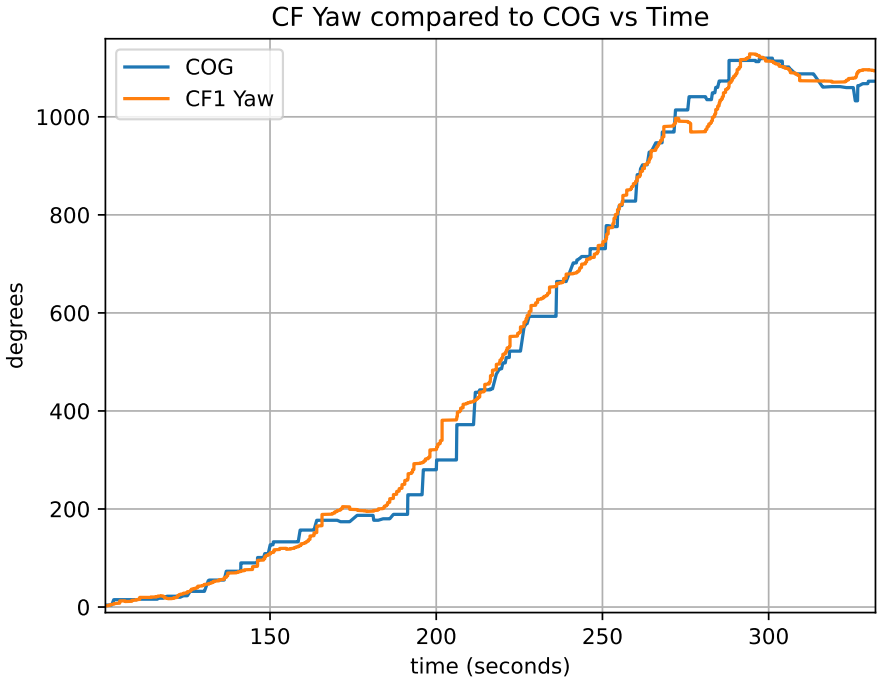


Figure 4.20: Another comparison of the COG angle and CF estimated yaw angle over time. There is some agreement between the two. This highlights the difference in COG and CF Yaw.

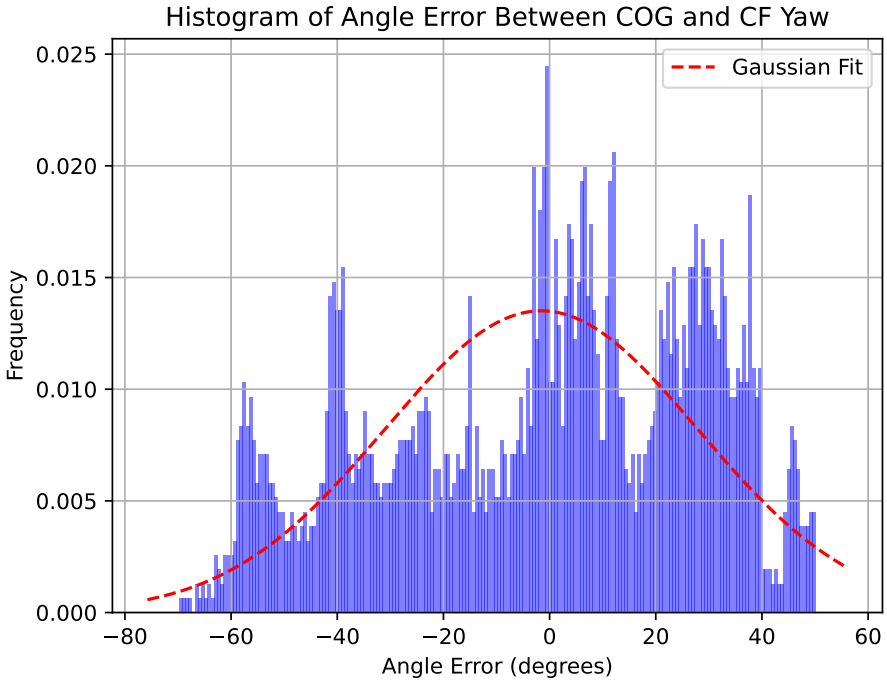


Figure 4.21: Another histogram of the frequency (vertical axis) of angle error (horizontal axis). The mean is $-1.573°$ and the standard deviation is $29.530°$

4.2 Conclusion and Caveats

This chapter showed the evaluation of a small-scale AHRS. Different methods of attitude estimation including EKF, TRIAD, and CF AHRSs were discussed, simulated, and experimental validated using a custom AHRS validation apparatus developed. Further AHRS validation was done by comparing the estimated yaw angle from the CF AHRS with the COG angle from the GPS receiver over time. The error signals between the two were examined. The results of the validation were promising. The attitude estimate matched the truth measurement to a mean of less than 0.3° and a standard deviation of less than 1.3° on the apparatus. The match between the GPS COG and yaw angle estimate had more error, with a standard deviation of 12°, but had good agreement with a mean of less than 3°.

It should be noted that there are many other forms of attitude estimation algorithms for AHRSs, and the ones discussed here were chosen due to their popularity, over-all robustness, and to highlight the key differences in these approaches. The AHRS algorithms discussed in this chapter are meant to represent the main approaches to solving the attitude estimation problem. At the speeds at which an ASV generally operates (which is usually less than 4-meters per second), there is no immediate advantage or disadvantage to choosing any particular AHRS algorithm considered in this thesis. However, from an implementation standpoint, the CF AHRS was chosen because of it's relatively simple design and low-computational cost. It was easily coded in C for the Max32 microcontroller.

One main caveat is that all of these methods require a substantial amount of time tuning parameters; whether it is the proportional gains which require tuning for the CF, or determining the correct process noise for the EKF. The time it takes to tune these algorithms can grow very fast. One exception to this is the TRIAD algorithm.

Another caveat shared by all of these algorithms that require a magnetometer is that their designs assume that the inertial magnetic field vector is constant. This is not true in two ways: 1) noise from motors, servos, and other electronic components can easily distort the local magnetic field, and less commonly 2) if the vehicle moves far enough the Earth's magnetic field will be different depending on where the vehicle is located on the surface of Earth. This is another reason why COG is a useful metric to use with these ARHS algorithms for the purposes of navigation.

Chapter 5

System Modeling

In order to control the ASV, the system dynamics must be known. This can be done at different levels of complexity, stemming from the sub-system dynamics to the macro-system dynamics. The sub-system dynamics include components such as the motors for the propellers and the servos for the rudders. The question of conducting system identification in relation to the models that are discussed in this chapter is presented in Ch. 6

5.0.1 Connection to the Overarching Theme

The researcher that deploys an ASV neither needs nor wants to know the exact mathematical model of the vehicle. However, it is important to outline how to model and describe the vehicle as a dynamical system to later design controllers so that the ASV can execute specific maneuvers. Ideally, the researcher should be able to designate an area of water or field, deploy the ASV, and allow it to explore or navigate autonomously. In later chapters, control of the ASV will be essential for implementing different path planners with subsequent spatial estimators to explore and estimate a field.

One of the most fundamental aspects of autonomous systems is building a model of the system. Generally, this means that the inputs and outputs of the system in question are specified and their relation is described by one or more mathematical equations (often differential equations). A set of parameters are determined as part of these equations to further aid in the mathematical description of a specific system. In an ideal world few equations would be necessary to describe the system and they would be very simple. However, many (if not most systems) are so complicated such that a *fully* descriptive mathematical model, based on first-physics principles can easily become so detailed and complicated that it becomes infeasible to compute, let alone control. This is where the idea of *approximating* these systems becomes useful. Furthermore, when these systems are part of a closed-loop feedback control system, modeling errors are mitigated by the control system.

This chapter discusses a number of system models to describe the motion of an ASV or similar vehicle. These models are approximations of otherwise complicated systems. A boat for example, could be modeled to include the surface friction of the water along the hull, the drag of the rudder as a function of angle, the effects of wind disturbances, and all of the hydrodynamic interactions due to fluid mechanics of the water it moves through. However, the work in this thesis avoids such detailed models. Instead, the focus is on a set of simpler models with a handful of parameters that can be identified or estimated. This includes the rudder servo, the kinematic inverse bicycle model, the ship steering model, and a limited set of extensions to the models.

5.1 Rudder Servo

The effects of disturbances on the rudder actuator prevents the commanded rudder angle from being the actual rudder angle. This mismatch is important; the controller is required to account for this mismatch, but this requires knowledge of the parameters of the rudder model.

The servo can be modeled in the frequency domain as a low-pass filter:

$$G_s(s) = \frac{\alpha(s)}{u(s)} = \frac{b}{s+a} \tag{5.1}$$

where $\alpha(s)$ is the output angle and $u(s)$ is the input command. Note that this simplified model has a non-unity gain, and a time constant. The system can be discretized as follows:

$$G_s(z) = \underbrace{(1-z^{-1})}_{\text{ZOH}} \mathcal{Z}\left\{\frac{1}{s}G_s(s)\right\} \tag{5.2}$$

$$G_s(z) = (1-z^{-1})\mathcal{Z}\left\{\frac{b}{s(s+a)}\right\} \tag{5.3}$$

$$G_s(z) = \frac{b_0}{z+a_0} \tag{5.4}$$

where $b_0 = \frac{b}{a}(1-e^{-a\Delta T})$, $a_0 = -e^{-a\Delta T}$. Estimation of this parameter will be discussed later in Sec. 6.1.1.

5.2 Kinematic Model

Among the various methods to model a surface vehicle (e.g.: a boat) there exists a relatively simple kinematic model known as the inverse bicycle model. This model has the advantage of being simple to compute, but is not very accurate;

it is based on a no-cross-flow condition on the rudder. It is non-linear and time varying, and further assumes a constant velocity through the water. Work by [51] used this model successfully for experimental autonomous navigation research missions in the ocean. The model is,

$$\begin{bmatrix} \dot{N} \\ \dot{E} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} v_w \cos(\psi) \\ v_w \sin(\psi) \\ -\frac{v_w \tan(\delta)}{l_{\text{base}}} \end{bmatrix} + \underbrace{\begin{bmatrix} v_{c_n} \\ v_{c_e} \\ 0 \end{bmatrix}}_{\text{water current velocity}} \tag{5.5}$$

where v_w is the water velocity and v_{c_n} and v_{c_e} are the water current velocities in the North and East directions respectively, and $\dot{N}$ and $\dot{E}$ are the rate of change of position in the North and East directions respectively. This model corresponds to Fig. 5.1. The heading angle is ψ, and the yaw-rate is $\dot{\psi}$. The rudder angle is δ. The wheel-base, l_{base}, of the model can be related to the radius, r, of an arcing turn with the following equation.

$$\tan(\delta) = \frac{l_{\text{base}}}{r} \tag{5.6}$$

The model can be linearized for small steering angles, and localized into the body frame or local path coordinates for control. This removes the non-linearity, but still allows for global coordinate modeling.

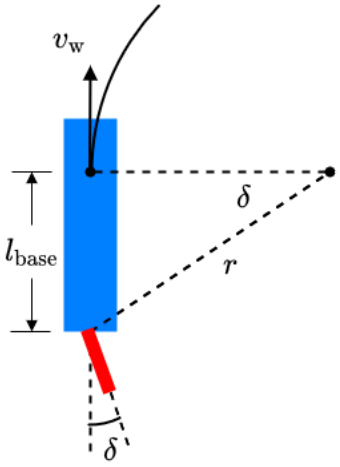


Figure 5.1: Inverse bicycle model implemented in [51]

5.3 Nomoto Model

In order to simulate the vehicle within a local tangent plane, a discrete state space representation is developed. Let

$$\underline{\mathbf{x}}_k = \begin{bmatrix} \psi_k & r_k & x_k & y_k & v_k \end{bmatrix}' \tag{5.7}$$

be the state of the vehicle such that k represents the $k-$th time step, ψ is the vehicle heading in radians with respect to North, East, Down (NED) coordinates, r is the yaw rate in radians per second, x and y is the vehicle position in meters in North and East on a local tangent plane, and v is the magnitude of the velocity. The classical discrete state space formulation is

$$\underline{\mathbf{x}}_{k+1} = \underline{\Phi}_k \underline{\mathbf{x}}_k + \underline{\Gamma}_k u_k \tag{5.8}$$

where $\underline{\Phi}_k$ is the discrete dynamics matrix, or state transition matrix, $\underline{\Gamma}_k$ is the discrete input matrix, and u_k is the input. In this case, u_k is a scalar input. It is

now necessary to specify the system of equations that describe the vehicle motion. The first equation describes the heading angle of the vehicle. It can be written simply and discretely as

$$\psi_{k+1} = \psi_k + r\Delta T \tag{5.9}$$

where ΔT represents the time between each time step. The yaw rate is more complex, but a partial derivation from [76] is restated here for completeness. Let T_c be the effective yaw-rate time constant associated with the model in the continuous (frequency) domain. This is dependent on the size of the vehicle. For larger vehicles, it generally takes more time to establish an appreciable yaw-rate, so T_c will be larger. Conversely, for smaller vehicles the time constant will be smaller. There are exceptions to this depending on vehicle geometry and speed, but in general this is a good assumption. Let

$$T_c\dot{r} + r = K_c\delta \tag{5.10}$$

where K_c is the static yaw-rate gain associated with the model in the continuous domain, and δ is the rudder angle of the vehicle. Eq 5.10 can be converted to the frequency domain:

$$\mathbf{s}r(\mathbf{s})T_c + r(\mathbf{s}) = K_c\delta(\mathbf{s}) \tag{5.11}$$

This results in the continuous first order Nomoto ship steering model in the frequency domain, discussed in [76],

$$\frac{r(\mathbf{s})}{\delta(\mathbf{s})} = \frac{K_c}{1 + T_c\mathbf{s}} \tag{5.12}$$

$\mathcal{Z}$-transform of Zero-Order-Hold (ZOH) Equivalent

Discretizing Eq. (5.12) by applying the $\mathcal{Z}$-transform of the ZOH equivalent results in Eq. (5.13).

$$\frac{r(z)}{\delta(z)} = \frac{K_{\text{d,yaw}}}{z - T_{\text{d,yaw}}} \tag{5.13}$$

where $K_{\text{d,yaw}} = \frac{K_{\text{c}}}{T_{\text{c}}}(-e^{-\Delta T/T_{\text{c}}} + 1)$ and $T_{\text{d,yaw}} = e^{-\Delta T/T_{\text{c}}}$. Recall that ΔT is the sample time. Then the difference equation is:

$$r_{k+1} = T_{\text{d,yaw}} r_k + K_{\text{d,yaw}} \delta_k \tag{5.14}$$

or

$$r_{k+1} = e^{-\Delta T/T_{\text{c}}} r_k + \frac{K_{\text{c}}}{T_{\text{c}}}(1 - e^{-\Delta T/T_{\text{c}}})\delta_k \tag{5.15}$$

The heading angle ψ and position integration of the vehicle in both x and y, or East and North are as follows:

$$\psi_{k+1} = r_k \Delta T + \psi_k \tag{5.16}$$

$$x_{k+1} = x_k + \Delta T v_k \sin(\psi_k) \tag{5.17}$$

$$y_{k+1} = y_k + \Delta T v_k \cos(\psi_k) \tag{5.18}$$

5.4 Augmented Nomoto Model

Using Eq. 5.9, Eq. 5.14, Eq. 5.17, and Eq. 5.18, the discrete state transition matrix, $\underline{\Phi}_k$ from Eq. 5.8 and the input matrix, $\underline{\Gamma}_k$ can be specified.

$$\underbrace{\begin{bmatrix}\psi_{k+1}\\ r_{k+1}\\ x_{k+1}\\ y_{k+1}\\ v_{k+1}\\ \delta_{k+1}\end{bmatrix}}_{\underline{\mathbf{x}}_{k+1}} = \underbrace{\begin{bmatrix}1 & \Delta T & 0 & 0 & 0 & 0\\ 0 & T_{\mathrm{d,yaw}} & 0 & 0 & 0 & K_{\mathrm{d,yaw}}\\ 0 & 0 & 1 & 0 & \Delta T\sin(\psi_k) & 0\\ 0 & 0 & 0 & 1 & \Delta T\cos(\psi_k) & 0\\ 0 & 0 & 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 0 & 0 & -a_0\end{bmatrix}}_{\underline{\Phi}_k} \underbrace{\begin{bmatrix}\psi_k\\ r_k\\ x_k\\ y_k\\ v_k\\ \delta_k\end{bmatrix}}_{\underline{\mathbf{x}}_k} + \underbrace{\begin{bmatrix}0\\0\\0\\0\\0\\b_0\end{bmatrix}}_{\underline{\Gamma}_k} u_k \tag{5.19}$$

where a_0 and b_0 are a coefficients of the discrete rudder transfer function:

$$G_s(z) = \frac{b_0}{z + a_0} = \frac{\delta(z)}{u(z)} \tag{5.20}$$

This can be written as a difference equation:

$$\delta_{k+1} = -a_0\delta_k + b_0 u_k \tag{5.21}$$

These parameters, a_0 and b_0 describe the lag associated with the rudder command and true rudder angle.

Eq. 5.19 is the discrete, augmented, first-order Nomoto ship steering model. It should be noted that to simplify the control for this model, the velocity is assumed to be constant and is therefore not expressed as a controllable input in $\underline{\Gamma}_k$. While this greatly simplifies the simulation, it is not difficult to add back in. Additionally, the lag associated with a rudder command u_k, and the actual rudder angle δ is expressed in this model. Using Eqs. (5.19-5.21), Eq. 5.7 becomes:

$$\underline{\mathbf{x}}_k = \begin{bmatrix}\psi_k & r_k & x_k & y_k & v_k & \delta_k\end{bmatrix}' \tag{5.22}$$

The augmented Nomoto model is used in simulation to estimate the position of the vehicle and a GNC algorithm is designed around it. The algorithm handles switching waypoints, trajectory generation, and trajectory tracking.

5.5 Newtonian Model

As one of the goals of this work, an alternative Newtonian model is introduced. The design of the model considers forces, mass, and angular momentum. The full model in $\mathbb{R}^3$ is described below to show that it may be used to model other vehicles that, at first glance seem different. This model could potentially be used to describe other vehicles, such as a rocket. Fig. 5.2 depicts the top-down view of this Newtonian model. It is inspired, partially by the 1st-order Nomoto ship steering, but is primarily designed based on basic (Newtonian) first-physics principles.

The Newtonian model assumes three main parts: 1) a point-mass sub-model with position relative to the inertial reference frame, 2) an orientation sub-model with respect to the inertial reference frame, and 3) a thrust force vector $\mathbf{F}$ connected to the point-mass at a distance r, with thrust vector angle δ_r, and expressed in the body-fixed frame. Given the general continuous state-space representation,

$$\dot{\mathbf{x}}_{\mathrm{c,p}}(t) = \mathbf{A}_{\mathrm{c,p}}\mathbf{x}_{\mathrm{c,p}}(t) + \mathbf{B}_{\mathrm{c,p}}\mathbf{u}(t) \tag{5.23}$$

we populate the dynamics matrix and input matrix based on the equations of motion, where the acceleration in a single dimension is

$$\ddot{x}(t) = \frac{[\sum F_x]^I}{m_v} \tag{5.24}$$

such that m_v is the mass of the vehicle and $[\sum F_x(t)]^I$ is the sum of the forces in

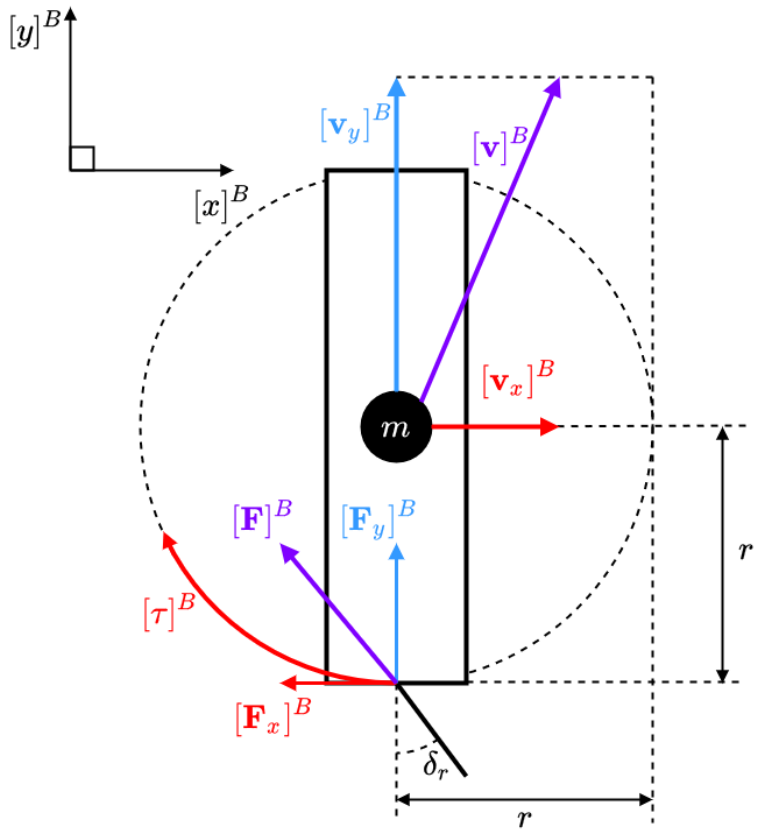


Figure 5.2: A partial representation showing the top-down view of the proposed Newtonian model. The rudder or thrust vector angle δ_r is shown in relation to the resulting torque for the orientation sub-model and linear force for the point-mass sub-model. An assumption here is that the center of mass is in the center of the vehicle, though this is not necessarily true for all vehicles.

the x dimension with respect to the inertial reference frame. The input is the sum of thrust and drag forces (and possibly other disturbances):

$$\mathbf{u}(t) = \mathbf{F} - \mathbf{F}_{\text{drag}} \tag{5.25}$$

The drag force is defined as

$$\mathbf{F}_{\text{drag}} = \frac{1}{2}\rho \mathbf{v}^2 C_{\text{drag}} A \tag{5.26}$$

The model parameters include the distance from the center of mass to the gimbal thrust joint of the vehicle $r = l_{\text{rad}}$ (see Fig. 5.2), the moment of inertia of the vehicle I (which may be a sphere, a rod, or other simple geometry), the cross-sectional area associated with the vehicle geometry A, the fluid medium density ρ, and the drag coefficient C_{drag}. These parameters can be found experimentally using a few different approaches, including system identification as described in Ch. 6. The advantages to this model is that it is more detailed than the bicycle model and the Nomoto ship-steering model. Unlike the other two it is also strictly controllable, simplifying the control to a single thrust vector angle, and the dynamic matrices are both linear.

5.5.1 Point-Mass Sub-model

We may write the continuous state space representation from Eq. (5.23) for the point-mass sub-model as

$$\underbrace{\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ \ddot{x} \\ \ddot{y} \\ \ddot{z} \end{bmatrix}}_{\dot{\mathbf{x}}_{\text{c,p}}} = \underbrace{\begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}}_{\mathbf{A}_{\text{c,p}}} \underbrace{\begin{bmatrix} x \\ y \\ z \\ \dot{x} \\ \dot{y} \\ \dot{z} \end{bmatrix}}_{\mathbf{x}_{\text{c,p}}} + \underbrace{\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1/m_v & 0 & 0 \\ 0 & 1/m_v & 0 \\ 0 & 0 & 1/m_v \end{bmatrix}}_{\mathbf{B}_{\text{c,p}}} \mathbf{u} \tag{5.27}$$

where the force of thrust and drag forces are considered, though other disturbance forces may be included if desired. The position of the vehicle in a local tangent space is represented by x, y, z. The velocity is represented by $\dot{x}, \dot{y}, \dot{z}$, and the acceleration is represented by $\ddot{x}, \ddot{y}, \ddot{z}$. The mass of the vehicle is m_v.

5.5.2 Orientation Sub-model

The orientation sub-model in continuous state space representation is introduced as

$$\dot{\mathbf{x}}_{\mathrm{c,o}}(t) = \mathbf{A}_{\mathrm{c,o}}\mathbf{x}_{\mathrm{c,o}}(t) + \mathbf{B}_{\mathrm{c,o}}\mathbf{u}(t) \tag{5.28}$$

Recall that the angular acceleration is equal to the torque divided by the moment of inertia of the vehicle. Consider the angular acceleration about the x-axis in the body-fixed frame.

$$\dot{p} = \frac{\tau_x}{I} = \frac{l_{\mathrm{rad}}}{I}\mathbf{F}_x \tag{5.29}$$

Since torque $\tau_x = l_{\mathrm{rad}} \times \mathbf{F}_x$, we may consider the angular acceleration due to torque about all three axes, assuming a perfect symmetrical rigid body, and rewrite Eq. (5.28) as

$$\begin{bmatrix}\dot{p}\\ \dot{q}\\ \dot{r}\end{bmatrix} = \underbrace{\begin{bmatrix}0 & 0 & 0\\ 0 & 0 & 0\\ 0 & 0 & 0\end{bmatrix}}_{\mathbf{A}_{\mathrm{c,o}}}\begin{bmatrix}p\\ q\\ r\end{bmatrix} + \underbrace{\begin{bmatrix}1/I & 0 & 0\\ 0 & 1/I & 0\\ 0 & 0 & 1/I\end{bmatrix}}_{\mathbf{B}_{\mathrm{c,o}}}\underbrace{\begin{bmatrix}\tau_x - \tau_{\mathrm{drag},x}\\ \tau_y - \tau_{\mathrm{drag},y}\\ \tau_z - \tau_{\mathrm{drag},z}\end{bmatrix}^B}_{\mathbf{u}} \tag{5.30}$$

where the angular rate vector $\omega_{\mathcal{B}} = \begin{bmatrix}p & q & r\end{bmatrix}$, and each element is the angular rate as measured in the body-fixed frame. It is emphasized that a key assumption is that the rigid body is symmetrical. This is not true in practice, but is a useful simplification. Alternatively, this results in terms in the off-diagonal elements of $\mathbf{B}_{\mathrm{c,o}}$. Note that the forces, as shown in Eq. (5.30) and Fig. 5.2 express the input with respect to the body-fixed frame. Eq. (5.30) corresponds with Newton's second law:

$$I\ddot{\Theta} = \sum \tau_{\text{ext}} \tag{5.31}$$

where τ_{ext} is the external torque applied to the rigid body and $\ddot{\Theta}$ is the angular acceleration. We may also consider Euler's equations for rigid body dynamics,

$$\mathcal{I}\dot{\omega}_{\mathcal{B}} + \omega_{\mathcal{B}} \times (\mathcal{I}\omega_{\mathcal{B}}) = \mathbf{M} \tag{5.32}$$

where $\mathcal{I}$ is the inertial matrix and the applied torques are represented by $\mathbf{M}$. We may rewrite Eq. (5.32) as,

$$\dot{\omega}_{\mathcal{B}} = \mathcal{I}^{-1}\left[\mathbf{M} - \omega_{\mathcal{B}} \times (\mathcal{I}\omega_{\mathcal{B}})\right] \tag{5.33}$$

We may discretize the Newtonian model as follows.

$$\begin{bmatrix} \Phi_{\text{d,p}} & \Gamma_{\text{d,p}} \\ \mathbf{0} & \mathbf{I} \end{bmatrix} \approx \exp\left(\begin{bmatrix} \mathbf{A}_{\text{c,p}} & \mathbf{B}_{\text{c,p}} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} \mathrm{d}t\right) \tag{5.34}$$

$$\begin{bmatrix} \Phi_{\text{d,o}} & \Gamma_{\text{d,o}} \\ \mathbf{0} & \mathbf{I} \end{bmatrix} \approx \exp\left(\begin{bmatrix} \mathbf{A}_{\text{c,o}} & \mathbf{B}_{\text{c,o}} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} \mathrm{d}t\right) \tag{5.35}$$

where $\mathrm{d}t$ is the time step of the simulation or onboard computation. Next, we integrate the angular rates using Eq. (4.57)-(4.63). The angular rates are a result of the sum of the input force(s) and disturbance force(s). This yields the net torque τ discussed earlier. We can define the angular rate vector as

$$\omega_{\mathcal{B}} = \begin{bmatrix} p & q & r \end{bmatrix}^{\mathrm{T}} \tag{5.36}$$

and

$$\Omega(\omega_{\mathcal{B}}) = \begin{bmatrix} -\left[\omega_{\mathcal{B}}\times\right] & \omega_{\mathcal{B}} \\ -\omega_{\mathcal{B}}^{\mathrm{T}} & 0 \end{bmatrix} \tag{5.37}$$

such that Eq. (4.57) and Eq. (4.62) can be used as follows

$$\dot{\mathbf{q}} = \frac{1}{2}\Omega(\omega_{\mathcal{B}})\mathbf{q} \tag{5.38}$$

where $\mathbf{q}$ is the attitude quaternion for the orientation sub-model. Recall from Eq. (4.63) that

$$\mathbf{q}_{k+1} = M(\Delta\theta)\mathbf{q}_k \tag{5.39}$$

is the discrete solution. $\Delta\theta$ stems from the constant rate assumption for small time steps from Eq. (4.61). After each t_k time step of duration $\mathrm{d}t$, the attitude quaternion $\mathbf{q}_{k+1}$ is normalized to ensure that the end of the quaternion vector properly represents attitude.

$$\mathbf{q}_{k+1} = \frac{\mathbf{q}_{k+1}}{\left\|\mathbf{q}_{k+1}\right\|} \tag{5.40}$$

This approach is based on those discussed in subsection 4.1.4 for a more complete derivation of the attitude quaternion. The reader is referred to that subsection to learn things such as how to extract the Euler angles from the attitude quaternion. After the orientation model is calculated for a step, the net force input vector can be rotated from the body-fixed reference frame to the inertial reference frame. Then Eq. (5.23) can be computed and model will have been fully propagated for a time step.

The orientation sub-model Eq. 5.30 can also be modified to become:

$$\underbrace{\begin{bmatrix}\dot{\phi}\\ \dot{\theta}\\ \dot{\psi}\\ \ddot{\phi}\\ \ddot{\theta}\\ \ddot{\psi}\end{bmatrix}}_{\dot{\mathbf{x}}_{\mathrm{c,o}}} = \underbrace{\begin{bmatrix}p\\ q\\ r\\ \dot{p}\\ \dot{q}\\ \dot{r}\end{bmatrix}}_{\dot{\mathbf{x}}_{\mathrm{c,o}}} = \underbrace{\begin{bmatrix}0 & 0 & 0 & 1 & 0 & 0\\ 0 & 0 & 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 0 & 0 & 1\\ 0 & 0 & 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 0 & 0 & 0\end{bmatrix}}_{\mathbf{A}_{\mathrm{c,o}}} \underbrace{\begin{bmatrix}\phi\\ \theta\\ \psi\\ p\\ q\\ r\end{bmatrix}}_{\mathbf{x}_{\mathrm{c,o}}} + \underbrace{\begin{bmatrix}0 & 0 & 0\\ 0 & 0 & 0\\ 0 & 0 & 0\\ 1/I & 0 & 0\\ 0 & 1/I & 0\\ 0 & 0 & 1/I\end{bmatrix}}_{\mathbf{B}_{\mathrm{c,o}}} \underbrace{\begin{bmatrix}\tau_x - \tau_{\mathrm{drag},x}\\ \tau_y - \tau_{\mathrm{drag},y}\\ \tau_z - \tau_{\mathrm{drag},z}\end{bmatrix}^B}_{\mathbf{u}} \tag{5.41}$$

where ϕ, θ, ψ correspond to roll, pitch, and yaw respectively. Eq. (5.41) make use of the fact that $p = \dot{\phi}$, $q = \dot{\theta}$, and $r = \dot{\psi}$. It also employees the tautology that $p = p$, $q = q$, and $r = r$ for the sake of showing controllability for angular position and angular rate (which is explored in Sub-section 5.5.3 below). Again, the angular rates constitute the elements of the vector $\omega_{\mathcal{B}}$ which is integrated using Eq. (4.57)-(4.63).

5.5.3 Controllability

It should be noted that both Eq. (5.27) and Eq. (5.30) represent sub-models for the Newtonian model. Both sub-models are linear and time-invariant. As such, we may show that the Newtonian model is controllable in the continuous domain. This is done using the following equation

$$\mathcal{C} = \begin{bmatrix}\mathbf{B} & \mathbf{AB} & \mathbf{A}^2\mathbf{B} & \cdots \mathbf{A}^{n-1}\mathbf{B}\end{bmatrix} \tag{5.42}$$

for both Eq. (5.27) and Eq. 5.30, where the $\mathbf{A}$ and $\mathbf{B}$ matrices correspond to $\mathbf{A}_{\mathrm{c,p}}$ and $\mathbf{B}_{\mathrm{c,p}}$ for the continuous point-mass sub-model, and to $\mathbf{A}_{\mathrm{c,o}}$ and $\mathbf{B}_{\mathrm{c,o}}$ for the continuous orientation sub-model. Following the same subscript notation we have the controllability matrix for the point-mass model being denoted as $\mathcal{C}_{\mathrm{c,p}}$ and the

orientation model as $\mathcal{C}_{\mathrm{c,o}}$. Computing the rank of $\mathrm{rank}(\mathcal{C}_{\mathrm{c,p}}) = 6$. The point-mass model is full rank, and therefore the position is controllable.

For the orientation sub-model we see that $\mathrm{rank}(\mathcal{C}_{\mathrm{c,o}}) = 6$ and is full rank with respect to the sub-model. This means that the angular orientation of the sub-model is also controllable. Both sub-models that constitute the Newtonian model are full rank and therefore controllable.

5.6 Conclusion and Caveats

This chapter explained how to model an ASV in a number of different ways and at different levels. This included the kinematic modeling approach, using the inverse bicycle model, the Nomoto ship-steering model, an augmented version of the Nomoto model, and Newtonian model that considered forces more explicitly and in full three-space. The simplest model was the inverse bicycle model, but it is time-varying and non-linear. The Nomoto models are slightly more complicated, but the augmented version allowing for a convenient discrete state-space representation. However it too is time-varying and non-linear. The Newtonian model is convenient because the point-mass sub-model is linear and time-invariant. It was shown that it was fully controllable. The orientation sub-model introduces more complexity, but can be conveniently described using the same equations used in the Ch. 4. Furthermore, the Newtonian model can describe vehicles that can move in more dimensions than both the inverse bicycle and Nomotor models. It does however rely on a simple geometry (shown in Fig.5.2). It assumes that the center of mass is in the center of the vehicle's geometry, which is not true for all vehicles. Also, the Newtonian model relies heavily on the fact that input requires calculating the rotation between the input force vector with respect to the body-fixed frame and to the inertial reference frame. For instance, a thrust

input to the point-mass model also has to be rotated into the body-fixed frame to propagate the orientation sub-model. That means this model is requires a bit more computation for simulation.

It should be noted that there are many other models and other versions of these models than the ones that were discussed in this chapter. See [76] and [10] for further discussion of mathematical models of boats and similar vehicles.

Chapter 6

System Identification

The vehicle models were developed in Ch. 5; however many of the parameters within these models are unknown or could be determined from physical measurements to estimate these parameters. We use system identification techniques to find our estimate using measured ouput data and known input data. The corresponding parameters for the motors and servos can be identified using methods such as ARX, ARMAX, recursive least-squares (RLS) or other similar methods outlined in [49]. The macro-system dynamics combine these now identified sub-systems to form an approximate model of the runtime vehicle dynamics. Pseudo-random inputs to the sub-systems and a one-step-ahead prediction are used to generate error signals to iteratively approximate the parameters for a given model.

6.0.1 Connection to the Overarching Theme

Similar to the previous chapter, the researcher using the system will likely not concern themselves with model-specific parameters that help describe a vehicle or subsystem. It is nonetheless important to discuss them here so that the ASV can

be controlled and proper navigation can occur.

6.1 ARX

6.1.1 Rudder Servo

In this section we consider a rudder servo for system identification. The rudder servo is simulated and the model parameters are estimated using Auto-Regression with eXternal inputs (ARX). This method is specifically chosen for its ease of implementation and practicality. The basic idea is to a generate pseudo-random input signal to the servo and use a sensor (such as an encoder) to measure the angle of the rudder for a given input. The inputs and output angles are recorded. In this case, the amplitude of the input signal is chosen such that the rudder reaches the maximum angle in both the positive and negative directions. Due to the fact that the control system is implemented on an embedded-digital system, Eq. (5.1) is converted to the $\mathcal{Z}$-domain. We use Eqs. (5.2-5.3). Both a_0 and b_0 are constants to be estimated. Eq 5.4 can be converted to a difference equation:

$$G_s(z) = \frac{b_0}{z + a_0} = \frac{\delta_k}{u_k} \tag{6.1}$$

$$\delta_k = \frac{b_0}{z + a_0} u_k \tag{6.2}$$

$$(z + a_0)\delta_k = b_0 u_k \tag{6.3}$$

$$\tag{6.4}$$

Thus:

$$\delta_k = -a_0 \delta_{k-1} + b_0 u_{k-1} \tag{6.5}$$

From Eq. 6.5, it is now possible to use raw data collected from the servo with

the ARX model to estimate the parameters.

$$\alpha_k = \begin{bmatrix} -\delta_{k-1} & u_{k-1} \end{bmatrix} \underbrace{\begin{bmatrix} a_0 \\ b_0 \end{bmatrix}}_{\theta_p} + e(t) \quad t = 0, 1, 2, \ldots, N \tag{6.6}$$

$$\underbrace{\begin{bmatrix} \delta_1 \\ \delta_2 \\ \vdots \\ \delta_N \end{bmatrix}}_{\Delta} = \underbrace{\begin{bmatrix} -\delta_0 & u_0 \\ -\delta_1 & u_1 \\ \vdots & \vdots \\ -\delta_{N-1} & u_{N-1} \end{bmatrix}}_{\mathrm{S}} \underbrace{\begin{bmatrix} a_0 \\ b_0 \end{bmatrix}}_{\theta_p} + \underbrace{\begin{bmatrix} e(1) \\ e(2) \\ \vdots \\ e(N) \end{bmatrix}}_{v} \tag{6.7}$$

The S matrix in Eq. 6.7 contains some of the raw angle outputs measured, $\delta(t)$, and the input $u(t)$. Note that Δ is indexed one step ahead of the corresponding output and input for a given row. Also, the error vector e is purposefully ignored for now but assumed to be small. It is assumed that there are N number of angle measurements and N number of inputs recorded. Eq 6.7 can expressed as follows:

$$\Delta = S\theta_p + v \tag{6.8}$$

This is solved using least squares, the parameter vector θ_p in Eq. 6.8, and thus the constants a_0 and b_0 can be estimated.

$$\hat{\theta}_p = (S^T S)^{-1} S^T \Delta \tag{6.9}$$

Note that because θ_p is being estimated, it is denoted as $\hat{\theta}_p$.

This method has been successfully applied in a V-Rep[1] simulation for a similar servo mechanism. It is assumed that there is a moderate difference in the

[1]V-Rep is a 3D robotics physics simulation software.

parameters estimated when the rudder is in the water versus out of the water. Specifically, the parameters in Eq. 6.1, for the in-water rudder measurements will likely show a longer delay between the input signal and output angle. This is due to the increased loading on the rudder due to hydrodynamic forces, and is likely to be a function of velocity. However, for now we simply use the parameters from $\hat{\theta}_p$ in our model.

The ARX method was applied in simulation out of water.Fig. 6.1 shows the raw pseudo-random input signal in blue and the resulting measured rudder servo angle in orange. Fig. 6.2 shows small time window of the simulated pseudo random rudder servo command signal $\delta(t)$ compared to the estimated $\hat{\delta}(t)$ using the ARX method.

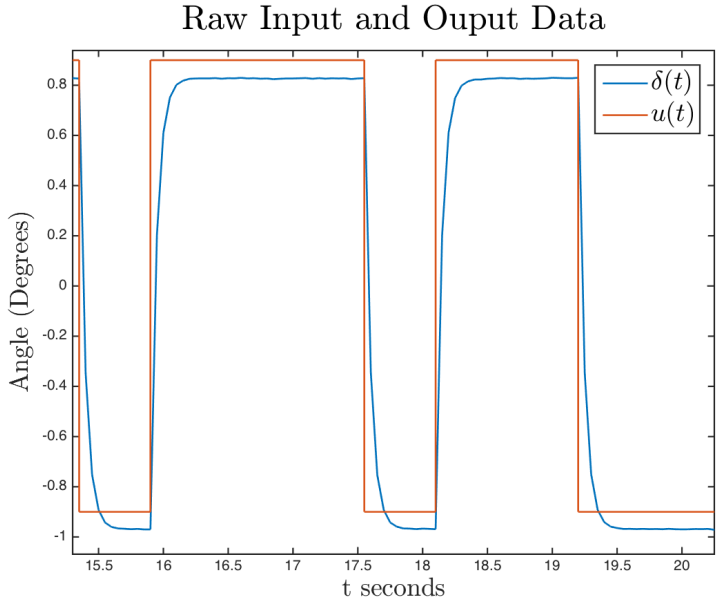


Figure 6.1: A small time window showing the raw pseudo random input $u(t)$ and the measured rudder servo angle $\delta(t)$. There is a noticeable offset and scaling between the two signals.

Note that in Fig. 6.1 one assumption is that PWM corresponds instantly to an angle command. In reality there is noise to the PWM signal.

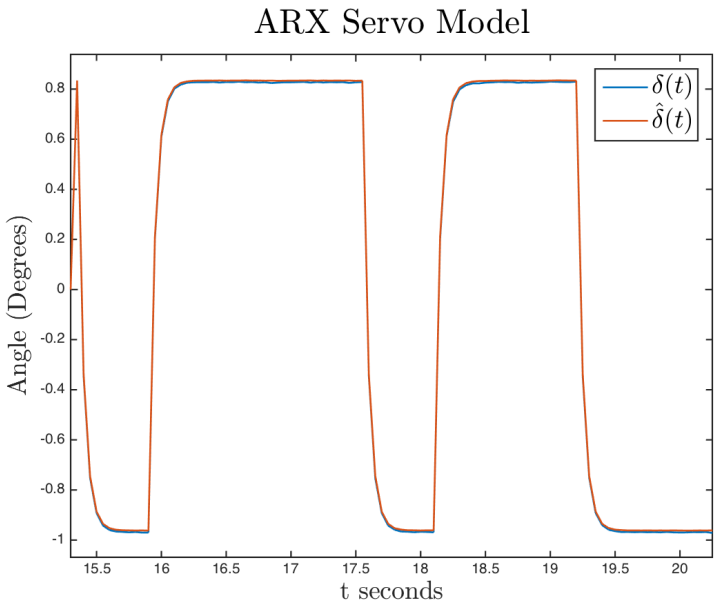


Figure 6.2: A simulation example ARX; the true servo angle δ compared to the estimated signal $\hat{\delta}$. There is good agreement between the two signals.

Fig. 6.3 shows the histogram of the ARX error. The mean $\mu = -5.34195 \times 10^{-19}$ and the standard deviation $\sigma = 5.59588 \times 10^{-05}$.

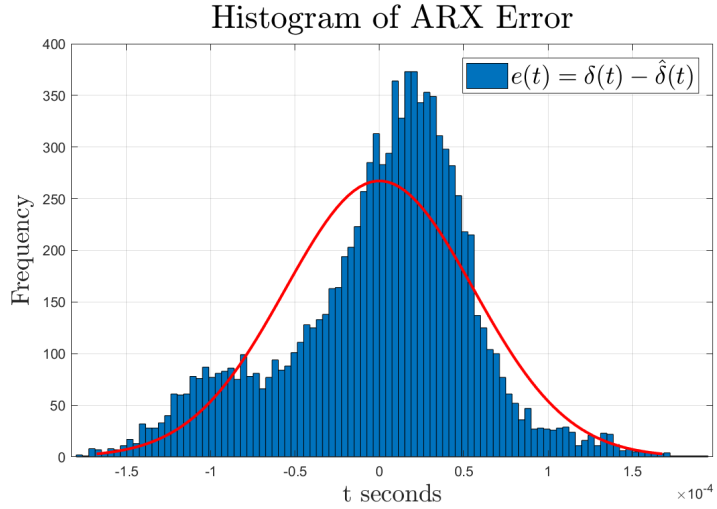


Figure 6.3: A histogram of the ARX error from simulation with a normal distribution fit to the resulting data. The mean $\mu = -5.34195 \times 10^{-19}$ and the standard deviation $\sigma = 5.59588 \times 10^{-5}$.

This ARX experiment was done in simulation. Another alternative is to apply an EKF with the model parameters added to the state vector.

6.2 Model Parameter Estimation

Model parameter estimation is an important aspect of system identification. A model can be formulated to have specific parameters that correspond to empirically known parameters of the physical system. For instance, the inverse bicycle model (Eq. 5.5) possesses the parameter l_{base} which is a physical value that can be inferred from the physical design of the hull of the boat and the center of mass.

6.2.1 Kalman Filter

The trusted Kalman filter (KF) is used for optimal estimation of the state and requires a linear dynamics matrix (state transition matrix). There are general extensions or modifications to the KF to deal with non-linearity, such as the extended Kalman filter (EKF), unscented Kalman filter (UKF), particle filter, etc. Subsection. 7.2.3 of Ch. 7 outlines the EKF implementation for position estimation.

6.2.2 Extended Kalman Filter

An EKF may also be used to estimate state variables, model parameters and conduct system identification by augmenting the state vector to include the model parameters. This section discusses an EKF for estimating model parameters for the augmented Nomoto ship-steering model (see Eq. (5.19)).

First, the model chosen for identification is the augmented Nomoto model described by Eq. (5.19), but with modifications. Since there is no encoder on the

rudder to measure the rudder angle, the model is reduced slightly to become

$$\underbrace{\begin{bmatrix} \psi_{k+1} \\ r_{k+1} \\ x_{k+1} \\ y_{k+1} \\ v_{k+1} \\ \hline T_{\text{d, yaw},k+1} \\ K_{\text{d, yaw},k+1} \end{bmatrix}}_{\underline{\mathbf{x}}_{k+1}} = \underbrace{\left[\begin{array}{ccccccc} 1 & \Delta T & 0 & 0 & 0 & 0 & 0 \\ 0 & T_{\text{d,yaw}} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & \Delta T \sin(\psi_k) & 0 & 0 \\ 0 & 0 & 0 & 1 & \Delta T \cos(\psi_k) & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{array}\right]}_{\underline{\Phi}_k} \underbrace{\begin{bmatrix} \psi_k \\ r_k \\ x_k \\ y_k \\ v_k \\ \hline T_{\text{d,yaw},k} \\ K_{\text{d,yaw},k} \end{bmatrix}}_{\underline{\mathbf{x}}_k} + \underbrace{\begin{bmatrix} 0 \\ K_{\text{d,yaw},k} \\ 0 \\ 0 \\ 0 \\ \hline 0 \\ 0 \end{bmatrix}}_{\underline{\Gamma}_k} u_k \tag{6.10}$$

There are three main differences; 1) the yaw rate time constant $T_{\text{d,yaw},k+1}$ is now a part of the augmented state vector, 2) the rudder lag coefficients a_0 and b_0 have been removed since we have no estimate of measure of u_k, and 3) the input vector has changed to include $K_{\text{d,yaw}}$ instead of the state transition matrix. The rudder lag coefficients were initially removed, to simplify the implementation of the state space model when applied in real-life in early experiments and then could be added back later. The model assumes that the yaw rate will change with a lag proportional to a time constant. This time constant is a model parameter that can be estimated. It is important to note that a key assumption here is that the velocity remains relatively constant.

Experimental Results

In order to estimate the model parameter, the Slug 3 ASV was used to collect experimental data logged onboard the vehicle. This data included, but was not limited to the GPS position of the vehicle (transformed into the local tangent

plane), the course over ground (COG) angle, and the rudder angle command signal (in radians). Fig. 6.4 shows the rudder command signal in radians versus time. Fig. 6.5 shows the resulting T_{yaw} estimate versus time.

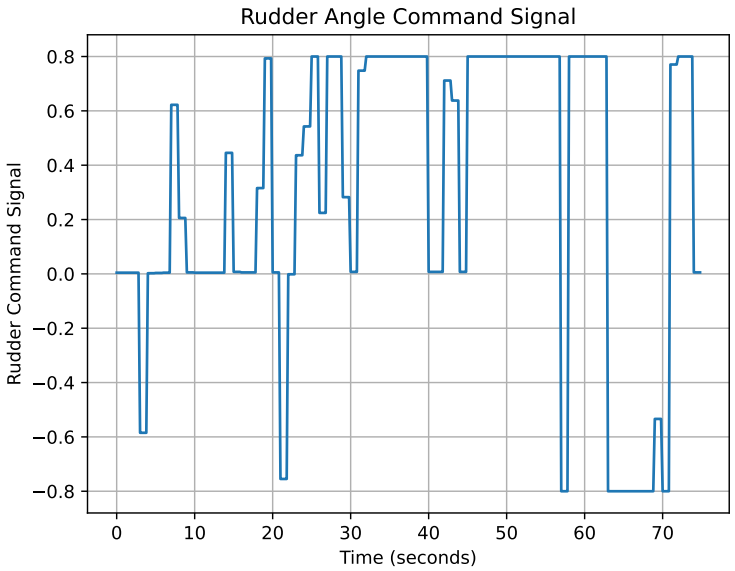


Figure 6.4: An example of rudder angle command signal (radians) versus time recorded onboard the Slug 3 ASV.

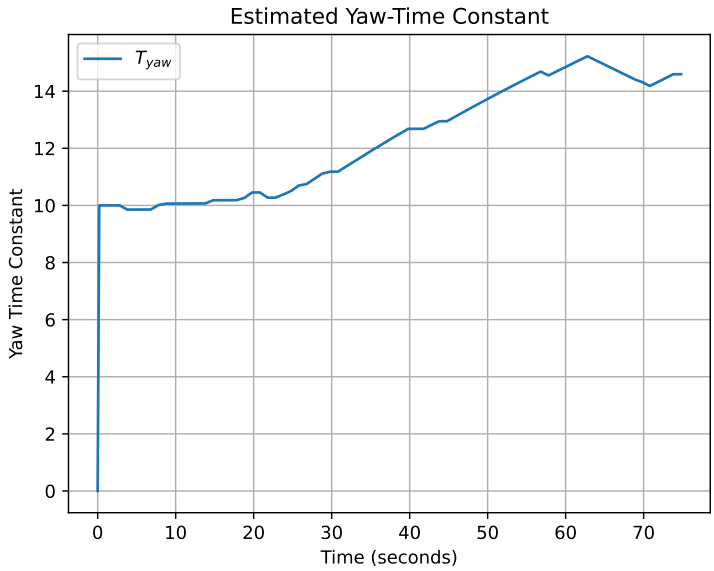


Figure 6.5: An example of the estimated yaw time constant $T_{\text{d,yaw}}$, converging to ~ 14.2 seconds.

6.3 Conclusion and Caveats

This chapter explained how to use the ARX method to estimate the rudder servo model parameters in simulation. This method is an effective way to identify small systems. For more complicated systems that rely on actuators such as servos in addition to having propulsion, an EKF can be more applicable. Such an EKF was introduced and was used to estimate the yaw rate time constant of the augmented Nomoto ship steering model.

One caveat to the ARX method in the context of the rudder servo example is that it assumes that the rudder angle can be measured. This might not be the case for all actuators, and in fact the Slug 2 and Slug 3 do not have encoders to measure the rudder angle. Further more it is difficult to use a state-space model, such as the augmented Nomoto ship steering model with an ARX formulation. This is partly why the EKF is so useful. Ch. 7 and Ch. 9 further highlight this

point.

Chapter 7

Guidance Navigation and Control

While this work describes a "system" and its applications with explanations on design and trade-offs, the central contribution lies within the guidance, navigation, and control (GNC) of the vehicle; to do it without autonomous trajectory following, the "system" cannot complete its mission. Guidance involves path planning and determining where the vehicle should go (often by the user). Navigation is largely focused on localization and estimating where the vehicle currently is and how the vehicle is oriented, based on sensor measurements. Control is the process of tracking and driving the vehicle or system towards the desired path or state. Regarding the guidance component of GNC, this chapter will discuss methods of trajectory generation to form a path for the vehicle to follow. Attitude estimation was previously discussed earlier in Ch. 4 as a standalone chapter. The control part of this chapter deals with rudder control, and how it relates to trajectory tracking. The main point of this chapter is to explain how to design a system for path-following that doesn't require a large amount of computation.

7.0.1 Connection to the Overarching Theme

This chapter begins to relate how the researcher or oceanographer can control an ASV to execute specific maneuvers without specific knowledge of the underlying control architecture. The researchers' interactions with the guidance navigation and control system essentially boils down to them selecting specific waypoints for the vehicle to travel through or near. In later chapters, the use of the GNC system will be connected to autonomous path planning informed by subsequent spatial estimators.

7.1 Trajectory Generation

Trajectory generation involves specifying the desired vehicle state (orientation, rotational rates, velocity, and position) in relation to a desired path based on the waypoints that have previously been selected. There are many different ways to generate trajectories for vehicles. The most simple approach is to connect waypoints with straight lines. For slow-moving vehicles, this can be sufficient and is certainly simple to implement. A more complicated approach is to use splines to connect the waypoints. Previous work such as [15], [43], and [65] use cubic splines and Bézier curves for trajectory generation of autonomous vehicles, usually UAVs.

7.1.1 Trajectory Tracking

Linear Path Segments

The GNC algorithm requires m number of waypoints defined in a set of North and East points. Trajectory tracking is applied to path segments defined by two waypoints for linear segments and three waypoints for curved segments, respectively. Tracking is primarily based on the cross track error between the vehicle's

current location and the closest point on the path. Let the matrix $\underline{W} \in \mathbb{R}^{m\times 2}$ represent all of the desired waypoints for the vehicle to visit with each row being a discrete point on the local tangent plane, $\mathbf{w}_i = \begin{bmatrix} x_i & y_i \end{bmatrix}$ where $i \in \mathbb{Z}, [1, 2, \ldots, m]$ is the index of the rows of $\underline{W}$. A Serret-Frenet[1] reference frame is used to help simplify how the system is described. The state of the vehicle in the Serret-Frenet frame, shown in Fig. 7.2 is expressed as

$$\underline{\mathbf{x}}^{\mathrm{SF}} = \begin{bmatrix} e & \dot{e} & \ddot{e} \end{bmatrix} \tag{7.1}$$

where e is the cross track error, or the distance from the vehicle to the closest point on the path within the Serret-Frenet frame, $\dot{e}$ is the velocity of the cross track error, and $\ddot{e}$ is the acceleration.

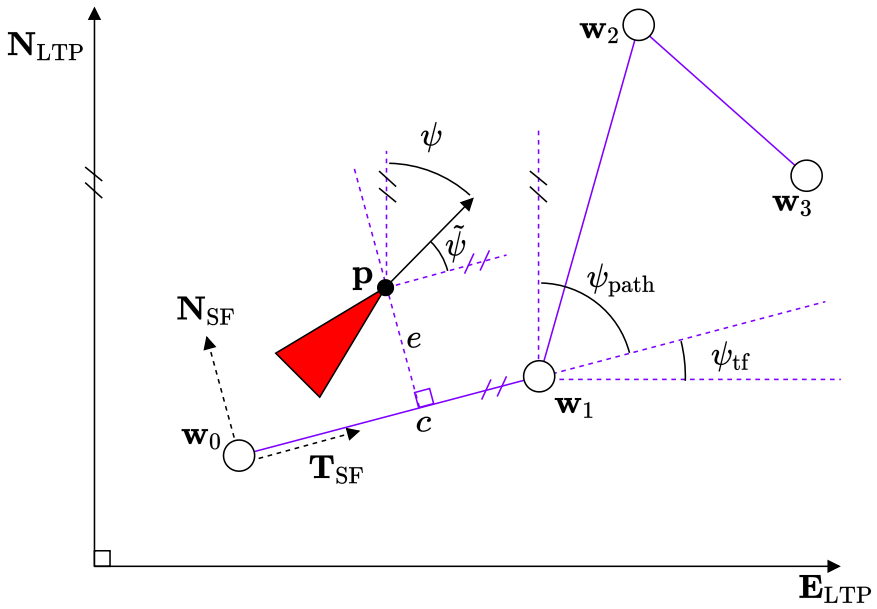


Figure 7.1: Cross track error e between the vehicle and the closest point on the path within the Serret-Frenet frame, along a linear path segment.

[1]The Serret-Frenet (sometimes referred to as Frenet-Serret) formulas are defined by three unit vectors: a tangent, normal, and bi-normal. These are used to help describe particle kinematics in $\mathbb{R}^3$

The terms in Fig. 7.1 are defined as follows:

$$\begin{aligned}
\mathbf{N}_{\text{LTP}} &= \text{North vector of the local tangent plane} \\
\mathbf{E}_{\text{LTP}} &= \text{East vector of the local tangent plane} \\
\mathbf{N}_{\text{SF}} &= \text{normal vector in the Serre-Frenet frame} \\
\mathbf{T}_{\text{SF}} &= \text{tangent vector in Serret-Frenet frame} \\
\psi &= \text{heading angle in the local tangent plane} \\
\tilde{\psi} &= \text{angle between } \mathbf{T}_{\text{SF}} \text{ and the heading vector} \\
\psi_{\text{path}} &= \text{the path angle in the local tangent plane} \\
\psi_{\text{tf}} &= \text{the angle between } \mathbf{T}_{\text{SF}} \text{ and } \mathbf{E}_{\text{LTP}}
\end{aligned}$$

The vehicle state in eq 7.1 can be rewritten as

$$\underline{\mathbf{x}}_k^{\text{SF}} = \begin{bmatrix} e_k^{\text{SF}} & v_k^{\text{SF}} & a_k^{\text{SF}} \end{bmatrix} \tag{7.2}$$

where again, e_k^{SF} is the cross track error, v_k^{SF} is the velocity of the cross track error, and a_k^{SF} is the acceleration, but all corresponding to a discrete point in time. The cross track error calculation depends on the definition of the path. For linear segments of the path, an analytical solution to calculate e_k^{SF} exists. This can be achieved by projecting the vehicle position onto the vector connecting two waypoints; (see Fig 7.1).

$$\mathbf{c} = (\mathbf{p} - \mathbf{w}_0)^{\text{T}}(\mathbf{w}_1 - \mathbf{w}_0) + \mathbf{w}_0 \tag{7.3}$$

It should be noted that the vectors such as $\mathbf{w}_0$ and $\mathbf{p}$ implicitly are vectors with their origin of waypoint $\mathbf{w}_0$. The projection is applied, and finally the elements of $\mathbf{w}_0$ are added back. This gives the closest point on the line segment, $\mathbf{c}$ and the

cross-track error can be calculated as follows:

$$e_k = (\mathbf{p} - \mathbf{c})^{\mathrm{T}} \mathbf{N}_{\mathrm{SF}} \tag{7.4}$$

This is the projection of the cross-track error onto the normal Serret-Frenet unit vector.

Arcing Path Segments

For non-linear segments, as shown in Fig 7.2, an analytic solution for finding the closest point on a b-spline or bezier-curve does not exist [14] and a numerical solution is needed. In this thesis this problem is avoided by ensuring that circular arcs are employed near the vertices of two connecting linear path segments. An analytic solution exists to find the closest point on a circular arc, with small increase in computing. The question may arise; why not only use straight line segments to define the entire path? The answer is that this produces an abrupt change in path angle, causing a possibly large control system response. The controller will generate a consequently larger control effort to counter the discontinuity, possibly saturating an actuator, in this case a rudder servo. This can consume excess power and can have other negative effects such as increasing the chance of actuator damage. The control effort can limited by introducing a slow change in path angle which can be done with curved segments near the vertices of the path.

To find e_k along a curved segment the closest point on the curve must be calculated. This requires the location of the pivot point of the arc of the curve and a threshold away from the vertex to define the start of the curve. The threshold is based on the smallest turning radius of the vehicle, to limit the cross track error. If the curve threshold is too small, the radius of the curve can be smaller than the turning radius of the vehicle, resulting in too much overshoot. Fig 7.3 shows how

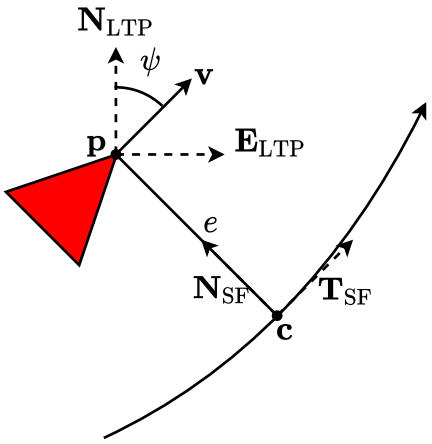


Figure 7.2: Cross track error e between the vehicle and the closest point on the path within the Serret-Frenet frame along a curved path segment.

the curve threshold changes depending on the angle, θ between $(\mathbf{w}_{i-1} - \mathbf{w}_i)$ and $(\mathbf{w}_i - \mathbf{w}_{i+1})$.

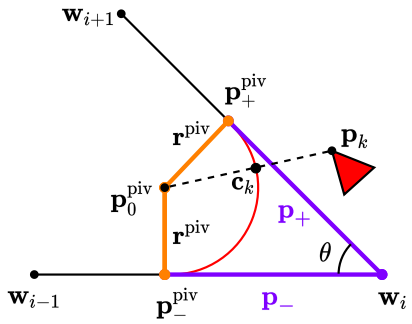


Figure 7.3: Geometry of an arcing segment and the closest point $\mathbf{c}_k$ on the arc to the position of the vehicle, $\mathbf{p}_k$

The magnitudes of the vectors and angle that describe the arc can be expressed by the following equation:

$$||\mathbf{p}_-|| = ||\mathbf{p}_+|| = \frac{\mathbf{r}^{\text{piv}}}{\tan\left(\frac{\theta}{2}\right)} \tag{7.5}$$

where $\mathbf{p}_-$ and $\mathbf{p}_+$ represent the beginning and ends of an arc, and r^{piv} is at least the minimum turning radius of the vehicle. The change between calculating the

closest point on the linear segment and calculating the closest point on the curved segment occurs once $||\mathbf{c}-\mathbf{r}^{\text{piv}}|| \leq ||\mathbf{p}_-||$. Eqs 7.6 and 7.7 define the beginning and end points of the curve segment for a given vertex, $\mathbf{w}_i$.

$$\mathbf{p}_-^{\text{piv}} = ||\mathbf{p}_-||\hat{\mathbf{w}}_- \tag{7.6}$$

$$\mathbf{p}_+^{\text{piv}} = ||\mathbf{p}_+||\hat{\mathbf{w}}_+ \tag{7.7}$$

where

$$\hat{\mathbf{w}}_- = \frac{(\mathbf{w}_i - \mathbf{w}_{i-1})}{||\mathbf{w}_i - \mathbf{w}_{i-1}||} \tag{7.8}$$

and

$$\hat{\mathbf{w}}_+ = \frac{(\mathbf{w}_{i+1} - \mathbf{w}_i)}{||\mathbf{w}_{i+1} - \mathbf{w}_i||} \tag{7.9}$$

To find the pivot point, $\mathbf{p}_0^{\text{piv}}$, a unit norm vector stemming from $\mathbf{p}_-^{\text{piv}}$ is rotated by $\frac{\pi}{2}$ in the direction of the curve.

$$\mathbf{p}_0^{\text{piv}} = \underbrace{\begin{bmatrix} \cos\left(d\frac{\pi}{2}\right) & -\sin\left(d\frac{\pi}{2}\right) \\ \sin\left(d\frac{\pi}{2}\right) & \cos\left(d\frac{\pi}{2}\right) \end{bmatrix}}_{\mathbf{R}} \left[\mathbf{r}^{\text{piv}} \frac{\mathbf{w}_i - \mathbf{w}_{i-1}}{||\mathbf{w}_i - \mathbf{w}_{i-1}||}\right] + \mathbf{p}_-^{\text{piv}} \tag{7.10}$$

where $\mathbf{r}^{\text{piv}}$ can be calculated using the law of sines, as shown in eq 7.11, and $d = \pm 1$ determined by Eq. (7.12)

$$\mathbf{r}^{\text{piv}} = ||\mathbf{p}_-|| \frac{\sin\theta}{\sin\left(\frac{\pi}{2} - \theta\right)} \tag{7.11}$$

$$d = \text{sign}([\mathbf{w}_{i+1} - \mathbf{w}_i]_x^{\text{SF}} - [\mathbf{w}_i - \mathbf{w}_{i-1}]_x^{\text{SF}}) \tag{7.12}$$

Note that the rotation matrix $\mathbf{R}$ in Eq. (7.10) will evaluate to either

$$\mathbf{R} = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \tag{7.13}$$

or

$$\mathbf{R} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \tag{7.14}$$

In eq 7.12, $[\cdots]_x^{\text{SF}}$ indicates the x-component of the two-dimensional point with respect to the Serret-Frenet frame formed by w_i and w_{i-1}. It's important to note that the Serret-Frenet frame is comprised of the closest two waypoints within $\underline{W}$ to the vehicles location, p_k.

With the pivot solved for, the closest point to the vehicle's location on the curve can be calculated, as shown in eq 7.15.

$$\mathbf{c}_k = \mathbf{p}_0^{\text{piv}} - \frac{\mathbf{r}^{\text{piv}}(\mathbf{p}_0^{\text{piv}} - \mathbf{p}_k)}{||\mathbf{p}_0^{\text{piv}} - \mathbf{p}_k||} \tag{7.15}$$

The cross track error can be found using eq 7.4 with respect to the Serret-Frenet frame. With the capability to calculate both the linear and curved segment closest point and cross track error a controller can be developed.

7.2 Control

7.2.1 PID Controller

The purpose of the PID controller for the GNC algorithm is to track the trajectory, such that the error between the desired state, $\underline{\mathbf{x}}_{\text{des},k}$ and the current state,

$\underline{\mathbf{x}}_k$ at a given time step is minimized. Rather than use the full state in Eq. (5.7), the error state, Eq. (7.2) is employed. This has the advantage of approximating the model for the vehicle to be expressed as a double integrator. This is shown in fig 7.4, along with the digital controller.

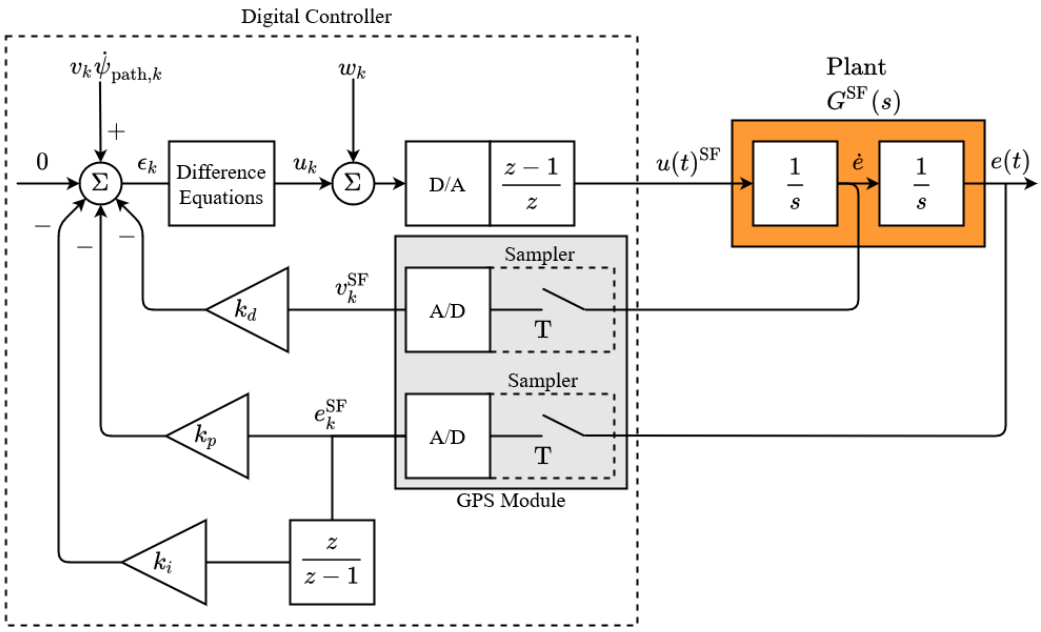


Figure 7.4: An approximation of the vehicle dynamics with respect to cross track error within the Serret-Frenet frame.

The digital PID controller uses feed-forward of the path curvature at the current time step, which is defined as $v_k\dot{\psi}_{\text{path},k}$. This is based on [65], and is useful because the path curvature is known ahead of time. The feed-forward accounts for the acceleration due to the path curvature. The difference equation block in Fig. 7.4 matches the continuous equivalent in [65] and is written explicitly as

$$u_k = v_k\dot{\psi}_{\text{path},k} + K_p e_k^{\text{SF}} + K_i \frac{z}{z-1} e_k^{\text{SF}} \Delta T + K_d v_k^{\text{SF}} \tag{7.16}$$

where, again v_k is the scalar magnitude of the vehicle's velocity, and the v_k^{SF} is the velocity of the cross track error with respect to the Serret-Frenet frame. It should

be noted that the difference between the path angle and the vehicle heading angle is equivalent to the velocity of the cross track error for small angles. Thus, in units of meters per seconds, the rate of change of the cross track error is proportional to the difference in heading angle and path angle;

$$v_k^{\mathrm{SF}} \equiv v_k(\psi_{\mathrm{path},k} - \psi_k) \tag{7.17}$$

allowing Eq. (7.16) to be rewritten as

$$u_k = v_k\dot{\psi}_{\mathrm{path},k} + K_p e_k^{\mathrm{SF}} + K_i \sum_{k=0}^{k=N} e_k^{\mathrm{SF}}\Delta T + K_d v_k(\psi_{\mathrm{path},k} - \psi_k) \tag{7.18}$$

Recall that the simulation uses the discrete state space representation defined by the dynamics in Eq. (5.19), and that the only input in $\underline{\Gamma}_k$ associated with the yaw rate, r, of the state, $\underline{\mathbf{x}}_k$, is the rudder angle δ_k from Eq. (5.14). Given $\underline{\Phi}_k$ and $\underline{\Gamma}_k$, representing the discrete state space transition matrix and input matrix respectively, then by the definition of controllability,

$$\underline{C}_k = \begin{bmatrix} \underline{\Gamma}_k & \underline{\Phi}_k\underline{\Gamma}_k & \underline{\Phi}_k^2\underline{\Gamma}_k & \dots & \underline{\Phi}_k^{n-1}\underline{\Gamma}_k \end{bmatrix} \tag{7.19}$$

The dynamics with respect to the Serret-Frenet frame are considered, such that

$$\underbrace{\begin{bmatrix} \dot{e} \\ \ddot{e} \\ \dddot{e} \end{bmatrix}}_{\dot{\underline{\mathbf{x}}}^{\mathrm{SF}}} = \underbrace{\begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}}_{\underline{A}} \underbrace{\begin{bmatrix} e \\ \dot{e} \\ \ddot{e} \end{bmatrix}}_{\underline{\mathbf{x}}^{\mathrm{SF}}} + \underbrace{\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}}_{\underline{B}} u \tag{7.20}$$

is the continuous state space representation of the system, and, given that

$$\begin{bmatrix} \underline{\Phi}_k & \underline{\Gamma}_k \\ \underline{0} & \underline{I} \end{bmatrix} \approxeq \mathbf{expm}\left(\begin{bmatrix} \underline{A} & \underline{B} \\ \underline{0} & \underline{0} \end{bmatrix} \Delta T \right) \tag{7.21}$$

using a Taylor series expansion,

$$\underbrace{\begin{bmatrix} e_{k+1}^{\mathrm{SF}} \\ v_{k+1}^{\mathrm{SF}} \\ a_{k+1}^{\mathrm{SF}} \end{bmatrix}}_{\underline{\mathbf{x}}_{k+1}^{\mathrm{SF}}} = \underbrace{\begin{bmatrix} 1 & \Delta T & \frac{\Delta T^2}{2!} \\ 0 & 1 & \Delta T \\ 0 & 0 & 1 \end{bmatrix}}_{\underline{\Phi}_k^{\mathrm{SF}}} \underbrace{\begin{bmatrix} e_k^{\mathrm{SF}} \\ v_k^{\mathrm{SF}} \\ a_k^{\mathrm{SF}} \end{bmatrix}}_{\underline{\mathbf{x}}_k^{\mathrm{SF}}} + \underbrace{\begin{bmatrix} \frac{\Delta T^3}{3!} \\ \frac{\Delta T^2}{2!} \\ \Delta T \end{bmatrix}}_{\underline{\Gamma}_k^{\mathrm{SF}}} u_k \tag{7.22}$$

is the discrete equivalent, then the system *is* controllable, because

$$C_k^{\mathrm{SF}} = \begin{bmatrix} \underline{\Gamma}_k^{\mathrm{SF}} & \underline{\Phi}_k^{\mathrm{SF}} \underline{\Gamma}_k^{\mathrm{SF}} & \underline{\Phi}_k^{\mathrm{SF}^2} \underline{\Gamma}_k & \dots & \underline{\Phi}_k^{\mathrm{SF}^{n-1}} \underline{\Gamma}_k \end{bmatrix} \tag{7.23}$$

is full rank, and has no dependency on the value of k.

7.2.2 GNC Algorithm

The following sub-components have been established: **1)** a method to calculate a desired path and trajectory with linear and curved segments, **2)** a discrete PID controller for trajectory tracking, and **3)** a controllable system. Using all of these sub components, the following algorithm is introduced:

Algorithm 1 GNC Path Following

Get GPS path coordinates
$\mathbf{W}$ = Map GPS coordinates to a local tangent plane (LTP)
$\mathbf{a} = \mathbf{w}_{i=0}$
$\mathbf{b} = \mathbf{w}_{i=1}$
$\mathbf{c} = \mathbf{w}_{i=2}$
Calculate initial $\mathbf{p}_0^{\text{piv}}, \mathbf{p}_-^{\text{piv}}, \mathbf{p}_+^{\text{piv}}$
while $i < m_{\text{total}}$
    Get vehicle's current GPS position
    $\mathbf{p}$ = Map GPS positon to LTP
    $\mathbf{c} = proj_{\overrightarrow{\mathbf{ab}}}\overrightarrow{\mathbf{ap}} = \frac{\overrightarrow{\mathbf{ab}} \cdot \overrightarrow{\mathbf{ap}}}{|\overrightarrow{\mathbf{ab}}|^2}$
    If $||\mathbf{c} - \mathbf{p}_-^{\text{piv}}|| \leq ||\mathbf{p}_-||$
        $\mathbf{c} = \mathbf{p}_0^{\text{piv}} - \frac{\mathbf{r}^{\text{piv}}(\mathbf{p}_0^{\text{piv}} - \mathbf{p})}{||\mathbf{p}_0^{\text{piv}} - \mathbf{p}||}$ // Calculate closest point on curve
    $\mathbf{e}_k = ||\mathbf{p} - \mathbf{c}||$// Calculate cross track error
    Run PID controller
    If $||\mathbf{c} - \mathbf{p}_+^{\text{piv}}|| \leq$ Switch Threshold
        $i = i + 1$
        $\mathbf{a} = \mathbf{w}_i$ // Update the previous waypoint
        $\mathbf{b} = \mathbf{w}_{i+1}$ // Update the next waypoint
        $\mathbf{c} = \mathbf{w}_{i+2}$ // Update the waypoint after the next waypoint
    Calculate new $\mathbf{p}_0^{\text{piv}}, \mathbf{p}_-^{\text{piv}}, \mathbf{p}_+^{\text{piv}}$

Note that the *Switch Threshold* is the distance around a waypoint that represents how close the vehicle must be in order to update the next and previous waypoints. The GNC algorithm is implemented in a MATLAB simulation and the resulting vehicle position plot is shown in Fig. 7.5. This algorithm was in simulation, where the "true" vehicle model was the augmented Nomoto ship steering model.

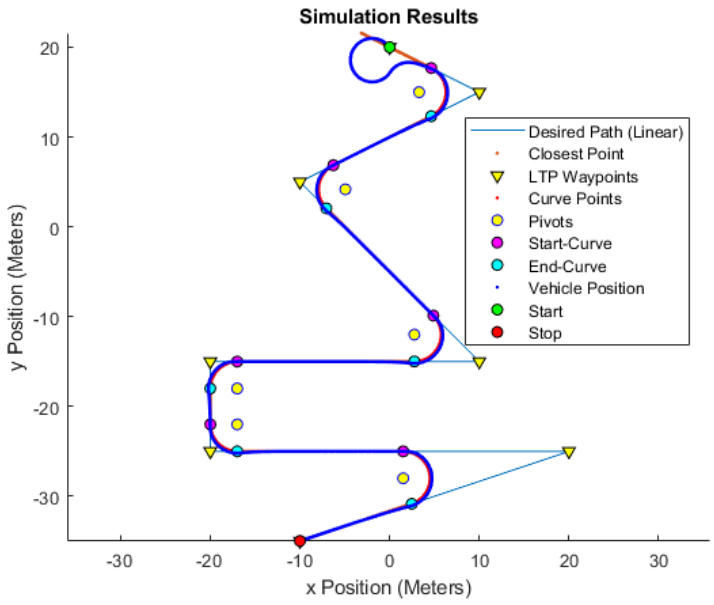


Figure 7.5: Simulation using the estimator in eq 5.8 and the discrete PID controller as part of the GNC algorithm

The results from the simulation show that the Alg. 1, is capable of following an arbitrary path defined by GPS waypoints. Hardware-in-the-loop testing and software-in-the-loop (SIL) simulations on the microcontroller for the vehicle, both confirm its functionality. SIL is also referred to as processor-in-the-loop (PIL) simulation. The next step is to use a an estimator, such as an Extended Kalman Filter (EKF) to estimate the position of the boat in between GPS updates (as they are relatively slow).

Choosing the Gains

The gains for the PID loop were chosen using the Ziegler–Nichols method. First, the derivative and integral gains were set to zero. Next, the proportional gain was increased from an arbitrary starting value until oscillations in the cross-track error were observed. The maximum proportional gain $k_{p,\max}$ was determined by the point at which oscillations were observed, along with the period of the os-

cillation $T_{\max}$. For no-overshoot $k_i = (0.4)k_{p,\max}T_{\max}$ and $k_d = (0.06\bar{6})k_{p,\max}T_{\max}$. In simulation other gains were also experimented with. Tuning PID gains is a rich area of discussion, and work by [33], [34], (and many others) provide a complete treatment (a brief discussion is provided in Appendix. A.0.1).

7.2.3 Position Estimation

Position measurements are less frequent than what is assumed in simulation. GPS modules generally have a new position, course over ground angle, speed, and other metrics available at one second intervals (1 Hz). This presents a problem for tracking the trajectory using the PID controller. If the vehicle moves too quickly, the controller may become unstable. Additionally, any guidance algorithm, such as Alg. 1 will have fewer points available to calculate the closest point on the trajectory. In the case of Alg. 1, if the vehicle turning radius is only a few meters, and the speed is greater than 1-2 meters per second, few curve points will be generated within the time it takes the vehicle to pass the curve segment.

Extended Kalman Filter

It is possible to estimate the position of the vehicle in between GPS measurements to facilitate having more frequent position updates. Position estimation can be accomplished *optimally* using an Extended Kalman Filter (EKF). Specifically, an EKF is required, because the vehicle dynamics are not linear.

The EKF is developed partially based on the vehicle dynamics shown in Eq. (5.19). The rudder lag is not considered in the model for the EKF to simplify the implementation and because an assumption (based on observation) is that the rudder lag is negligible for small angular corrections. This assumption is further based on the fact that the rudder is small, has low mass, and is easy to move

using a servo. The EKF uses the following model:

$$\underbrace{\begin{bmatrix}\psi_{k+1}\\ r_{k+1}\\ x_{k+1}\\ y_{k+1}\\ v_{k+1}\end{bmatrix}}_{\underline{\mathbf{x}}_{k+1}} = \underbrace{\begin{bmatrix}1 & \Delta T & 0 & 0 & 0\\ 0 & T_{\mathrm{d,yaw}} & 0 & 0 & 0\\ 0 & 0 & 1 & 0 & \Delta T\sin(\psi_k)\\ 0 & 0 & 0 & 1 & \Delta T\cos(\psi_k)\\ 0 & 0 & 0 & 0 & 1\end{bmatrix}}_{\underline{\Phi}_k} \underbrace{\begin{bmatrix}\psi_k\\ r_k\\ x_k\\ y_k\\ v_k\end{bmatrix}}_{\underline{\mathbf{x}}_k} + \underbrace{\begin{bmatrix}0\\ K_{\mathrm{d,yaw}}\\ 0\\ 0\\ 0\end{bmatrix}}_{\underline{\Gamma}_k} u_k \tag{7.24}$$

In Ch. 6 we demonstrated how Eq. (7.24) can be modified for the purposes of system identification. The discrete time, nonlinear model EKF, described in [36], is restated here for completeness. Let the stochastic discrete equation equation describing the vehicle's motion be written as

$$\underline{\mathrm{x}}_{k+1} = \underline{\mathrm{f}}_k(\underline{\mathrm{x}}_k, \underline{\mathrm{u}}_k) \tag{7.25}$$

where $\underline{\mathrm{x}}_k$ is the vehicle state with respect to the local tangent plane, and is equivalent to Eq. (5.7), $\underline{\mathrm{u}}_k$ is the control input, and $\underline{\mathrm{f}}_k()$ is a nonlinear vector function of the state, sometimes referred to as a state propagation vector function. It is defined as

$$\underline{\mathrm{f}}_k(\underline{\mathrm{x}}_k, \underline{\mathrm{u}}_k) = \begin{bmatrix}0\\ K_{\mathrm{d,yaw}}\mathrm{u}_k\\ v_k\sin(\psi_k)\Delta T\\ v_k\cos(\psi_k)\Delta T\\ 0\end{bmatrix} \tag{7.26}$$

The sampled nonlinear measurement is

$$\underline{\mathrm{y}}_k = \underline{\mathrm{h}}(\underline{\mathrm{x}}_k) + \underline{\mathrm{v}}_k \tag{7.27}$$

where $\underline{\mathrm{h}}$ is the observation, or output matrix.

The state transition matrix, $\underline{\Phi}_k$ is formed by the Jacobian comprised of the partial derivatives of the vector function of the state at index k, with respect to the state elements.

$$\underline{\Phi}_k = \frac{\partial \underline{\mathrm{f}}_k(\underline{\mathrm{x}}_k, \underline{\mathrm{u}}_k)}{\partial \underline{\mathrm{x}}_k} \tag{7.28}$$

Within the prediction step, the prediction of the step is defined as

$$\hat{\underline{\mathrm{x}}}_{k+1}^{(-)} = \underline{\mathrm{f}}_k(\hat{\underline{\mathrm{x}}}_k, \underline{\mathrm{u}}_k) \tag{7.29}$$

and the covariance matrix is

$$\underline{\mathrm{P}}_{k+1}^{(-)} = \underline{\Phi}_k \underline{\mathrm{P}}_k \underline{\Phi}_k^{\mathrm{T}} + \underline{\Gamma}_k \underline{\mathrm{Q}}_k \underline{\Gamma}_k^{\mathrm{T}} \tag{7.30}$$

where $\underline{\Gamma}_k$ is the input matrix, as noted before, and $\underline{\mathrm{Q}}$ is the initial guess of the variance for the stochastic terms. $\underline{\Gamma}_k$ may also be defined as

$$\underline{\Gamma}_k = \frac{\partial \underline{\mathrm{f}}_k(\hat{\underline{\mathrm{x}}}_k, \underline{\mathrm{u}}_k)}{\partial \underline{\mathrm{u}}_k} \tag{7.31}$$

along with

$$\underline{\mathrm{H}}_k = \frac{\partial \underline{\mathrm{h}}_k(\hat{\underline{\mathrm{x}}}_k^{(-)})}{\partial \underline{\mathrm{x}}_k} \tag{7.32}$$

Table 7.1: Cross track error mean and standard deviation comparison between EKF and GPS-only navigation

	Mean (meters)	Standard Deviation (meters)
With EKF	0.009349	0.274763
GPS only	-0.046454	0.800354

The optimal gain matrix of the update step is as follows

$$\underline{\mathrm{K}}_{k+1} = \underline{\mathrm{P}}_{k+1}^{(-)}\underline{\mathrm{H}}_{k+1}^{\mathrm{T}}\Big(\underline{\mathrm{H}}_{k+1}\underline{\mathrm{P}}_{k+1}^{(-)}\underline{\mathrm{H}}_{k+1}^{\mathrm{T}} + \underline{\mathrm{R}}_{k+1}\Big)^{-1} \tag{7.33}$$

where $\underline{\mathrm{R}}$ is the sensor noise covariance matrix. Next part of the update step is the state estimate.

$$\hat{\underline{\mathrm{x}}}_{k+1} = \hat{\underline{\mathrm{x}}}_{k+1}^{(-)} + \underline{\mathrm{K}}_{k+1}\Big(\underline{\mathrm{y}}_{k+1} - \underline{\mathrm{h}}_{k+1}(\hat{\underline{\mathrm{x}}}_{k+1}^{(-)})\Big) \tag{7.34}$$

Lastly, the covariance matrix is udpated

$$\underline{\mathrm{P}}_{k+1} = \Big(I - \underline{\mathrm{K}}_{k+1}\underline{\mathrm{H}}_{k+1}\Big)\underline{\mathrm{P}}_{k+1}^{(-)} \tag{7.35}$$

7.2.4 Results

Using the EKF position estimates extracted from Eq. (7.34), the trajectory tracking PID controller can be run at samples as fast as $\Delta T = 0.1$ seconds, rather than the one second GPS measurement intervals. A performance increase was observed in simulation, between strict GPS measurement navigation and estimated position navigation with Alg. 1. Table 7.1 shows the improved mean and standard deviation in meters of the vehicle cross track error along the path.

Fig. 7.6 and Fig. 7.7 show the position of the vehicle over time using GPS-only and EKF position estimates with GPS measurements respectively. It is important

to note that the curved segments of the path with feed-forward are significantly more visible when navigation with the EKF position estimates. Both plots employ Alg 1, but when the EKF is used to provide position estimates, Alg 1 can be called at a faster rate.

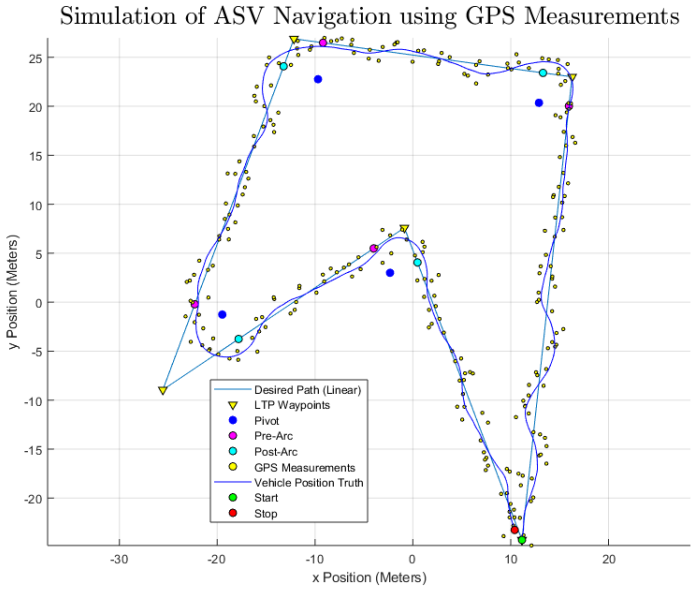


Figure 7.6: Simulation of trajectory following using only GPS measurements as input to the trajectory tracking PID controller. The pre-arc, pivot, and post-ark markers are shown in pink, dark blue, and cyan, respectively.

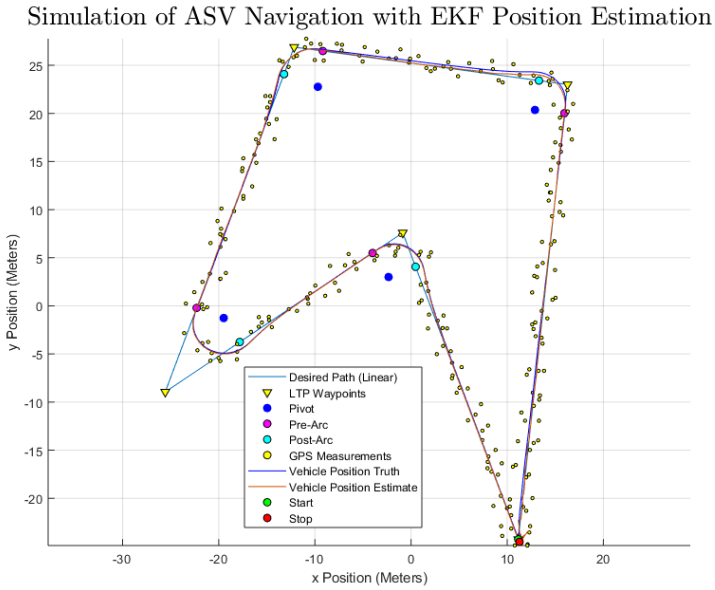


Figure 7.7: Simulation of trajectory following using EKF position estimates with GPS measurements as input to the trajectory tracking PID controller

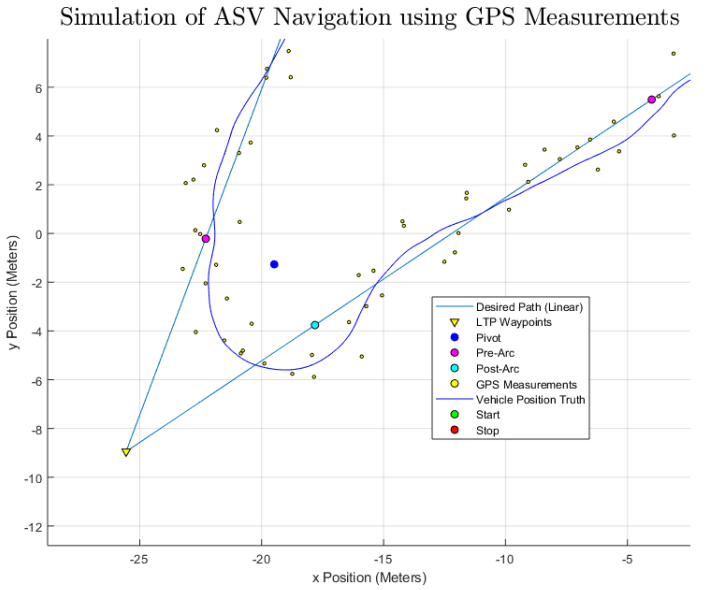


Figure 7.8: A portion of the simulation showing the vehicle position based on GPS measurements only. Trajectory tracking is noticeably oscillatory.

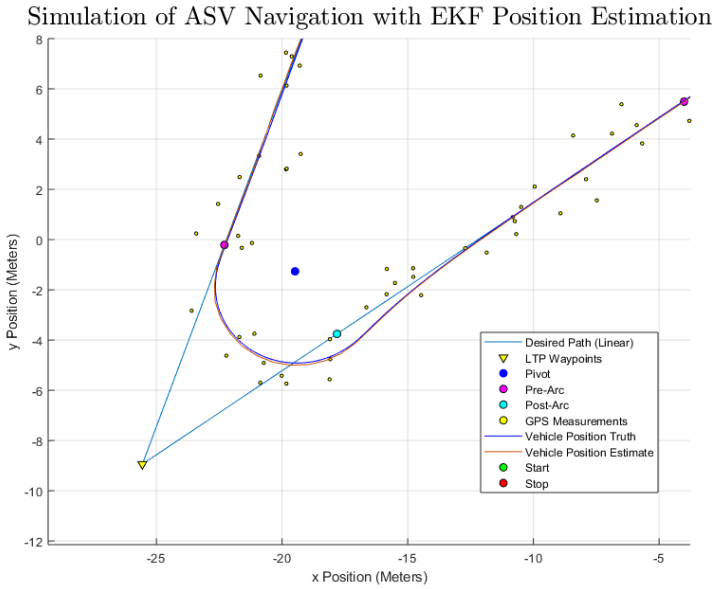


Figure 7.9: A portion of the simulation showing the vehicle position, the position estimates, and GPS measurements. Trajectory tracking is much smoother than if only using GPS measurements.

Fig. 7.8 and Fig. 7.9 show a closer look at a vertex along the desired path. There is considerably better tracking when using the EKF position estimates in between GPS measurements.

Remarks

This subsection explored the use of an EKF in simulation assuming the augmented Nomoto ship steering model. An improvement in tracking with respect to the mean cross track error was shown, compared to only using noisy GPS position measurements at low update rates. The formulation of the EKF was discussed in addition to the simulation results. In addition to being able to estimate position, the EKF can estimate the other state parameters. This is shown later in Ch. 9. The standard PID Ziegler–Nichols method was used to find gains for both the EKF and GPS-only simulations respectively. One idea worth additional investigation is to determine if there are a set of gains that allow the GPS-only navigation to outperform the EKF-based navigation in simulation (assuming use of Alg. 1). It should be noted that during simulation testing many different gains were experimented with outside of a formal procedure to explore this notion. Better GPS-only performance was not observed.

7.3 Conclusion and Caveats

This chapter showed how to design a path-following system that incorporated trajectory generation, control, and attitude estimation. Both simple straight-line trajectories, and more complicated arc-ed paths were examined. A PID controller was designed for tracking a trajectory with respect to the Serret-Frenet reference frame. The difference in performance between GPS-only navigation and EKF-based navigation was examined in relation cross-track error mean and standard

deviation (see Table 9.2).

It should be noted that splines could be used with the algorithms discussed in this chapter. They are more computationally demanding than simple straight lines, but can interpolate points and form paths that are smooth, continuous, and easier for certain vehicles (such as a boat) to follow. For a further discussion of b-splines see App. A.

Chapter 8

Intelligent Exploration

This chapter examines the question; how can an autonomous system decide where to take location-specific measurements within a given space (or field) to learn the most about the local environment? In this thesis the system is an ASV and it is taking depth measurements of a small body of water at various locations. However, this problem exists in various forms. For example, soil moisture is not easily detectable at certain depths without *in situ* measurements. It can also change over time. This means that even after surveying the field, different parts of the field will possess varying amounts of moisture. We can think of this field as being a sparsely sampled area of space and assume that the field attribute (soil moisture, depth, oxygen concentration, etc.) has a stationary distribution within some time-window. Furthermore, this chapter discusses fields that possess a field attribute with a *some* degree of auto-correlation; measurements that are close together are more likely to have a similar values than measurements that are far apart. For instance, consider if one was to take a set of measurements of the elevation of a hill. Two elevation measurements separated laterally by a few centimeters will likely be similar in value. However, two measurements separated laterally by a kilometer are less likely to be of similar value. This idea of auto-

correlation is observable in many different aspects of nature (mineral deposits, geological features, galactic super-structures, etc.).

In order to effectively explore an unknown or sparsely sampled field, it is useful to be able to estimate the field during exploration. An ASV may measure the field in a particular location, estimate the field, and then based on the estimate move to a location within the field to maximally improve the field estimate. There are many methods of spatial estimation, such as simultaneous localization and mapping (SLAM) and Gaussian Process Regression (GPR). SLAM in particular has many variations in and of itself, including acoustic, audio-visual, collaborative, biologically inspired, EKF, and many more. A classical and effective method of spatial estimation is ordinary kriging.

In this chapter we discuss: 1) how to construct a Gaussian random field (GRF) to simulate a realistic field (initially unknown to the ASV), 2) ordinary kriging, 3) a novel variation of kriging meant to decrease the computation time to estimate the field, 4) GPR, 5) simulation results, and 6) an intelligent path planner to optimize exploration based on field estimates.

8.0.1 Connection to the Overarching Theme

This chapter highlights the algorithms for intelligent exploration that yield a reliable estimate of an auto-correlated sparsely sampled phenomenon. The idea of intelligent exploration within this thesis is separated into path planning and spatial estimation. A key focus is placed on spatial estimation, because there are a number of different methods to estimate a field based on a growing set of measurements. In relation to the oceanographer/researcher example that has been discussed throughout this thesis, we assume that there is a field attribute sensor that provides a measured value of a field at different locations. The extent to

which field measurements are related by distance to one-another is discussed in detail along algorithms that take advantage of field measurement auto-correlation. These ideas can be of particular interest to researchers studying different environmental attributes as they are generally intended in the estimated field.

8.1 Ordinary Kriging

This section provides an explanation of ordinary kriging and how it may be used to inform path planning for an ASV. [83] and [85] both outline how ordinary kriging works. A brief overview based on these works is presented here for completeness. [85] in particular shows how ordinary kriging may be improved with regard to computational complexity. Ordinary kriging (OK) is classified as a best linear unbiased estimator (BLUE) of a random field in this case (and in general). Put another way, we are trying to estimate an attribute of a field, such as elevation, depth, or an amount of something that is spatially auto-correlated. The field is assumed to be a random field with some spatial correlation. For example, assume the goal is to estimate the distribution of gold in a given area. If the field is auto-correlated to some degree, then measuring the amount of gold in two places that are very close together will result in a *similar* measurement; the amount of gold measured is likely to be a similar amount. Now take two measurements of the field that are far apart, and the two measurements will likely be very different. It is important to realize that this is only true if the field is auto-correlated. If there is zero auto-correlation, then any two measurements, regardless of the distance have no relation in measurement value. The key components of ordinary kriging that are discussed below include forming an empirical variogram, fitting a model of the variogram to the empirical variogram, forming a covariance matrix with the variogram model, and finally inverting the covariance matrix to solve a

system of equations and estimate or predict a field based on previously acquired measurements (including those used to form the variogram).

The specific scenario in which OK is discussed is as follows; measurements are taken by an ASV within the bounds of a 2-dimmensional area (the "field"). A measurement of the field may be of any specific field attribute (e.g: elevation, temperature, salinity, etc.). As n, the number of measurements increases, the estimate of the field produced by OK takes more computational time. The computational complexity of OK is $O(n^3)$. One of the biggest contributing factors for the computational complexity is the inversion of the covariance matrix, $\underline{\mathrm{C}}$. The covariance matrix represents the specific correlation between different measurements. The farther apart any two measurement locations are, the less correlated they will be.

The computational complexity associated with OK and inverting the covariance matrix poses a problem for embedded computation. If an ASV (or similar autonomous vehicle) is meant to continually take field measurements, estimate the field, and travel to locations with high variance, then eventually enough measurements are taken that the computation time becomes intractable. The ASV would cease motion to complete the inversion, resulting in limited exploration and reduced area covered. Alternatively, if the ASV is incorporating multi-threading or higher performance computing, then the cost is higher energy usage, which again impacts vehicle performance, reducing mission duration. Thus computational complexity negatively impacts the field exploration.

Fortunately, [85] shows that the computational complexity can be reduced to as little as $O(n^2)$ by implementing an incremental inversion method, $O(nr^2)$ with a recursive matrix inversion method, where the covariance is divided into sub-matrices with dimension $r \times r$, where $r << n$. This is referred to as fixed

rank kriging (FRK). Within this section, a novel incremental method is discussed: iterative inverse ordinary kriging (IIOK).

8.2 The Variogram

First, it is important to explain the empirical variogram, because it is used to quantify auto-correlation of field measurements, and later to form the covariance matrix. The empirical variogram is defined as

$$N(h) = \sum_{i=0,j=0}^{m-1,m-1} \begin{cases} 1 & \text{if } (h-\delta) \leq \|\mathbf{s}_i - \mathbf{s}_j\| \leq (h+\delta) \\ 0 & \text{otherwise} \end{cases} \tag{8.1}$$

$$\hat{\gamma}(h) = \frac{1}{2N(h \pm \delta)} \sum_{i=0}^{m-1} \sum_{j=0}^{m-1} |\tilde{\mathbf{Z}}(\mathbf{s}_i) - \tilde{\mathbf{Z}}(\mathbf{s}_j)|^2 \tag{8.2}$$

where $\tilde{\mathbf{Z}}(\mathbf{s}_i) = \tilde{\mathbf{Z}}(x_i, y_i)$, such that a generic position on the field is represented by $\mathbf{s}_i = \begin{bmatrix} x_i & y_i \end{bmatrix}$ and $N(h \pm \delta)$ is the number of field measurements locations within a range based on some spatial separation, h, also known as a lag, and tolerance, δ. The lags and tolerances together act as bins. The function, $N()$ is used to count how many field measurement locations lie within each lag accounting for some tolerance. The tolerance is necessary, because it is unlikely that the distances between any two field measurements will lie exactly on a specific lag value. $\tilde{\mathbf{Z}}(x_i, y_i)$ is the measured value of the field in the local tangent plane, where $\mathbf{Z}$ is a matrix of true field values. In reality there is some noise associated with taking field measurements. This noise is inherent to the measurement process:

$$\tilde{\mathbf{Z}}(x_i, y_i) = \mathbf{Z}(x_i, y_i) + v \tag{8.3}$$

where v is the sensor noise (usually assumed to be zero-mean and Gaussian).

The usefulness of $\hat{\gamma}(h)$ is limited, because it cannot be used to continuously infer the variance of an arbitrary distance. By definition, it is limited to discrete bins. It is more useful to have a continuous model, such that any distance between measurements can have an associated variance computed. This is where a model of the variogram is introduced; $\gamma(h)$ is fit to the empirical variogram. Here, a Gaussian model is chosen for fitting the empirical variogram using least-squares. For reference, the Gaussian model takes on the following form,

$$\gamma(h) = (s - n)\left(1 - e^{\left(-\frac{h^2}{r^2 a}\right)}\right) + n \tag{8.4}$$

s is known as the *sill*, and is the point in the variogram where the variance becomes non-monotonic (stops increasing). n is the *nugget*, and can be thought of as a variance offset for the variogram. r is the *range*, and is defined as the lag (or distance bin) where the variance stops being monotonic. a is an arbitrary scaling factor. In this thesis the term $r^2 a$ is treated as a single variable r_0.

It should be noted that there are other variogram models, including spherical, exponential, etc. For more details on variogram models, see [83].

8.2.1 Fitting the Variogram

As mentioned earlier, in order to approximate the covariance between any two points continuously, a statistical model is employed. The model is meant to act as a continuous analog to the empirical variogram. The Gaussian model is chosen for its simplicity in implementation. It is fit to the empirical variogram using least-squares to solve for the Gaussian model's coefficients. For similar techniques, [19] explores using weighted least-squares for fitting variograms of geo-statistical data. Least-squares is chosen as it provides a reasonable fit in the absence of other information about the field. Additionally, the Gaussian model

displays consistency in relation to random processes, especially if those processes posses a normal distribution.

Least-squares (LS) can be used to approximately solve for the coefficients (nugget, sill, range) of the Gaussian model. The Maclaurin series is used to approximate the exponential term of the Gaussian variogram model in Eq. (8.4) extended to m terms. As will be shown below, this leads to an intuitive method for fitting the variogram with a closed-form solution.

The Maclaurin series as a function of a single scalar distance, or lag h_i is:

$$\mathbf{f}(h_i) = e^{-\frac{h_i^2}{r_0}} = \sum_{k=0}^{\infty} \frac{\left(-\frac{h_i^2}{r_0}\right)^k}{k!} \tag{8.5}$$

where $r_0 \triangleq r^2 a$. Let $\mathbf{h}$ be a vector in $\mathbb{R}^{n\times 1}$ of lags such that

$$\mathbf{h}^{\mathrm{T}} = \begin{bmatrix} h_0 & h_1 & \cdots & h_{n-1} \end{bmatrix} \tag{8.6}$$

Recall that the empirical variogram uses discrete bins, given some tolerance δ. The elements of $\mathbf{h}^T$ increase uniformly such that $h_i - \mathrm{d}h = h_{i+1}$ for $i \in [0, 1, 2, ..., n-1]$, where $\mathrm{d}h$ is the distance between bins of the empirical variogram. Expanding Eq. (8.5) to m terms to approximate $\mathbf{f}(h_i) \approx \hat{\mathbf{f}}(h_i)$

$$\hat{\mathbf{f}}(h_i) = 1 - \frac{\left(\frac{h_i^2}{r_0}\right)}{1} + \frac{\left(\frac{h_i^4}{r_0^2}\right)}{2} - \frac{\left(\frac{h_i^6}{r_0^3}\right)}{6} + \cdots + \frac{\left(-\frac{h^2}{r_0}\right)^{m-1}}{(m-1)!} \tag{8.7}$$

Eq. (8.7) can be rewritten as

$$\hat{\mathbf{f}} = 1 - h_i^2 \frac{1}{r_0} + h_i^4 \frac{1}{2r_0^2} - h_i^6 \frac{1}{6r_0^3} + \cdots + \left(-h_i^2\right)^{m-1} \frac{1}{{r_0}^{m-1}(m-1)!} \tag{8.8}$$

Rewriting Eq. (8.8) as the dot product of two vectors

$$\hat{\mathbf{f}}(h_i) = \mathbf{b}_i\mathbf{x} = \underbrace{\begin{bmatrix} -h_i^2 & h_i^4 & -h_i^6 & \cdots & 1 \end{bmatrix}}_{\mathbf{b}_i} \underbrace{\begin{bmatrix} \frac{s}{r_0} \\ \frac{s}{2r_0^2} \\ \frac{s}{6r_0^3} \\ \vdots \\ n \end{bmatrix}}_{\mathbf{x}} \tag{8.9}$$

where s is the sill and n is the nugget from the variogram model in Eq. (8.4) and

$$\mathbf{x}^{\mathrm{T}} = \begin{bmatrix} \frac{s}{r_0} & \frac{s}{r_0^2} & \frac{s}{r_0^3} & \cdots & n \end{bmatrix} \tag{8.10}$$

with

$$\mathbf{b}_i = \begin{bmatrix} -h_i^2 & h_i^4 & -h_i^6 & \cdots & 1 \end{bmatrix} \tag{8.11}$$

By combining Eq. (8.11) and Eq. (8.6), we have the following matrix form:

$$\mathbf{H} = \begin{bmatrix} \mathbf{b}_0 \\ \mathbf{b}_1 \\ \cdots \\ \mathbf{b}_{n-1} \end{bmatrix} = \begin{bmatrix} -\mathbf{h}^2 & \mathbf{h}^4 & -\mathbf{h}^6 & \mathbf{1} \end{bmatrix} \tag{8.12}$$

where $\mathbf{1}$ is an $n \times 1$ vector of ones. Assume that there exists a vector $\mathbf{v}(\mathbf{h})$ such that each element of $\mathbf{v}$ is the variogram function evaluated at h_i. By combining

Eq. (8.10) and Eq. 8.12 we obtain the following system of equations:

$$\mathbf{v}(\mathbf{h}) = \begin{bmatrix} \gamma(h_0) \\ \gamma(h_1) \\ \vdots \\ \gamma(h_{n-1}) \end{bmatrix} = \mathbf{H}\mathbf{x} \tag{8.13}$$

It should be noted that at this point we don't have access to the vector $\mathbf{v}$ representing the Gaussian variogram model evaluated at each element of $\mathbf{h}$. However, if we substitute $\mathbf{v}$ for the empirical variogram vector $\hat{\gamma}(\mathbf{h})$, then we may estimate the elements of $\mathbf{x}$ containing the unknown parameters of the Gaussian variogram model, $\gamma(\mathbf{h})$. By combining Eqs. (8.4)$\sim$(8.10) we estimate $\mathbf{x}$ using least-squares:

$$\hat{\mathbf{x}} = (\mathbf{H}^{\mathrm{T}}\mathbf{H})^{-1}\mathbf{H}^{\mathrm{T}}\hat{\gamma}(\mathbf{h}) \tag{8.14}$$

After estimating $\hat{\mathbf{x}}$, we can further solve for s, n, and r_0. An example of applying this to a simulated field is shown in Fig. 8.1. For more examples of the application of this technique, see appendix A.2.

Note that non-Gaussian variogram models may be chosen. If, for example a radial basis function is used instead, then this spatial estimation method may begin to resemble Gaussian process regression [55]. Another note is that a polynomial, or other function may be fit to the empirical variogram. Regardless of the model function used, the main assumption is that the variance between points increases as the distance between points grows up until a set distance: the range, r. The variance beyond the range is expected to stop growing and remain regularly constant. The range therefore represents the distance beyond which measurement points are essentially uncorrelated. There are of course exceptions to this as well, namely periodic covariance functions; these can be used for fields that have

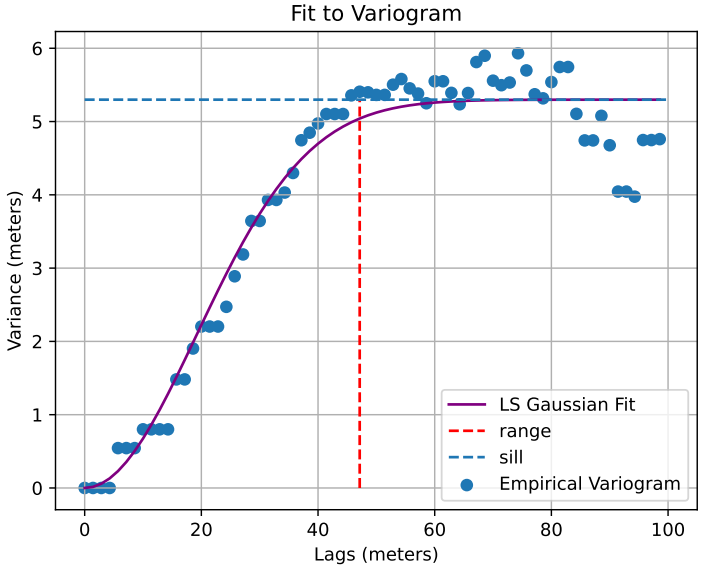


Figure 8.1: An example of a Gaussian variogram model, fit to an empirical variogram. Note that the discrete bins of the Empirical variogram are visible.

patterns that are known to be periodic in structure.

With a fitted variogram model, the covariance matrix can be formed and used to predict the values at all unmeasured locations within the field. Let $\mathbf{s}_i = \begin{bmatrix} x_i & y_i \end{bmatrix}$ be a single location on the local tangent plane defining the field, with x and y-coordinates. Each element of the covariance matrix is based on the γ of the distance between two measurement-locations;

$$\mathbf{C}(\mathbf{s}_i, \mathbf{s}_j) = \gamma\left(|| \underbrace{\begin{bmatrix} x_i, y_i \end{bmatrix}}_{\mathbf{s}_i} - \underbrace{\begin{bmatrix} x_j, y_j \end{bmatrix}}_{\mathbf{s}_j} || \right) \tag{8.15}$$

where $\mathbf{C} \in \mathbb{R}^{\ell \times \ell}$, and ℓ is the number of measurements. For all points, $\mathbf{p}_i = \begin{bmatrix} x_i & y_i \end{bmatrix}$ in the field, represented by an $m \times n$ matrix, $\mathbf{Z}$,

$$\mathbf{d}_i = \begin{bmatrix} \gamma(||\mathbf{s}_0 - \mathbf{p}_i||) & \dots & \gamma(||\mathbf{s}_{m-1} - \mathbf{p}_i||) \end{bmatrix}^{\mathrm{T}} \tag{8.16}$$

where $\mathbf{d} \in \mathbb{R}^{\ell \times 1}$ acts as a proximity, or spatial sensitivity vector for the point $\mathbf{p}_i$. The kriging weights, $\lambda_{\mathbf{p}_i}$ and Lagrangian multiplier, $\eta_{\mathbf{p}_i}$ can be found by augmenting the covariance matrix with vectors of ones $\mathbf{1} \in \mathbb{R}^{\ell \times 1}$ to solve the system of equations, as shown in both [83] and [28], such that

$$\begin{bmatrix} \lambda_{\mathbf{p}_i} \\ \eta_{\mathbf{p}_i} \end{bmatrix} = \begin{bmatrix} \mathbf{C}^{-1} & \mathbf{1} \\ \mathbf{1}^{\mathrm{T}} & 0 \end{bmatrix} \begin{bmatrix} \mathbf{d}_i \\ 1 \end{bmatrix} \tag{8.17}$$

where,

$$\lambda_{\mathbf{p}_i} = \mathbf{C}^{\text{-}1}\mathbf{d}_i + \mathbf{1} \tag{8.18}$$

and

$$\eta_{\mathbf{p}_i} = \mathbf{1}^{\mathrm{T}}\mathbf{d}_i \tag{8.19}$$

impose the constraint for the unbiased condition of this estimate. This means that $\lambda_{\mathbf{p}_i} - \mathbf{C}^{\text{-}1}\mathbf{d}_i = \mathbf{1}$ and $\eta_{\mathbf{p}_i} - (\mathbf{1}^{\mathrm{T}}\mathbf{d}_i) = 0$. Finally, the point associated with the field estimate matrix, $\hat{\mathrm{Z}}$ is found

$$\hat{\mathbf{Z}}(\mathbf{p}_i) = \begin{bmatrix} \mathbf{Z}(\mathbf{s}_0) & \dots & \mathbf{Z}(\mathbf{s}_{m-1}) \end{bmatrix}^{\mathrm{T}} \lambda_{\mathbf{p}_i} \tag{8.20}$$

$$\mathrm{var}(\hat{\mathbf{Z}}(\mathbf{p}_i)) = \begin{bmatrix} \mathbf{d}_{\mathbf{p}_i} & 1 \end{bmatrix} \begin{bmatrix} \lambda_{\mathbf{p}_i} \\ \eta_{\mathbf{p}_i} \end{bmatrix} \tag{8.21}$$

A useful feature of this method includes calculating the variance of the field estimate at $\mathbf{p}_i$ as shown in Eq. (8.21). A more detailed description is given in [83], upon which the above was based.

8.2.2 Iterative Covariance Matrix Inverse Update

The initial covariance matrix, $\mathbf{C}_0$, can be calculated from an initial batch of measurements. A speedup is discussed for calculating the inverse, $\mathbf{C}_k^{-1}$ after making additional measurements using a variation of the Sherman-Morrison formula for small-rank perturbations (see [39]). Eqs. (8.22 - 8.24) represent how to apply the formula to the covariance matrix. Using this matrix inversion technique with ordinary kriging is refereed to as iterative inverse ordinary kriging (IIOK) within this thesis.

The covariance matrix for each new measurement is described using the following equation,

$$\mathbf{C}_{k+1} = \begin{bmatrix} \mathbf{C}_k & \alpha \\ \alpha^{\mathrm{T}} & \beta \end{bmatrix} \tag{8.22}$$

where α is the newest, and therefore last column of $\mathbf{C}_k$, and corresponds to a new field measurement. With each measurement the covariance matrix grows in size. The inverse, $\mathbf{C}_{k+1}^{-1}$ is then given by

$$\mathbf{C}_{k+1}^{-1} = \begin{bmatrix} \mathbf{C}_k^{-1} + \frac{1}{v}\mathbf{C}_k^{-1}\alpha\alpha^{\mathrm{T}}\mathbf{C}_k^{-1} & \frac{1}{v}\mathbf{C}_k^{-1}\alpha \\ -\frac{1}{v}\alpha^{\mathrm{T}}\mathbf{C}_k^{-1} & \frac{1}{v} \end{bmatrix} \tag{8.23}$$

where $\mathbf{C}_k^{-1}$ is the inverted covariance matrix at the previous measurement index, and

$$v = \beta - \alpha^{\mathrm{T}}\mathbf{C}_k^{-1}\alpha \tag{8.24}$$

Note that, $\beta = n$, the nugget, may be zero depending on the variogram. This corresponds to calculating the correlation of the same point in Eq. (8.15) with

$h = 0, \implies \gamma(h) = n$, from Eq. (8.4). The purpose of using this variation of the Sherman-Morrison formula in Eq. (8.23) is to avoid the $O(n^3)$ computational complexity and resulting cubic computation time.

Remarks on the Update Rate

It is noted in [38] that the Sherman-Morrison formula is susceptible to numerical rounding errors. This sensitivity can be observed to a varying extent depending on the rate that the field estimate is updated, or the field estimate update rate. For instance, assume that a field is re-estimated for every new measurement collected. It should be apparent that the MSE versus the number of measurements over time will drop at a faster rate than compared to re-estimating the field using every n-th new measurement. Attempting to apply IIOK at low rates, implying the addition of a batch of a large number of new measurements yields a slower decrease in MSE than a compared to a higher rate of field estimate updates. The effect of field estimate update rate for IIOK is shown in Fig. 8.2. This shows how the field estimate update rate effects the MSE signals of IIOK.

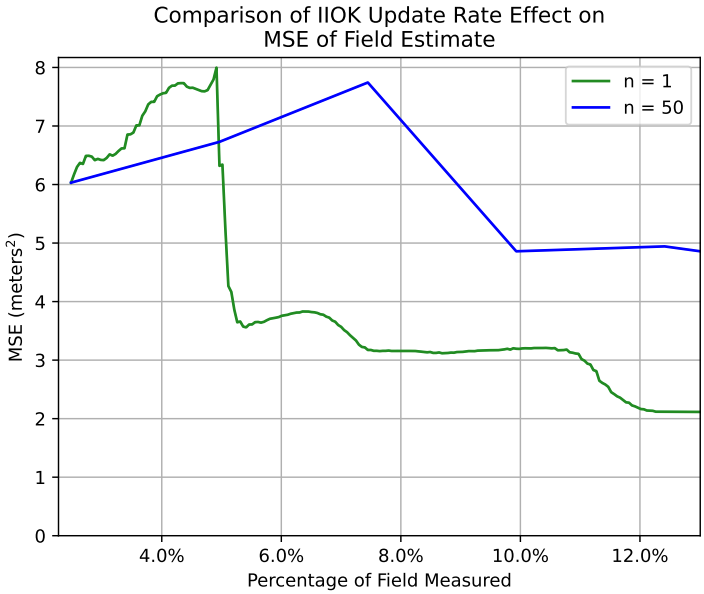


Figure 8.2: A comparison of MSE versus the number of points measured in a field. The MSE signals correspond to different update rates $1/n$, where n represents the number of new measurements before a new estimate is calculated using every nth measurement. The higher frequency update rates show a faster decrease in MSE.

8.3 Gaussian Process Regression

Within the context of this thesis Gaussian Process Regression (GPR) can be thought of as a generalized version of kriging. GPR has been used in a wide variety of applications, including real-time ground segmentation for autonomous vehicles [13], increasing the resolution of single-images [41], and optimal exploration trajectory planning for autonomous vehicles [74]. In this section we will discuss GPR, explore some examples with varying dimensions, and examine a simulation example of spatial estimation using GPR.

8.3.1 Review of GPR

Consider the joint distribution:

$$p(\mathbf{f}, \mathbf{y}) = \mathcal{N}\left(\begin{bmatrix}\mathbf{f} \\ \mathbf{y}\end{bmatrix}; \quad \begin{bmatrix}\mathbf{a} \\ \mathbf{b}\end{bmatrix}, \begin{bmatrix}\mathbf{A} & \mathbf{C}^{\mathrm{T}} \\ \mathbf{C} & \mathbf{B}\end{bmatrix}\right) \tag{8.25}$$

where $\mathbf{f}$ and $\mathbf{y}$ are vectors whose elements represent unknown and measured values respectively. Each element of $\mathbf{y}$ is a measurement with noise: $y(x_i) \sim \mathcal{N}(f(x_i), \sigma_y)$. Put another way; we may assume that there is a zero-mean Gaussian noise added to the true function evaluated at some x_i, such that $y(x_i) = f(x_i) + v$, where $v \sim \mathcal{N}(0, \sigma_y)$. Due to the fact that any conditional distribution of a Gaussian is Gaussian itself, we can state the following:

$$p(\mathbf{f} \mid \mathbf{y}) = \mathcal{N}(\mathbf{f}; \mathbf{a} + \mathbf{C}\mathbf{B}^{-1}(\mathbf{y} - \mathbf{b}), \mathbf{A} - \mathbf{C}\mathbf{B}^{-1}\mathbf{C}^{\mathrm{T}}) \tag{8.26}$$

Now consider two values x_i and x_j. If we knew the true function $f(\cdot)$, we could evaluate it at these points, but in the real world the best we can do is take measurements $y(x_i)$, $y(x_j)$ at x_i and x_j respectively. In addition to taking a measurement, we can use x_i and x_j as input to the following covariance, or kernel function:

$$k(x_i, x_j) = \sigma_f^2 e^{\left(-\frac{\left\|x_i - x_j\right\|^2}{2\ell^2}\right)} \tag{8.27}$$

where $2\ell^2$, sometimes denoted by λ is the common length scale of the Gaussian process, and σ_f^2 is the variance of the process. These are often referred to as hyper-parameters, and they are not necessarily known. The purpose of the kernel function is similar to the variogram in kriging, in that it is meant to relate the variance of $f(\cdot)$ to the distance between points. Essentially we have an estimate of variance as a function of distance. It should be noted that there are many other kinds of kernel functions, including periodic, linear, Matérn, and others. It

is also possible to combine certain types of kernel functions by multiplying them together. We can create a matrix that uses the kernel function such that

$$\mathbf{K}(\mathbf{x}, \mathbf{x})_{i,j} = k(\mathbf{x}_i, \mathbf{x}_j) \tag{8.28}$$

where $\mathbf{x}_i$ is the i-th element of the vector $\mathbf{x}$ and $\mathbf{x}_j$ is the j-th element of the vector $\mathbf{x}$.

8.3.2 1-D Example

This sub-section discusses an example of GPR for a 1-dimensional signal. This example is based on [55] and is partly repeated here for completeness. Using the kernel function, Eq. (8.27), we can build the matrices of a joint distribution that we will use to estimate $\mathbf{f}$. First, let $\mathbf{x}^u \in \mathbb{R}^{n\times 1}$ be a vector representing all of the *unobserved* n discrete points comprising a signal and let $\mathbf{x}^o \in \mathbb{R}^{m\times 1}$ be a vector representing all of the *observed* m measured points. We can construct three matrices that describe a new joint distribution based on Eq. (8.28). Let

$$\mathbf{A}(\mathbf{x}^u, \mathbf{x}^u) = \mathbf{K}(\mathbf{x}^u, \mathbf{x}^u), \qquad \mathbf{A} \in \mathbb{R}^{n\times n} \tag{8.29}$$

$$\mathbf{B}(\mathbf{x}^o, \mathbf{x}^o) = \mathbf{K}(\mathbf{x}^o, \mathbf{x}^o) + \sigma_y^2\mathbf{I}, \qquad \mathbf{B} \in \mathbb{R}^{m\times m} \tag{8.30}$$

$$\mathbf{C}(\mathbf{x}^u, \mathbf{x}^o) = \mathbf{K}(\mathbf{x}^u, \mathbf{x}^o), \qquad \mathbf{C} \in \mathbb{R}^{n\times m} \tag{8.31}$$

This yields the following joint distribution:

$$p\left(\begin{bmatrix}\mathbf{f}^u\\ \mathbf{y}\end{bmatrix}\right) = \mathcal{N}\left(\begin{bmatrix}\mathbf{f}^u\\ \mathbf{y}\end{bmatrix};\ \mathbf{0},\begin{bmatrix}\mathbf{A} & \mathbf{C}^{\mathrm{T}}\\ \mathbf{C} & \mathbf{B}\end{bmatrix}\right) \tag{8.32}$$

$$= \mathcal{N}\left(\begin{bmatrix}\mathbf{f}^u\\ \mathbf{y}\end{bmatrix};\ \mathbf{0},\begin{bmatrix}\mathbf{K}(\mathbf{x}^u,\mathbf{x}^u) & \mathbf{K}(\mathbf{x}^u,\mathbf{x}^o)\\ \mathbf{K}(\mathbf{x}^o,\mathbf{x}^u) & \mathbf{K}(\mathbf{x}^o,\mathbf{x}^o)+\sigma_y^2\mathbf{I}\end{bmatrix}\right) \tag{8.33}$$

where $\mathbf{f}^u \in \mathbb{R}^{n\times 1}$ is a vector with n elements representing the true values of the signal evaluated at the n unobserved discrete points in $\mathbf{x}^u$. A prediction using the following two equations describes the posterior over function values $p(\mathbf{f}^u \mid \mathbf{y})$ as

$$\mu_{\mathbf{f}} = \mathbf{C}\mathbf{B}^{-1}\mathbf{y} \tag{8.34}$$

$$\kappa_{\mathbf{f}} = \mathbf{A} - \mathbf{C}\mathbf{B}^{-1}\mathbf{C}^{\mathrm{T}} \tag{8.35}$$

where $\mu_{\mathbf{f}}$ is the mean-predicted function, and $\kappa_{\mathbf{f}}$ is the resulting covariance matrix for the prediction. The variance for each unobserved or observed discrete point corresponds to the diagonal of the covariance matrix. Fig. 8.3 shows an example plot of a true signal, noisy measurements, the predicted mean, and variance associated with the prediction.

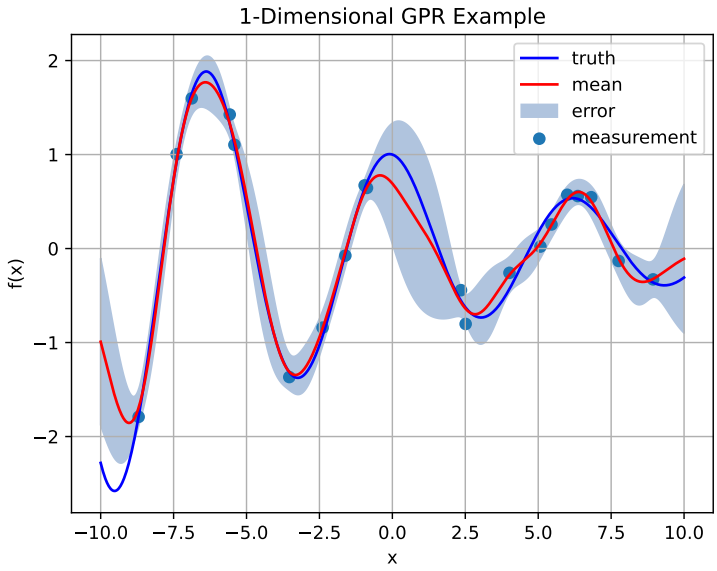


Figure 8.3: A 1-dimension example of GPR. The true signal $f(x)$ is shown in dark blue, the noisy measurements are shown as blue dots, the mean predicted signal is shown in red, and the error bounds (2-standard deviations) are shown in light blue.

The corresponding **A**, **B**, and **C** matrices are shown in Fig. 8.4. Note that the covariance matrix in Fig. 8.4c depicts a "hill" or concentration of higher uncertainty near the middle. This corresponds with the large error bounds in Fig. 8.3 near *its* middle. This makes sense because there is wide space between measurements in that same area and therefore the uncertainty is higher. As a result, the mean-prediction in that area is noticeably less accurate.

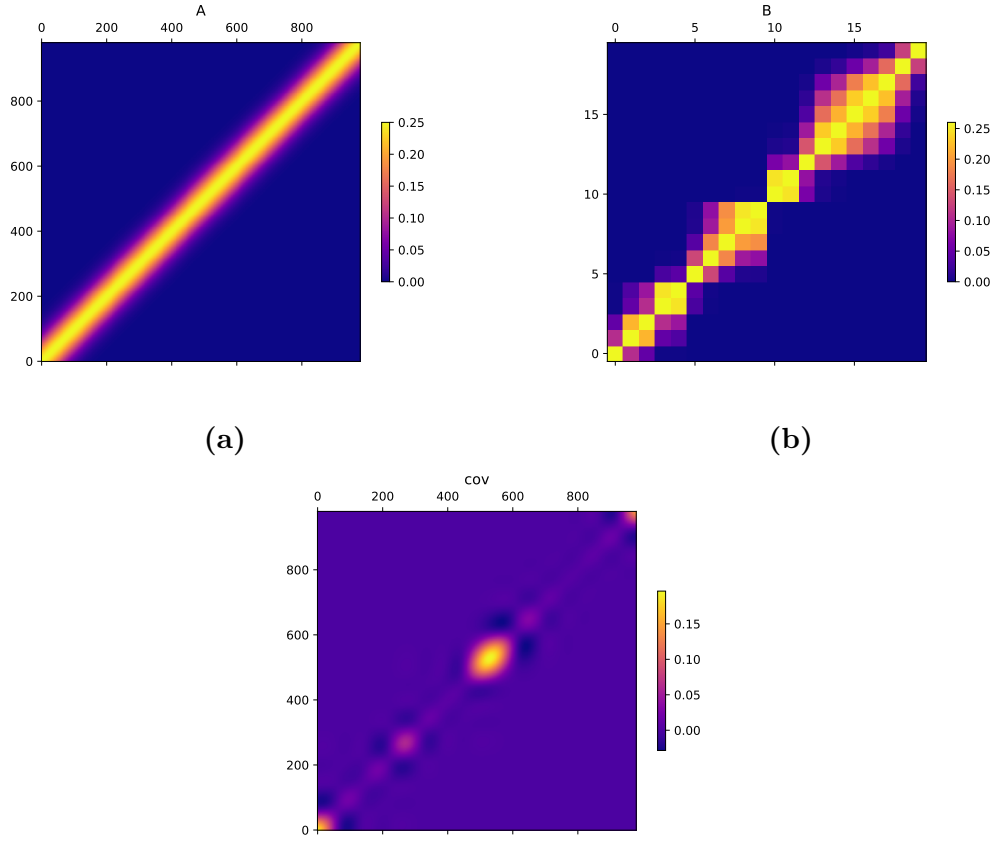


Figure 8.4: The matrices that comprise the joint distribution. The **A** matrix is shown in (a), the **B** matrix is shown in (b), and the covariance matrix $\kappa_{\mathbf{f}}$ is shown in (c).

8.3.3 2-D Example

We extend the same principles to a 2-dimensional case for spatial estimation. Assume that we have m measurements of a field (as shown in Fig. 8.5).

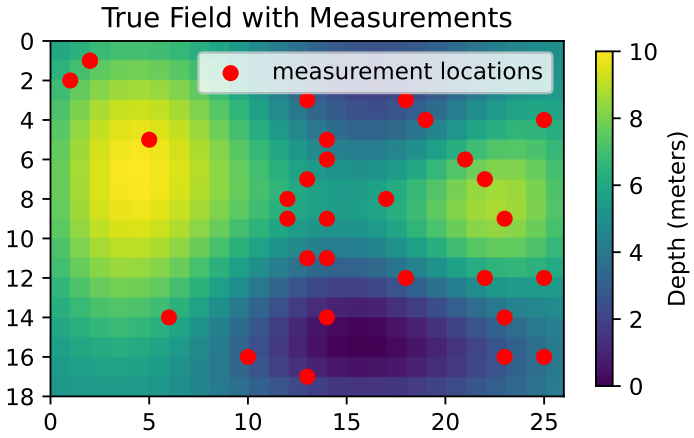


Figure 8.5: A 2-dimension signal $f(x)$ or field is shown as the gradient of dark-to-light color. The noisy measurements are shown as red dots

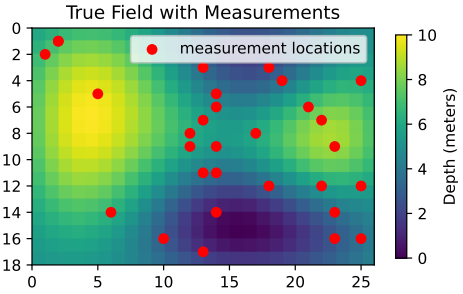
Recall that in the 1-dimensional example, $\mathbf{x}^u \in \mathbb{R}^{n\times 1}$ was a vector that represented the unobserved n discrete points and that $\mathbf{x}^o \in \mathbb{R}^{m\times 1}$ was a vector representing the observed m measured points. In the 2-dimensional case $\mathbf{x}^u$ and $\mathbf{x}^o$ are now matrices such that $\mathbf{x}^u \in \mathbb{R}^{n\times 2}$ and $\mathbf{x}^o \in \mathbb{R}^{m\times 2}$. Each row in these matrices represents a location on the field such that $\mathbf{x}_i^o = \begin{bmatrix} x_i & y_i \end{bmatrix}$ is the i-th row of $\mathbf{x}^o$ where x_i is the x-coordinate on the field and y_i is the y-coordinate on the field. Eqs. (8.29)-(8.31) can still be applied in the same manner as the 1-dimensional case, but the kernel function, Eq. (8.27), is changed to

$$k_{\text{2D}}(\mathbf{x}_i, \mathbf{x}_j) = \sigma_f^2 e^{\left(-\frac{\left\|x_i - x_j\right\|^2}{2\ell^2} - \frac{\left\|y_i - y_j\right\|^2}{2\ell^2}\right)} \tag{8.36}$$

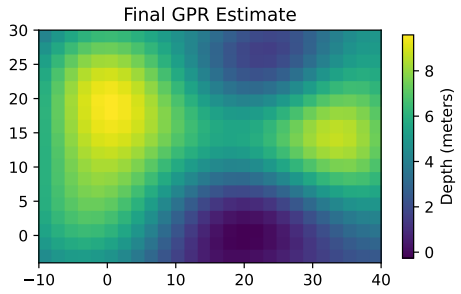
where the two arguments of the function are points on the field (rather than a point on a line, as in the 1-dimensional case). Note that a kernel matrix can be formed similar to the 1-dimensional case, but Eq. (8.28) is rewritten so that it uses Eq. (8.36) as follows:

$$\mathbf{K}(\mathbf{x}, \mathbf{x})_{i,j} = k_{\text{2D}}(\mathbf{x}_i, \mathbf{x}_j) \tag{8.37}$$

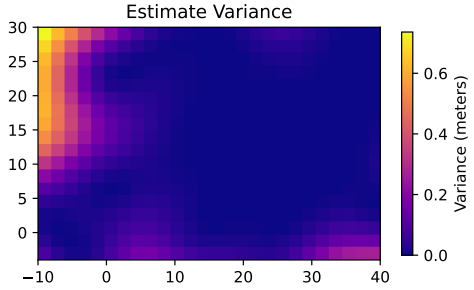
Based on the measurements $\mathbf{x}^o$ of a 2-dimensional field $\mathbf{Z}$, we can estimate all the points in the field that have not been measured. These points are represented by $\mathbf{x}^u$. This is shown in Fig. 8.6. It is clear that GPR works well for spatial estimation in this context; Fig. 8.6b looks nearly identically to Fig. 8.5. However, it is important to recall that this predictive ability of GPR is only possible if the signal or field has some degree of spatial auto-correlation.



(a) True field with measurement locations



(b) Estimated field using GPR



(c) Field estimate variance matrix

Figure 8.6: An example comparison of the estimated field using GPR in 2-dimensions based on the measurements from Fig 8.6a. Note that the locations with a higher density of measurements have a lower variance in the corresponding estimate variance matrix.

A quantitative method to evaluate the degree of error between an estimated signal and the true signal is desirable. In nature we don't necessarily have access to the actual or true field (or true signal), but in simulation we can find the discrete differences at every point between a true field and the estimated field. Later on in this chapter we will identify methods for comparing spatial estimations with the ground truth (in simulation).

8.3.4 Hyper-parameters

The kernel functions Eq. (8.27) and Eq. (8.36) assume the use of hyper-parameters, σ_f^2 and ℓ. There may be additional hyper-parameters if we assume a mean offset or if we employ a different kernel function (see [74]). However, determining the hyper-parameters can be difficult, often times requiring the use of numerical methods such as gradient descent. Another option is to choose the variance scale factor σ_f^2 and common length scale factor ℓ based on the variogram. This is an approximate method. In this thesis two scale factors are chosen, such that k_s corresponds with the sill s of the variogram and k_r corresponds to the range r_0 of the variogram. Both k_s and k_r essentially gains that can be tuned. This leads to the following two equations:

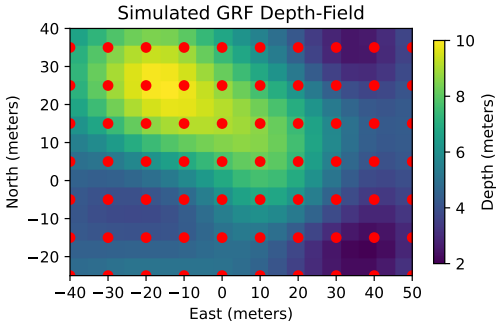
$$\hat{\sigma}_f^2 = k_s s \tag{8.38}$$

$$\hat{\ell} = k_r r \tag{8.39}$$

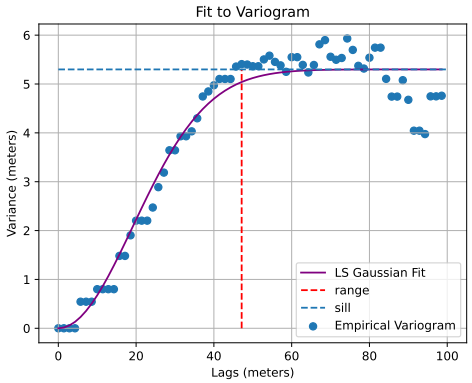
where $\hat{\ell}$ and $\hat{\sigma}_f^2$ represent the approximation for the common-length scale factor variance parameter respectively from Eq. (8.27) and Eq. (8.36).

As an example to demonstrate the usefulness of this hyper-parameter approx-

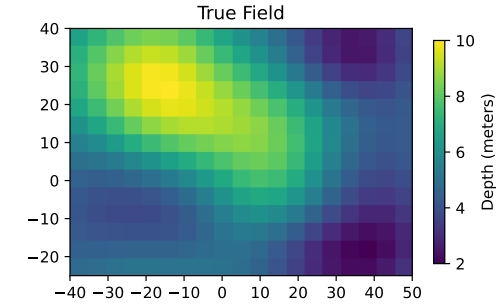
imation technique, consider Fig. 8.7. It shows a field with uniform measurements in a grid pattern (Fig. 8.7a), along with the variogram (Fig. 8.7b), and the GPR-based estimate of the field (Fig. 8.7d).



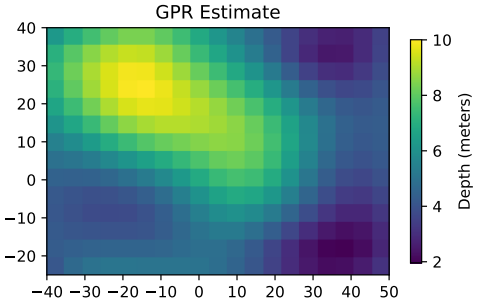
(a) A simulated, "true" field with measurement locations marked by red dots.



(b) The resulting variogram with a sill $s = 5.30$ and range $r_0 = 47.14$ meters



(c) The simulated, "true" field shown without measurements.



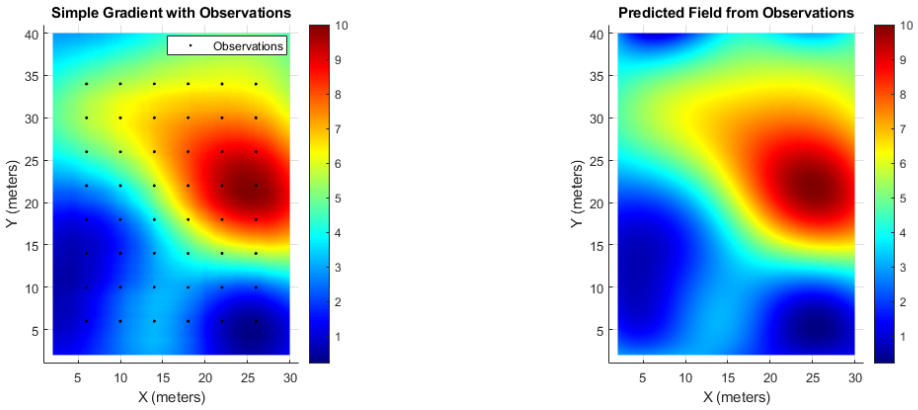
(d) GPR estimated field with $k_r = 0.1$ and $\ell = k_r r_0 = 4.714$, $k_s = 0.1$, and $\sigma_f^2 = 0.53$

Figure 8.7: An example of using GPR to estimate a field with hyper-parameters based on the variogram.

A vehicle may use GPR to estimate the field given a number of measurements. The hyper-parameters can be approximated using Eq. (8.38) and Eq. (8.39), as shown in Fig. 8.7. This means that for each new measurement, or for every nth measurement, the variogram *may* be recalculated, and a new set of approximate hyper-parameters can be calculated.

8.4 Results of Ordinary Kriging

Results of some example simulations of Ordinary Kriging are shown in Figs. 8.8, 8.9, and 8.10. This includes equally spaced observations and resulting predictions or estimates of the field. Two main cases are shown below: 1) a simple field with a single peak and evenly spaced observations, and 2) a more complicated field that is that is less auto-correlated.



(a) A simple, smooth field with observations.

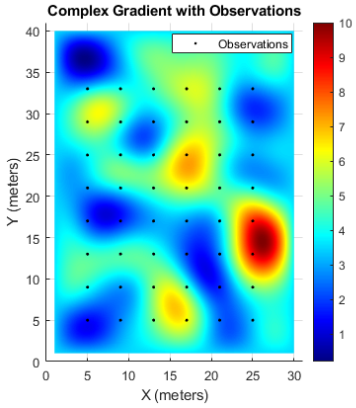
(b) The predicted or estimated field

Figure 8.8: A visual comparison between the true field with observations indicated by black dots (a), and the predicted predicted field based on the observations (b).

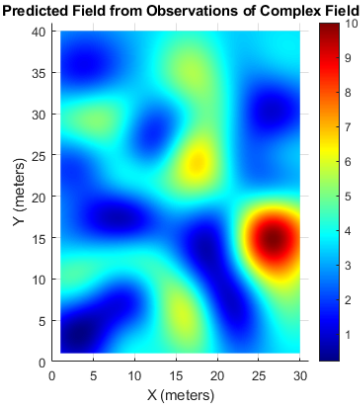
Figs. 8.10a and 8.10b show what happen if random observation locations are chosen as input for the prediction. The uniformity of the observations appears to have an effect on the prediction accuracy. These are provided as an example to highlight this effect, but more quantitative analysis is provided subsequently.

8.5 Partitioned Ordinary Kriging

Instead of pursuing other matrix inversion methods, the cost of matrix inversion is addressed in Partitioned Ordinary Kriging (POK), via recursively dividing

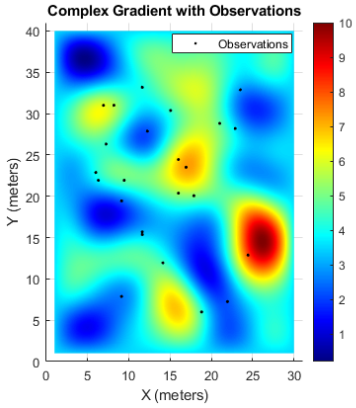


(a) A complex field, having many extrema with uniformly-spaced observations.

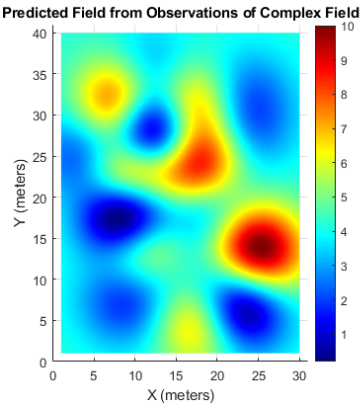


(b) The predicted or estimated field shown in figure 8.9a.

Figure 8.9: A visual comparison between the true field with observations indicated by black dots (a), and the predicted predicted field based on the observations (b).



(a) A complex field with randomly spaced observations.



(b) The predicted or estimated field shown in figure 8.10a.

Figure 8.10: A visual comparison between the true field with observations indicated by black dots (a), and the predicted predicted field based on the observations (b).

the initial field into quadrants, or sub-fields, depending on the ASV location and variogram range. The quadrant that the vehicle currently resides in is the only one for which the Ordinary Kriging is conducted. POK, therefore, is essentially Ordinary Kriging applied to a sub-field whose minimal size allows for reduced

computational complexity

Note that the field should be partitioned based on the range of the variogram, at which point the spatial auto-correlation by definition ceases. Put another way, it is not useful to predict points in the field that exist beyond the range of the variogram. This is because the variance at those points is constant, and therefore the field prediction at those points will also be constant. Using those points as part of a covariance matrix (which must be inverted) is computationally wasteful, in terms of predicting *correlated* parts of the field. We state a condition for subdivision for the maximum level of recursion as,

$$l_{\max} = \left\lfloor \frac{\sqrt{({q_x}^2 + {q_y}^2)}}{r_0} \right\rfloor - 1 \tag{8.40}$$

where q_x and q_y indicate the length and width of a sub-field. If the range $r_0 = 0$, this equation cannot be used because the very idea of using a variogram no longer applies. That is, if $r_0 = 0$ then the field has no auto-correlation for the given field discretization.

Dividing a continuous space into discrete sub-spaces is often referred to as meshing. Extensive literature exists concerning meshing methods in fields such as in computer graphics, finite element analysis, and computational fluid dynamics. Here we present a simple field partitioning algorithm that divides a field, and potential sub-fields into quadrants.

8.5.1 POK Procedure

Assume that a field is first discretized into an $m \times n$ matrix, $\mathbf{Z}$, with discrete points separated by δ_{xy}. Let, $\mathbf{x}$ and $\mathbf{y}$ be $n \times 1$ and $m \times 1$ vectors respectively, with each element representing the $x-$ and y-coordinates of corresponding points in $\mathbf{Z}$. Assume that we will not measure any point in the field more than once.

Using a binary search in both $\mathbf{x}$ and $\mathbf{y}$, we seek to subdivide the field in relation to the ASV's location in the global field. The binary search for determining the x and y bounds, v_0, and v_f of a single axis of a quadrant is defined in Alg. 2, where v is a scalar coordinate value in the x or y-axis, i_l and i_r represent the left and right indices of the vector, $\mathbf{v}$, l is the level of recursion, and $l_{\max}$ is the maximum recursion depth. Note that boundary conditions are not dealt with and we are only concerned with simple partitioning. Alg. 2 uses $l_{\max}$ which is determined from the variogram. We build upon this idea in Alg. 3, where $\begin{bmatrix} x_0 & x_f & y_0 & y_f \end{bmatrix}$ represents the boundary coordinates of a sub-field based on the binary search of the x and y coordinates of the point, $\mathbf{s}_i$. Sub-field boundary coordinates can be generated by using Alg. 3 with a point or a vector of points, and a maximum level of recursion as input. Fig. 8.11 shows an example of applying Alg. 2 and Eq. (8.40).

Algorithm 2 bnry_srch$(v, i_l, i_r, \mathbf{v}, l_{\max}, l_{\text{lvl}})$

If $l_{\text{lvl}} \geq l_{\max}$
    return $\begin{bmatrix} \mathbf{v}[i_l] & \mathbf{v}[i_r] \end{bmatrix}$
$i = \text{round}(i_l + i_r)/2$
if $v \leq \mathbf{v}[i]$
    $i_r = i$
else
    $i_l = i$
return **bnry_srch**$(v, i_l, i_r, \mathbf{v}, l_{\text{lvl}} + 1, l_{\max})$

An explanation for the POK method used with a highest-variance path-planner is provided in [80]. A brief explanation is repeated here for completeness with modifications made for tests outlined in Section 9.1.2.

POK uses a binary search (Alg. 2), in the indices of the x and y coordinate vectors to partition a 2-dimensional plane into quadrants based on a maximum level of recursion, $l_{\max}$. Let v be a value-element in the coordinate vector $\underline{\text{v}}$. The left and right indices are i_l and i_r. The current level of recursion depth is l_{lvl} and

the maximum level is $l_{\max}$.

Eq. (8.40) offers a way to find the maximum level of recursion, where q_x and q_y are the lengths in the x and y dimensions respectively, and $r_0 = r^2 a$ from Eq. (8.4). Note that if the desired smallest distance between field measurements is known, then $l_{\max}$ may be set to that.

Algorithm 3 calculates a quadrant based on a point in the local tangent plane (LTP), $\underline{s}_i$ and the maximum level of recursion. This algorithm effectively provides a quick substitute to other meshing algorithms (e.g.: from computational fluid dynamics or finite element analysis). It need not be called strictly for every measurement point, by instead checking if a new point falls within the bounds of already existing quadrants.

Algorithm 3 calculate_quadrant$(\mathbf{s}_i, l_{\max})$

1: $\begin{bmatrix} x_0 & x_f \end{bmatrix} = \textbf{bnry\_srch}(\mathbf{s}_i[0], 0, \text{len}(\mathbf{x}) - 1, \mathbf{x}, 0, l_{\max})$
2: $\begin{bmatrix} y_0 & y_f \end{bmatrix} = \textbf{bnry\_srch}(\mathbf{s}_i[1], 0, \text{len}(\mathbf{y}) - 1, \mathbf{y}, 0, l_{\max})$
3: **return** $\begin{bmatrix} x_0 & x_f & y_0 & y_f \end{bmatrix}$

A sub-field may be estimated using ordinary kriging, IIOK, or POK. Fig. 8.12 demonstrates the results of a full field estimated with ordinary kriging, IIOK, and POK.

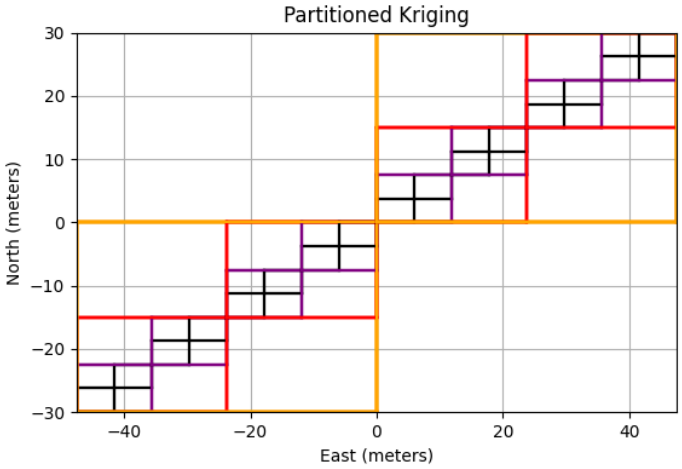


Figure 8.11: An example of generating sub-fields based on measurement locations along the diagonal, using Alg. 3 with $l_{\max} = 4$. Orange boundaries represent the first level of recursive partitioning, red represents the second level, purple is the third, and black is the fourth.

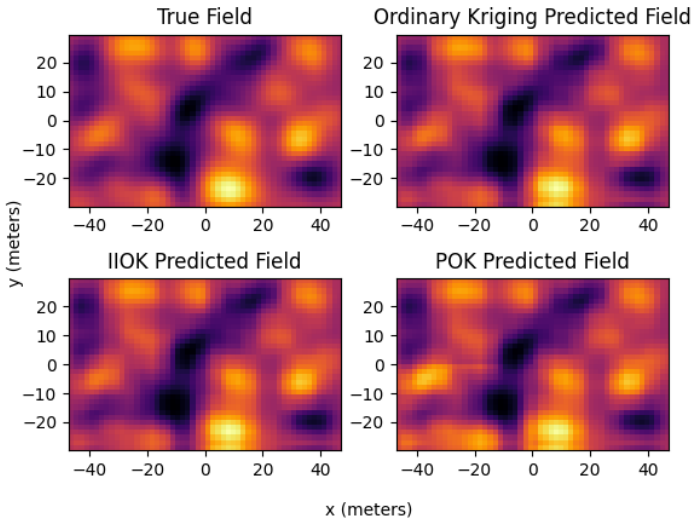


Figure 8.12: Estimated fields using ordinary kriging, IIOK, and POK. Specifically $l_{\max}$ in Alg. 2 was set using Eq. (8.40).

As stated earlier, if a field is partitioned into sub-fields and estimated independently from each other, the boundaries of adjacent sub-fields are not guaranteed to be continuous. A simple solution to this problem, and the one we employ, is to use measurements along the boundary of adjacent sub-fields in the estimate of the target sub-field, thereby aiding in boundary continuity.

8.5.2 Creating a Simulated Field

The simulated field is used in part to help simulate the various algorithms and compare the accuracy and computation times in a consistent fashion. We create a "true field" so that we can compare the error between the true field and the estimated field at every corresponding discrete point. Specifically we create a Gaussian Random Field (GRF). Let $\mathbf{x} = \begin{bmatrix} x_0 & x_1 & \dots & x_{n-1} \end{bmatrix}$, where $x_i = j\mathrm{d}s, j \in [0, 1, 2, \cdots, n-1]$, and where $\mathrm{d}s$ represents the smallest spatial separation between points in a grid in meters. Similarly, $\mathbf{y} = \begin{bmatrix} y_0 & y_1 & \dots & y_{m-1} \end{bmatrix}$, where $y_i = j\mathrm{d}s, j \in [0, 1, 2, \cdots, m-1]$. Again, $\mathbf{x}$ and $\mathbf{y}$ are coordinate vectors corresponding to points in the field $\mathbf{Z}$, which is an $m \times n$ matrix with initially random values for each element pulled from a normal distribution. The range of the distribution represents a change in elevation (deviation in $3-$axis) in this work, but may be chosen to represent a range of any similarly measurable field attribute that possesses auto-correlation.

The next step in creating the true field is to convolve a Gaussian kernel, $\mathbf{K}$ with $\mathbf{Z}$ to enforce the auto-correlation of the field. Let, $g(x, y) = \frac{1}{2\pi\sigma^2} e^{-\left(\frac{(x-\mu_x)^2+(y-\mu_y)^2}{2\sigma^2}\right)}$ be a 2D Gaussian function, where μ_x and μ_y are the mean in x and y respectively, and σ is the standard deviation. The corresponding Gaussian kernel is, $\mathbf{K} = \frac{1}{\sum_{i=0}^{b-1}\sum_{j=0}^{b-1}\mathrm{K}_{i,j}}\left(\sum_{i=0}^{b-1}\sum_{j=0}^{b-1}\mathrm{K}_{i,j} = g(j,i)\right)$ such that $\mathbf{K}$ is a $b \times b$ square matrix. The resulting convolution is the true field $\mathbf{Z}$. An example true field is

shown in Fig. 8.12, panel (a).

Other Methods for Creating Fields

It should be noted that the above method for creating field is only one of many. There are, in fact, a vast number of methods to create fields to describe everything from the theoretical distribution of cosmic dark matter, to the spatial correlation on the hydro-mechanical behavior of large open pit problems, and others. A review of recent advancements and applications of GRFs specifically is given in [48]. Additionally, [48] discusses various other mathematical approaches to generating GRFs, including the use of fast Fourier transforms, and power spectrum matrices (which can significantly decrease the field generation computation time).

8.5.3 Path Planning

Path planning in this work is defined in two parts: waypoints and trajectories. Waypoints are defined as the locations within the field where the vehicle will pass through (or near) and take measurements. Trajectories are defined as the interpolation of the waypoints subject to the constraint of a realizable path for the vehicle. This may be linear, spline-based, hybrid, or any other similar function. Although it is shown in [83] that path planners such as the Monte Carlo path planner perform well in simulation, these sorts of paths are physically unrealizable by conventional vehicles. Most vehicles have smooth motion by nature (defined as continuity limits on the path and its derivatives). This is generally the case for fixed-speed marine surface vehicles conducting field exploration.

Highest Variance Path Planner

A variety of path planners are discussed and tested in [83] using ordinary kriging. We choose the highest variance path planner (HV) to show the computation time improvement between Ordinary Kriging, IIOK, and POK.

The HV path planner sets the next waypoint for the ASV to travel to as,

$$\mathbf{s}_{i+1} = \max(\text{var}(\hat{\mathbf{Z}})) \tag{8.41}$$

In other words, the next waypoint is chosen to be the location in the field one where the variance of the estimate - and therefore the uncertainty in the value of field - is highest.

Trajectory Generation

We employ a linear trajectory (e.g.: straght lignes) between waypoints selected by the path planner. The ASV follows the trajectories from one highest variance waypoint to the next, collecting field measurements along the way. Every time the ASV arrives at a waypoint, the empirical variogram is calculated using Eq. (8.2), the empirical variogram is fit with a Gaussian model, and a new local field estimate is made using Eqs. (8.15-8.24).

The steps of the POK procedure are shown in Alg. 4. In Alg 4, the ASV position is denoted by $\mathbf{s}_{\text{ASV}}$. The next waypoint with the highest variance is $\mathbf{s}_{i+1}$ The maximum number of measurements, $n_{\max}$ per field can vary depending on the desired spatial resolution, field size, vehicle, and sensor(s).

Algorithm 4 POK Procedure

Travel to the center of the field, taking measurements
If Arrived at center
    Generate the empirical variogram, $\hat{\gamma}(h)$ using Eq. (8.2)
    Fit $\gamma(h)$ to $\hat{\gamma(h)}$ using least-squares
while $n \leq n_{\max}$
    Generate the global empirical variogram, $\hat{\gamma}_g(h)$ using Eq. (8.2)
    Fit $\gamma_g(h)$ to $\hat{\gamma}_g(h)$
    Calculate $l_{\max}$ using Eq. (8.40)
    $q = \textbf{calculate\_quadrant}(\mathbf{s}_{\text{ASV}}, l_{\max})$
    Generate the local quadrant empirical variogram, $\hat{\gamma}_q(h)$ using Eq. (8.2)
    Fit $\gamma_q(h)$ to $\hat{\gamma}_q(h)$
    Generate local covariance matrix, $\mathbf{C}_q$ using $\gamma(h)$ and Eq. (8.22)
    Invert $\mathbf{C}_q$ using Eq. 8.23
    $\forall p_i \in \mathbf{Z}_{\text{quad}}$
        $\mathbf{d}_i = \begin{bmatrix} \gamma_q(||\mathbf{s}_0 - \mathbf{p}_i||) & \dots & \gamma_q(||\mathbf{s}_{m-1} - \mathbf{p}_i||) \end{bmatrix}^{\text{T}}$
        $\begin{bmatrix} \lambda_{\mathbf{p}_i} \\ \eta_{\mathbf{p}_i} \end{bmatrix} = \begin{bmatrix} \mathbf{C}_q^{-1} & 1 \\ 1^{\text{T}} & 0 \end{bmatrix} \begin{bmatrix} \mathbf{d}_i \\ 1 \end{bmatrix}$
        $\hat{\mathbf{Z}}_q(\mathbf{p}_i) = \begin{bmatrix} \mathbf{Z}_q(\mathbf{s}_0) & \dots & \mathbf{Z}_q(\mathbf{s}_{m-1}) \end{bmatrix}^{\text{T}} \lambda_{\mathbf{p}_i}$
        $\text{var}(\hat{\mathbf{Z}}_q(\mathbf{p}_i)) = \begin{bmatrix} \mathbf{d}_{\mathbf{p}_i} & 1 \end{bmatrix} \begin{bmatrix} \lambda_{\mathbf{p}_i} \\ \eta_{\mathbf{p}_i} \end{bmatrix}$
        $\hat{\mathbf{Z}}(\mathbf{p}_i) = \hat{\mathbf{Z}}_q(\mathbf{p}_i)$ // update global field estimate at point $\mathbf{p}_i$
    $\mathbf{s}_{i+1} = \max(\text{var}(\hat{\mathbf{Z}}(\mathbf{p}_{0:i})))$
    Travel to the next point of highest variance, taking field-measurements along the way
return $\hat{\mathbf{Z}}, \text{var}(\hat{\mathbf{Z}})$

8.5.4 Simulation Results and Comparisons

To evaluate the three methods, the z-component of a synthetic field was chosen as a generic field attribute to estimate. The field had a spatial resolution $\mathrm{d}s = 0.5$ meters and had a total of $4,670$ discrete points to potentially measure. The 2D Gaussian kernel used for convolution of the initial random field was a 5×5 matrix. The convolution was applied 7 times, to achieve a field with a high level of auto-correlation. The simulations were performed using two different path planners. The first used randomized waypoints with measurements taken along the way, but with no explicit straight-line segment planning. The second used the HV path planner with straight-line segments between waypoints. In both, the full vehicle dynamics were used (see Eq. (6.10)). Both simulations compare Ordinary Kriging, IIOK, and POK respectively for computation time and spatial estimation accuracy. Accuracy is defined here as the mean squared error (MSE), e_k of all discrete points between the current estimated field and the true field at a given number of k points scanned.

$$e_k = \frac{1}{mn} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} (\hat{\mathbf{Z}}_k(i,j) - \mathbf{Z}(i,j))^2 \tag{8.42}$$

Randomized Waypoint Strategy

The set of random field measurement locations were used as input for Ordinary Kriging, IIOK, POK methods (Alg. 4), GPR, and Partitioned Gaussian Progress Regression (PGPR). For each new field measurement, the computation time of each algorithm was recorded, along with the MSE of the field estimate using Eq. 8.42. The results are shown in Fig. 8.13.

The computation time for generating the field estimate is compared and shown in Fig. 8.14. The POK and PGPR methods show a logarithmic plot of the com-

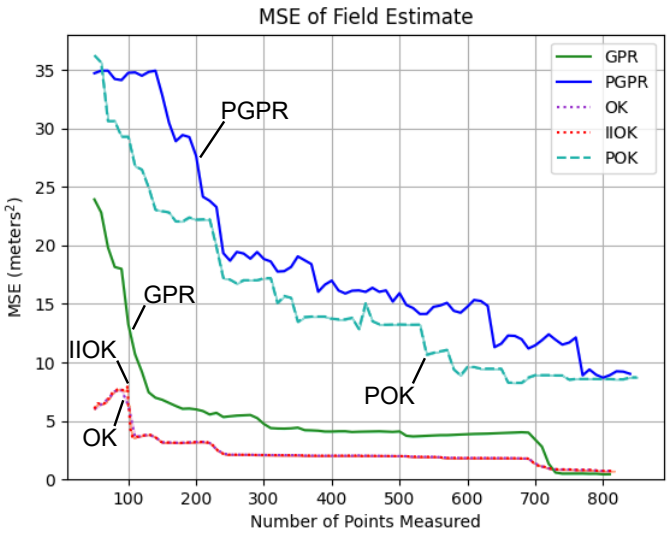


Figure 8.13: Comparison of mean squared error for field estimates versus the number of points scanned. The waypoints for measurements were generated using a random waypoint path planner.

putation time trend that is significantly less than the other methods.

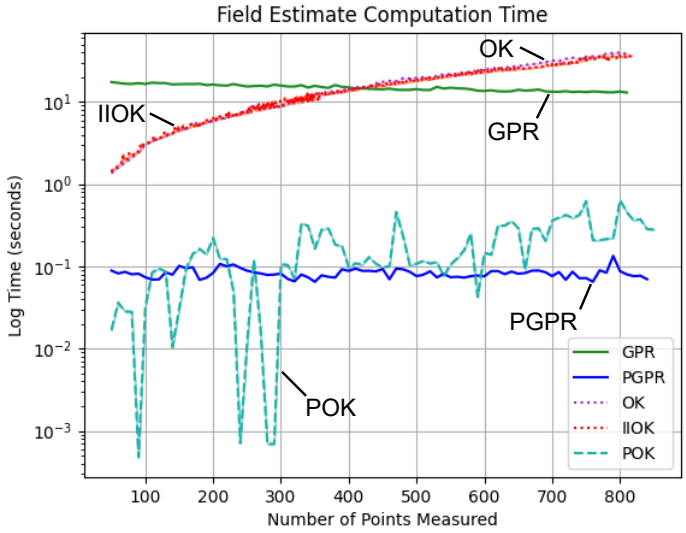


Figure 8.14: Comparing computation time (on a logarithmic scale) for estimating the field using Ordinary Kriging, IIOK, POK, GPR, and PGPR with respect to the number of points scanned based on a random waypoint path planner.

Remarks on GPR Computation Time

Interestingly, GPR shows an initially higher computation time compared to OK and IIOK, followed by an apparent decrease in computation time as the number of measurements grow. The initially high computation time is due to the fact that GPR calculates a joint distribution for all measured *and* unmeasured points. Therefore it should be expected to have a higher computation time at the start of collecting measurements. The apparent decrease is due to some underlying optimizations in the python modules that were used to implement GPR. At low numbers of measurements, the majority of the joint distribution maintains a higher ratio of unmeasured to measured points. The underlying GPR code takes this into account, briefly decreasing the computation time. In Fig. 8.15 we see that GPR *does* increase in computation time as a function of the numbers of measurements, as would be expected. Furthermore, there is a key difference between

GPR and Ordinary Kriging in terms of computation time. Ordinary Kriging must iterate through the points in the field for every measurement, which coincides with Eqs. (8.17-8.21), where as GPR updates all points through Eq. (8.34), which eliminates an extra looping step making it more computationally efficient. Note that the point at which the computation time begins to observably increase, corresponds with $\sim \%50$ of the total number of possible measurable points in the field (the field is discretized with a resolution of ds meters between each adjacent point).

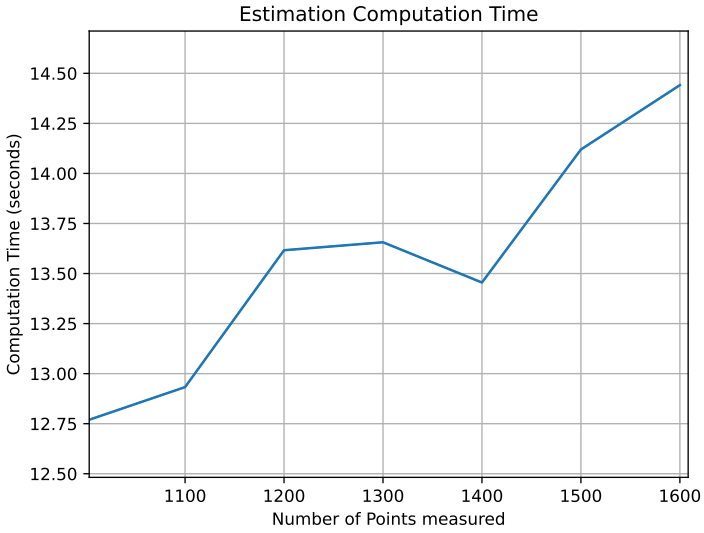


Figure 8.15: The computation time for GPR compared to the number of points.

8.5.5 Path-Planning Simulation Results

The HV path planner was used for waypoint generation and trajectory development. Ordinary kriging, IIOk, POK, GPR, and PGPR were compared in terms of MSE. Fig. 8.16 shows MSE versus the number of points measured and Fig. 8.17 shows the computation time on a logarithmic scale for each spatial estimation

algorithm versus the number of points measured.

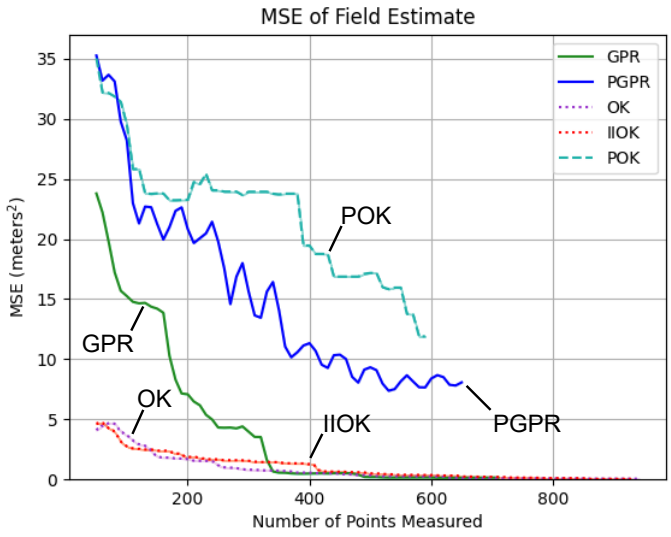


Figure 8.16: MSE of the different kriging methods and GPR methods versus the number of points measured while using the HV path planner

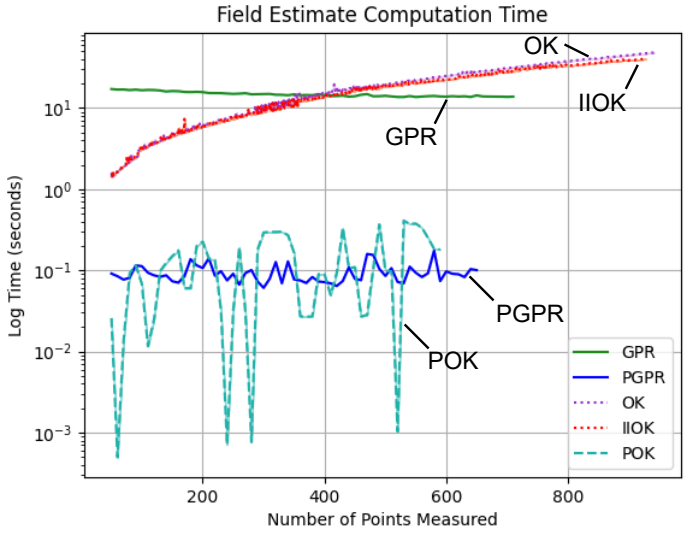


Figure 8.17: Computation time on a logarithmic scale of the different kriging methods and GPR methods versus the number of points measured while using the HV path planner.

8.5.6 Remarks on MSE and Computation Time Trade-offs

This portion of the thesis introduced a unique partition method for spatial estimation that will allow online trajectory planning due to computation time reduction. Ordinary Kriging, IIOK, POK, GPR, and PGPR are implemented with an HV path planner in simulation. A superior computation time was demonstrated as compared to Ordinary Kriging and IIOK. An apparent trade-off is that the MSE converges mores slowly, though the ability to provide more frequent field estimates may still be considered advantageous. POK and PGPR exhibit computation times and accuracy that are favorable for experimental online trajectory planning and field exploration with an ASV.

Comparing the MSE signals associated with using random waypoints versus the HV path planner (see Fig. 8.13 and Fig. 8.16) there is a notable difference; the random waypoints results in MSE signals that decrease more gradually than compared to the MSE signals for the HV path planner. This seems to be the case regardless of the spatial estimator. The reason for this faster decrease in MSE is due to the fact that the HV path planner will tend to leave regions of the field that are explored because - by definition - explored regions have less variance than unexplored regions. In Fig. 8.13 there is a noticeable plateauing effect in the MSE signals. This is because the random waypoints path planner does *not* consider the variance associated with the estimate(s) of the field. Using random waypoints means that the ASV may visit a new point in a region of the field near many previous measurements. This is not very "useful" and will not impact the accuracy of the next field estimate compared to the HV path planner which will always identify the points of highest variance to explore next. From these simulations it is clear that the choice of path planner matters and that considering the variance associated with a field estimate can be useful for

exploring.

8.6 Optimal Exploration

For an ASV to explore a given area, a question arises; what is the best way to estimate the field? The answer to that question depends on the what we are trying to measure in the field and how it is distributed. As mentioned earlier and throughout this thesis, if a field has no auto-correlation associated with the field attribute, then the best method of exploring is to simply conduct a zig-zag search pattern. However, nature is full of auto-correlated features, and many if not most natural geological features possess a Gaussian distribution (or at the very least can be assumed to possess such a distribution). Under the assumption that the field is auto-correlated, the answer for how to optimally explore a field changes; a zig-zag pattern is not guaranteed to optimally reduce the uncertainty associated with a field estimate. This section investigates how to conduct optimal exploration with respect to maximizing the rate of reducing uncertainty of the field estimate over time.

Given a variance matrix associated with a spatial estimate of a field, we seek to find an optimal path between two points on the field that maximizes the variance along the path. This is essentially the path that would suppress the most uncertainty if followed.

8.6.1 Maximizing Variance Along A Path

The problem of minimizing the estimation error can't be directly solved, because we do not have access to the true field prior to exploration. We restate the problem to one of uncertainty suppression. Specifically, we seek a path that

maximizes the total variance along the path but doesn't exceed the maximum path distance the ASV can travel. We apply Bellman-Ford optimization in a graph-search to achieve this result. The Bellman-Ford graph search guarantees an optimal solution in polynomial time, but terminates in the presence of negative cycles in the graph. Therefore we choose to constrain the formation of the graph to be specifically directed and without any cycles, also referred to as a directed a-cyclic graph (DAG).

Graph Formation Algorithm Description

This section discusses forming a DAG using Algorithm 5. First, a node matrix is formed, such that each node exists on top of an *active* discrete coordinate, or point on the LTP. This means that the graph edges pass over discrete points as well and thus the edge weights are also associated with the variance at points they pass over. Nodes can exist at discrete points, but do not take any value associated with that point, and so that point is said to be *inactive* in relation to that node. The minimum distance between adjacent nodes is denoted by $\mathrm{d}s$. It follows that diagonal distances between nodes is $\sqrt{2}\mathrm{d}s$. If $\mathrm{d}s$ is very small, implying a high-resolution and large variance matrix, then it may be computationally preferable to assign a node to be associated with every n-th discrete element. For sufficiently large matrices, this can allow for a more reasonable computation time, because the Bellman-Ford algorithm scales as $O(V_G E_G)$, where V_G and E_G are the number of vertices and graph edges.

Algorithm 5 Directed a-cyclic graph formation algorithm

1: Form a node matrix between points a and b with each n-th node corresponding to an element in the variance matrix
2: For every node in the node matrix add an edge weight w_i in the horizontal, vertical, and diagonal direction for adjacent
3: nodes that point from point a to point b. In order to maintain the a-cyclic quality of the graph, the following edge conditions must be satisfied:
4: If a horizontal edge is to be added, it must point from point a towards point b
5: If a vertical edge is to be added, it must point from point a towards point b
6: If a diagonal edge is to be added, it must point from point a towards point b
7: **return** G

It should be noted that there are many different ways to form a directed a-cyclic graph with varying degrees of complexity, but this method is chosen because it is straight forward to implement in code. Another example of a graph for a similar purpose is a hexagonal graph outlined in [32]. It is important to mention that the DAG is restricted in Alg. 5 such that the edges never point away from point b; the edges may point in an orthogonal direction, but never backwards toward point a. This is a restriction that could *potentially* be lifted with a more complicated DAG structure, but it would likely involve more complicated methods of ensuring that no cycles were formed. This idea is left for future work.

Assuming we have a DAG, and a variance matrix from applying kriging or GPR, we may choose to apply the Bellman-Ford graph search algorithm to maximize the variance along a path connecting points a and b. As mentioned earlier, the Bellman-Ford algorithm terminates in the presence of negative cycles, but since the graph has no cycles, and therefore the algorithm will not terminate until an optimal variance path is found. However, consider the case where we attempt to find a path with maximize variance. Because there are no cycles, the path that maximizes the integrated variance along it is also the path that travels through

the most edges in the DAG. This would correspond to the most integrated variance. If the node spacing is larger than the point spacing it is possible to visit all nodes without traversing all edges. For example, in Fig. 8.21 a horizontal "lawn-mower" pattern could be executed, and this would avoid all diagonal edges. Depending on the DAG geometry this path can take many forms, but in the DAG geometry outlined earlier this results in a zig-zag pattern. In reality this is not efficient either in terms of time and energy use for an ASV. The ASV might not have enough energy or fuel to travel to every node in the graph in a given area. The Bellman-Ford algorithm gives the minimum variance to each node. Distance between nodes can be used to control energy use.

The classical A* algorithm would seem to be a good choice to solve this problem, but it is limited to finding the shortest path assuming positive edge weights as a cost. Since we want to *maximize* the variance, A* would treat variance as a cost. Using the negative of the variance also leads to a problem: A* cannot use negative weights. Using the sum of the inverse of the variance $\sum \frac{1}{\text{var}}$ leads to edge-cases when the variance is small (e.g.: var < 1). Fig. 8.18 shows one edge case where the "shortest" inverse variance cost function results in an incorrect (and sub-optimal) solution with a path that does *not* possess maximum variance. Note that stating the problem this way is akin to the finding the "longest" path and is *np* hard.

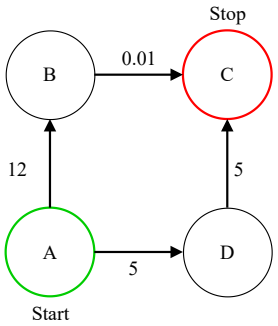


Figure 8.18: An example of a directed a-cyclic graph with weights. This specific graph shows that if the A* search uses the sum of the inverse variance as the cost function then the path of maximum variance is not returned.

The inverse variance cost to traverse path A-B-C in Fig. 8.18 is $\frac{1}{12}+\frac{1}{0.01} = 100.08\bar{3}$, while the sum of the variance is var(A-B-C) = 12.001. The sum of the inverse variance cost for path A-D-C is $\frac{1}{5} + \frac{1}{5} = 0.4$, but the sum of the variance is var(A-D-C) = 10. A* will return path A-D-C because the sum of the inverse variance cost is lower, but var(A-D-C) < var(A-B-C). The path of maximum variance was not returned. This is because A* uses the addition operator in its cost function and so we end up running into the fact that $\frac{1}{a} + \frac{1}{b} \neq \frac{1}{(a+b)}$.

Since A* is cannot be used, the solution is to minimize the negative of the variance using the Bellman-Ford algorithm (which *can* accommodate negative weights). In this way, the "cheapest" path is the shortest path with the most negative integrated total variance. This can be done by assigning the negative of each element of the variance matrix as a weight for each corresponding edge on the DAG. An example of an optimal variance path is shown in Fig. 8.21. Again, if the vehicle had limitless energy and time, the optimal path would be to visit all possible edges in the graph in a zig-zag pattern. This would yield the highest total variance. With energy constraints the problem is to determine to what extent a zig-zag pattern is followed. The width of the pattern and the location of the turns must be calculated to maximize variance. Algorithm 5 takes Eq. (8.43) as input

and produces a graph such as the one shown in Fig. 8.19.

$$\mathbf{N} = \begin{bmatrix} 0 & 1 & \dots & n-1 \\ n & n+1 & \dots & 2n-1 \\ \vdots & \vdots & \dots & \vdots \\ m+n-1 & m+n & \dots & m+2n-1 \end{bmatrix} \tag{8.43}$$

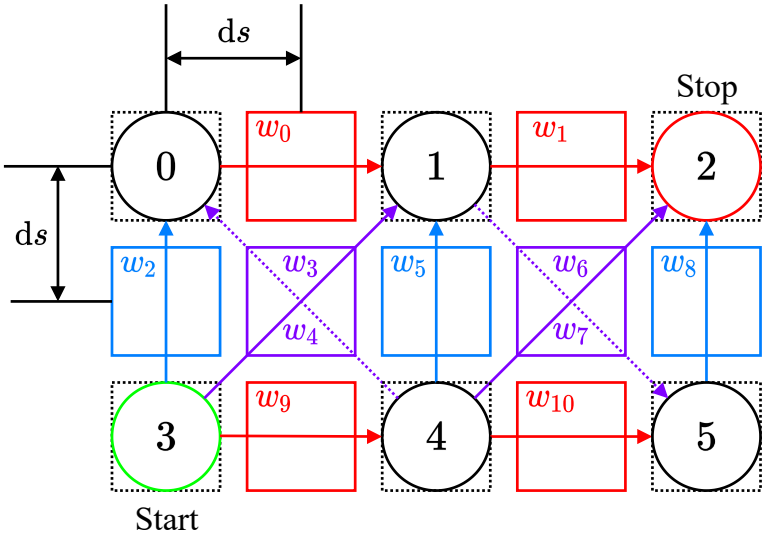


Figure 8.19: Example of a variance directed a-cyclic graph formed by Algorithm 5. Each square represents a discrete point within the LTP. The graph nodes are represented by circles. There are purposefully fewer nodes than discrete points to show that node spacing is adjustable, and can be spaced every $n \times \mathrm{d}s$. This is a feature so that a graph search can take less time, if fewer nodes are desired.

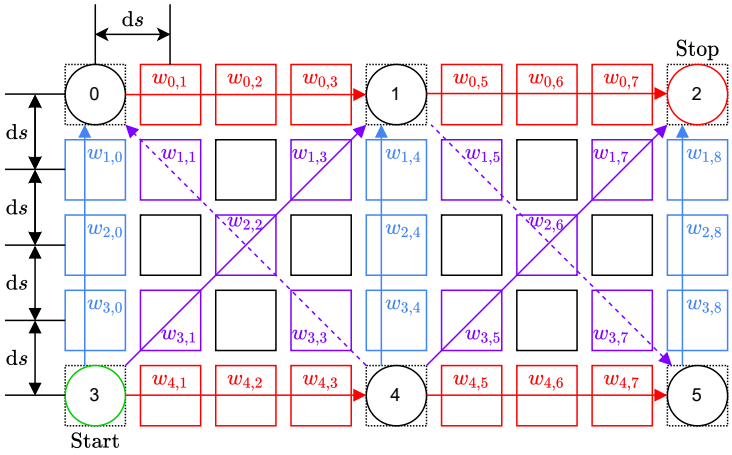


Figure 8.20: An example of a variance directed a-cyclic graph formed by Algorithm 5. Here the DAG is shown with $4 \times \mathrm{d}s$ spacing to further highlight the variability of Algorithm 5. The start node (bottom left) is labeled and marked by a green circle. The end node (top right) is also labeled and marked with a red circle.

The diagonal edge weights in Fig. 8.19 are equal. For instance $w_3 = w_4$ and $w_6 = w_7$. The DAG can be represented in the matrix form:

$$\mathbf{G} = \begin{bmatrix} a_0 & b_0 & c_0 \\ a_1 & b_1 & c_1 \\ \vdots & \vdots & \vdots \\ a_{m-1} & b_{m-1} & c_{m-1} \end{bmatrix} \tag{8.44}$$

where a given row G_i represents a directed graph edge connecting node a_i to node b_i with weight c_i. Note that the node numbers correspond with the elements from Eq. (8.43). With $\mathbf{G}$, consider Alg. 6:

Algorithm 6 Bellman Ford Optimal Graph Search

Require: G

Initialize:
    v, the number of vertices in the DAG
    $\mathbf{d} \in \mathbb{R}^{v \times 1}$, a vector whose elements are all initially ∞
    $\mathbf{p} \in \mathbb{R}^{v \times 1}$, a vector to hold the parent nodes.
For $i = [0, 1, 2, \cdots, v-1]$:
    For each row $(a, b, c) \in \mathbf{G}$:
        If $\mathbf{d}(a) \neq \infty$ and $(\mathbf{d}(a) + c) < \mathbf{d}(b)$:
        $\mathbf{d}(b) = \mathbf{d}(a) + c$
        $\mathbf{p}(b) = a$, set the parent of node b
For each row $(a, b, c) \in \mathbf{G}$:
    If $\mathbf{d}(a)! = \infty$ and $(\mathbf{d}(a) + c) < \mathbf{d}(b)$:
        Terminate because a negative cycle was found.
return $\mathbf{p}$, The optimal path that maximize variance along the path

Recall that the number of nodes that comprise the DAG can be adjusted, and the geometry of the DAG can also be customized to different patterns.

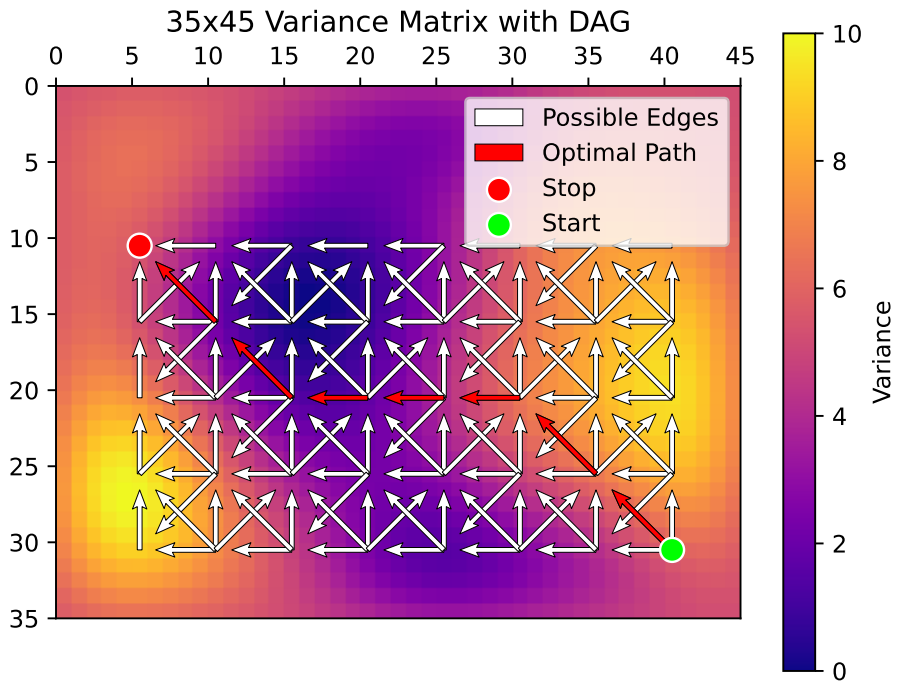


Figure 8.21: Example of a maximum variance, minimum distance path (red arrows) between a start point (green dot) and a stop point (red dot).

Optimal Path Update Algorithm

As the vehicle moves, it takes measurements that are used to recalculate the variogram, and form a new estimate of the field along with a variance matrix of the estimate. This means that the optimal path can be updated up to as frequently as new measurements can be made and if computation time allows. We present Algorithm 7.

Algorithm 7 Bellman Ford Kriging Exploration

1: $\mathbf{r}_0, \mathbf{r}_f, \mathbf{W}, \Delta r$
2: Initialize:
3: $i = 0$
4: $\mathbf{r}_i = \mathbf{r}_0$ // Initialize the vehicle position
5: Initialize the global weight matrix $\mathbf{W}$
6: **While** $\|\mathbf{r}_i - \mathbf{r}_f\| > \Delta r$
7: If vehicle position is at a new discrete point:
8: Take a field attribute measurement
9: If $\text{mod}(i, n) == 0$:
10: Form the empirical variogram
11: Fit the Gaussian model variogram to the empirical variogram
12: Form the covariance matrix $\mathbf{C}$
13: Apply Ordinary Kriging to obtain the variance matrix $\hat{\mathbf{V}}$
14: Form a DAG using Alg. 5 and map the weight matrix $\mathbf{W}$ elements to the graph edges
15: Apply Bellman-Ford graph search algorithm to obtain the optimal path P_{opt} connecting start point p_0 to end point p_1
16: Control thrust mechanism to follow P_{opt}
17: $i++$
18: **return** P_{opt}, The optimal path that maximizes the variance along the path

Note that $\mathbf{r}_i = \begin{bmatrix} x_i & y_i \end{bmatrix}$ is the vehicle position in East and North positions in the local tangent plane. The initial and final vehicle positions are $\mathbf{r}_0$ and $\mathbf{r}_f$ respectively. Δr is the distance

An end point may be defined as the stop node in a DAG (see Fig. 8.20), or it may be set to fulfill another object such (e.g: setting the end point to the next highest variance point). Algorithm 7 may be run after reaching an end point p_1 by specifying a new end point p_2, p_3, and so on. Steps $2-8$ are executed at every $n-$th new measurement, where n may be determined based on the resolution of the field, the speed of the processor, how often a new range may be desired, etc. A new DAG can be formed at every $n-$th new measurement. If endpoints are set to points of highest variance and one happens to falls outside of the current DAG, a new DAG will be able to connect to it. Additionally, the last variance matrix can

be used as input for the algorithm towards the new end point. Some paths could cost more energy than a vehicle could spend. However all path costs are known; all end point path costs are a default output of the Bellman-Ford algorithm, so these nodes are known. Again, we assume that reducing the spatial estimation error is the primary goal for using this algorithm. Note that the end points can be chosen freely, but may be associated with other algorithms or path planners.

8.6.2 Minimizing the Sum of Negative Variance

Bellman-Ford optimization (Alg. 6) is applied to a DAG to minimize the sum of negative variances (SNV) along the edges of the DAG. This yields the path - comprised of DAG edges - that sum to the most negative variance. We consider a cost function to represent how the Bellman-Ford algorithm minimizes the SNV. Recall that the variance discussed here is associated with the spatial estimate of the field and *not* individual measurements. The Bellman-Ford graph search algorithm is capable of using negative weights. This fact is used in relation to a cost function:

$$J = \int_0^{t_f} L(x, v)\mathrm{d}t + \gamma(t) \tag{8.45}$$

where $L(\cdot)$ is the cost (negative variance) v at at each edge x of the DAG, and $\gamma(t)$ is a weighting function that effectively limits the final path length. Note that $\gamma(t)$ may be constant. Fig. 8.22 shows the result of recording the SNV over time for 100 different field exploration simulations. The fields possessed 266 discrete points.

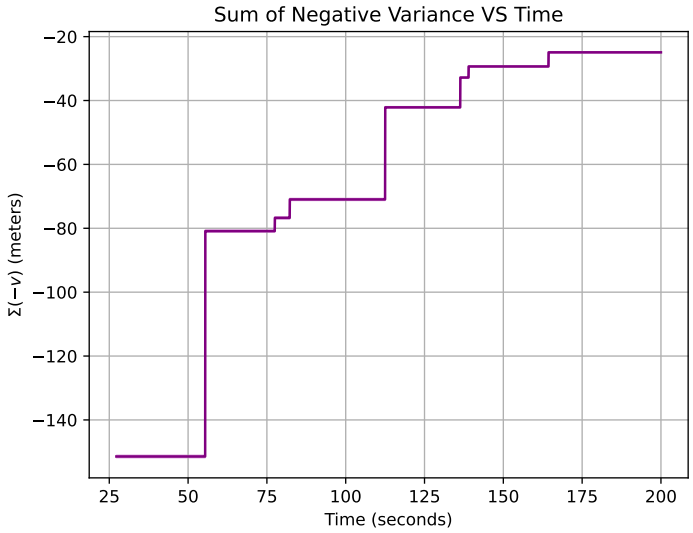


Figure 8.22: The sum of the negative of the variance (SNV) versus time. This is averaged for the exploration of 100 different fields with 266 discrete points per field. The SNV plot is asymptotic as it approaches zero.

Finding the path that the minimizes the SNV is important for exploration in terms of suppressing uncertainty; we seek to explore most uncertain (highest variance) points within the field. We treat variance as a cost by making it negative. Thus, when we apply the Bellman-Ford algorithm Alg. 6, the resulting "shortest" path is the path that minimizes the SNV, or the path with the most variance.

8.6.3 Simulation Results

Here we present simulation results for a Mars Ingenuity-like vehicle approximated with the Newtonian model from Ch. 5. Recall that the Newtonian model is a state space model that accounts for aerodynamic drag, moment of inertia, torque due to thrust, and other disturbances in discrete state space. This level of complexity was desired to show that the algorithm is vehicle agnostic. We apply Alg. 7 between two points in an unexplored field. The model parameters where chosen such that the mass $m = 20.0$kg, the radius $r = 0.045$m, and a solid sphere

was assumed for the moment of inertia $\mathcal{I} = \frac{2}{5}mr^2$.

Setup

The field was artificially created using the same 2D convolution methods shown in [80]. The field resolution was $\mathrm{d}s = 4.0$-meters. The DAG node spacing was $n = 2$, $n \times \mathrm{d}s$. The starting position was $\mathbf{r}_0 = \begin{bmatrix} -52.0 & -4.0 \end{bmatrix}$. The stopping position was $\mathbf{r}_f = \begin{bmatrix} 32.0 & 50.0 \end{bmatrix}$. The distance cost or DAG weight matrix $\mathbf{W}$ for is based on Fig. 8.25.

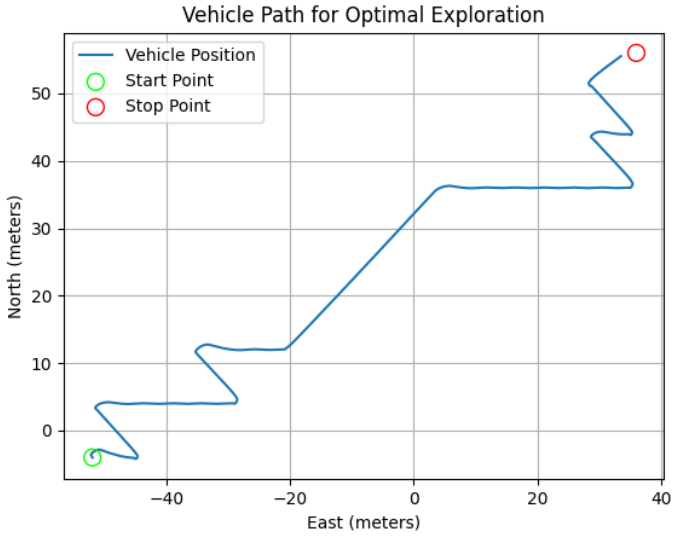


Figure 8.23: Vehicle position from the starting to ending point.

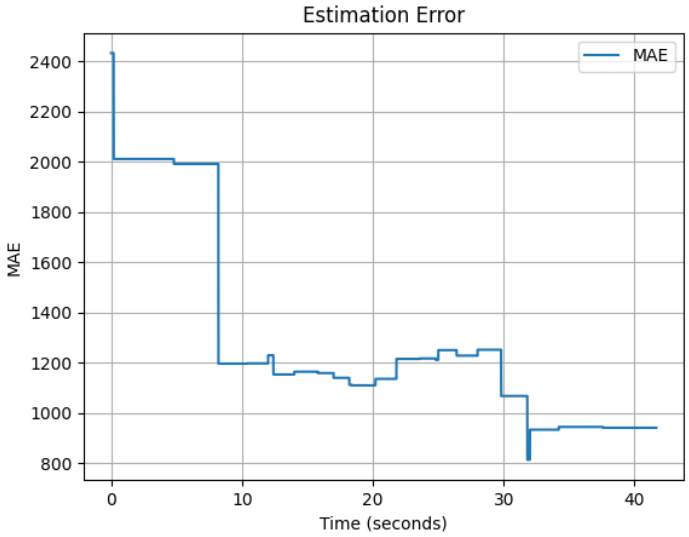


Figure 8.24: Estimation error versus time.

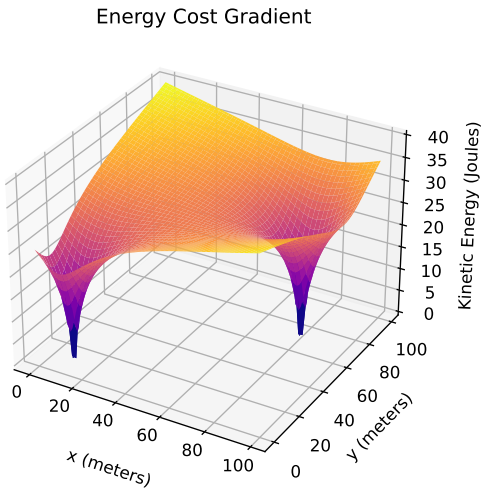


Figure 8.25: The energy cost gradient used as a cost constraint in simulation with Alg. 7.

Remarks on Results

Fig. 8.23 shows the final path taken by the vehicle after applying Alg. 7. The structure of the DAGs can be seen in the final path. Fig. 8.24 shows the mean

absolute error (MAE) of the field estimates over time. The sharp drops in MAE correspond to new field estimates that incorporated one or more particularly "useful" field measurements. It is important to consider that the MAE can be plotted with respect to time, number of measurements made, the percent of total available discrete measurements, or other metrics. The field attribute being measured could be depth, or it could be another aspect of a local environment that will not change rapidly over time (such as the concentration of certain elements or chemical compounds in the soil).

8.7 Numerical Comparison of Path Planners and Spatial Estimation

Similar to the approaches outlined in [32], [83], and [74], a handful of path planners are compared to the path planner based on the graph search method shown above. The comparisons are made in terms of the reduction of the mean square error (MSE) versus the percentage of field measured (the number of measured discrete points divided by the total number of discrete points) over time, as well as the MSE versus the time spent exploring, respectively. All path planners are choosing waypoints based on the variance of their most recent respective field estimates. That is, the path planners are informed by a spatial estimation technique, such as Ordinary Kriging, GPR, POK, or PGPR. The path planners include 1) a "greedy" or myopic planner that travels to the nearest point of highest variance within a prescribed radius of the vehicle's location, 2) a simple zig-zag pattern that the vehice follows regardless of the variance of the field estimate, and 3) the highest variance (HV) Bellman Ford path planner that plans an optimal route that maximizes the variance along the path with respect to a local graph

(as outlined above) towards the current global point of highest variance.

Zig-zag Path Planner

This is the simplest path planner of the three. A set of waypoints are statically defined that makeup a zig-zag path between the vehicle's starting position $\mathbf{w}_0 = \begin{bmatrix} x_0 & y_0 \end{bmatrix}$ and a pre-defined stopping position $\mathbf{w}_f = \begin{bmatrix} x_f & y_f \end{bmatrix}$. The starting position $\mathbf{w}_0$ is defined in this work as the lower left corner or minimum East and North position of the field. The stopping position $\mathbf{w}_f$ is defined as the top right or maximum East and North position of the field. The Bellman-ford graph search Alg. 6 can be run once assuming some ds and no distance cost to form a zig-zag path as mentioned earlier. This generates a static path consisting of an unchanging list of waypoints for the ASV to follow.

Myopic Path Planner

The myopic path planner is defined in Alg. 8. It essentially travels to the point of immediate highest variance at some distance away, and does not incorporate future considerations of overlapping paths over time. This algorithm is different than the highest variance path planner, in that it only picks from among points that are a certain radius way from the vehicle position. By contrast the highest variance path planner chooses from all the points in the field, regardless of vehicle position.

Algorithm 8 Myopic Path Planner

1: Move towards the next waypoint
2: Make measurements along the way to the next waypoint
3: For every nth new measurement estimate the field
4: Once at the next waypoint designate the new next waypoint as the point r distance away that possesses the highest estimate variance.
5: repeat steps 1 - 4
6: **return** $\hat{\mathbf{Z}}_k, \hat{\mathbf{V}}_k$

HV Bellman Ford Path Planner

The HV Bellman Ford path planner is defined in Alg. 9. The idea is to continually calculate the optimal path with respect to the latest field estimate variance, as simlar to Alg. 7.

Algorithm 9 HV Bellman Ford Path Planner

1: Move towards the next waypoint
2: Make measurements along the way to the next waypoint
3: For every nth new measurement in between waypoints:
4: estimate the field
5: calculate the optimal path to the next waypoint using the Bellman Ford graph search Alg. 6 that maximizes the variance along the path
6: Once at the next waypoint set the new next waypoint as the global point that possesses the highest estimate variance.
7: repeat steps 1 - 6
8: **return** $\hat{\mathbf{Z}}_k, \hat{\mathbf{V}}_k$

8.7.1 Simulation Procedure

A number of different simulations were run to test combinations of path planners with spatial estimators. The spatial estimators included GPR, PGPR, Ordinary Kriging (OK) and Partitioned Ordinary Kriging (POK). These tests included making a batch of fields represented by GRFs (see subsection 8.5.2). One of the parameters of the tests was the spatial resolution for the fields ds. A batch of 100 fields were generated and explored with simulated ASVs using the augmented

Nomoto ship steering model from Ch. 5 with the aforementioned path planners and spatial estimators. One batch of 100 fields was created with $\mathrm{d}s = 5.0$ meters for a total of 266 discrete points per field, and another batch of 100 fields was created with $\mathrm{d}s = 2.0$ meters for a total of 1,518 discrete points per field. Fig. 8.27 shows an example of a batch of fields used in the simulation. A high level outline of the simulation procedure is shown in Fig. 8.26.

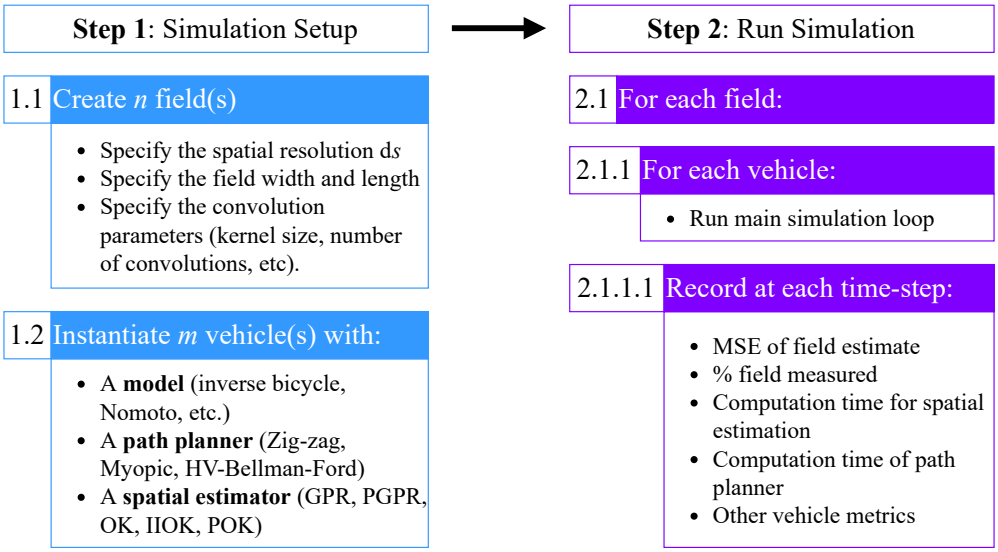


Figure 8.26: A high-level overview of the simulation procedure.

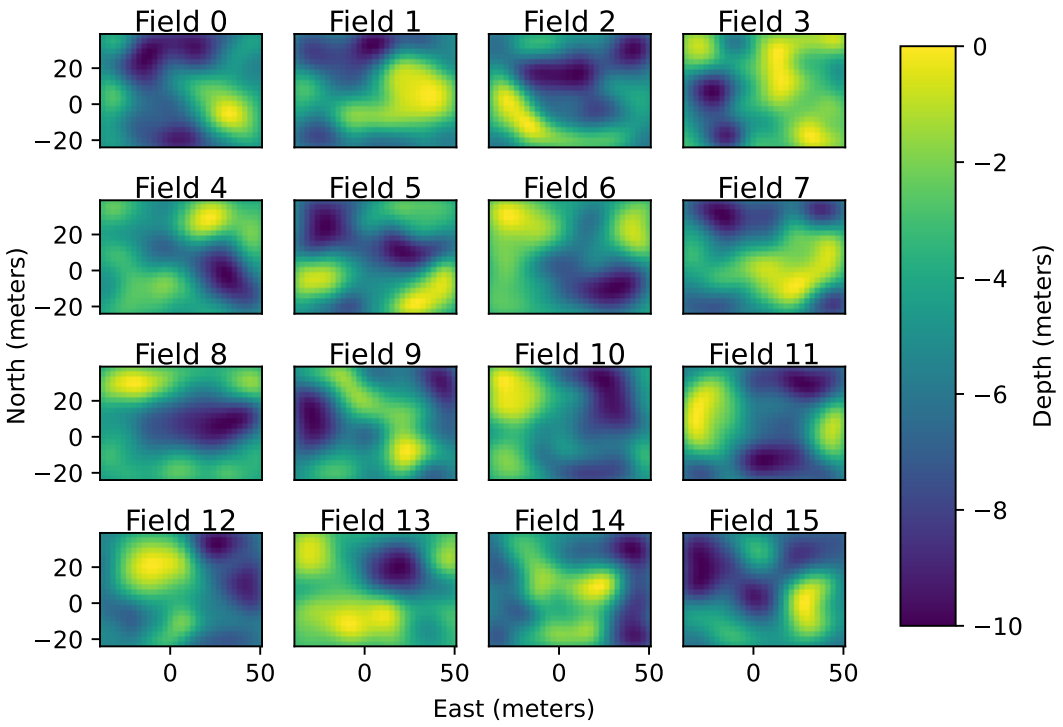
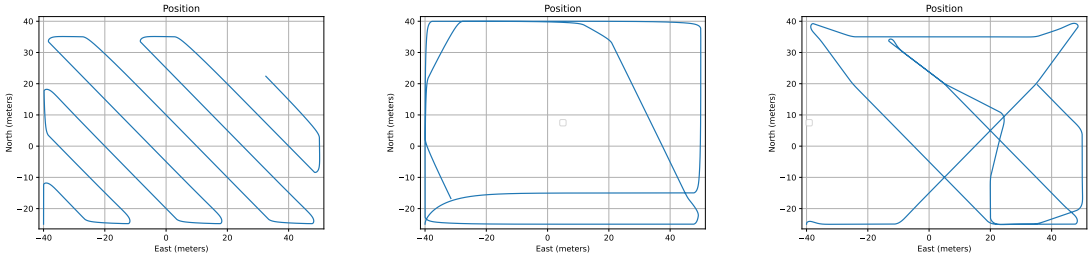


Figure 8.27: An example of 16, rather than 100 GRFs acting as the "true" simulated fields for the ASVs to explore and estimate during simulation.

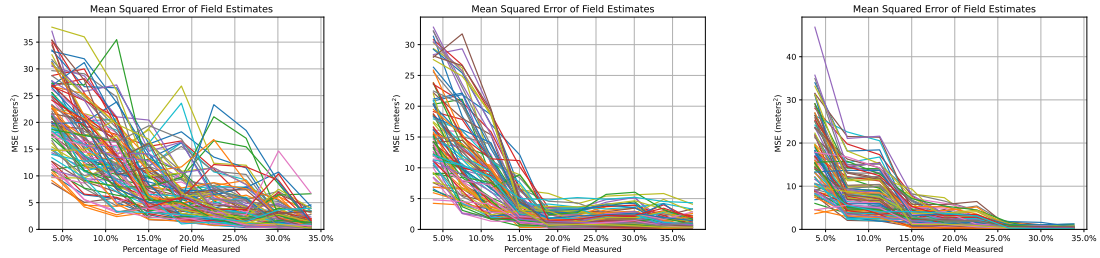
8.7.2 Simulation Results

After an ASV explored a field, the MSE versus the percentage of the field measured and the MSE versus time was recorded. For example Fig. 8.28 shows the paths of three separate ASVs after exploring approximately 40% of a field. Fig 8.29 shows all of the MSE signals versus the percentage of the field measured corresponding to each of the fields in the 100 batches. Figs. 8.31 and 8.32 are the results of the simulation batches.



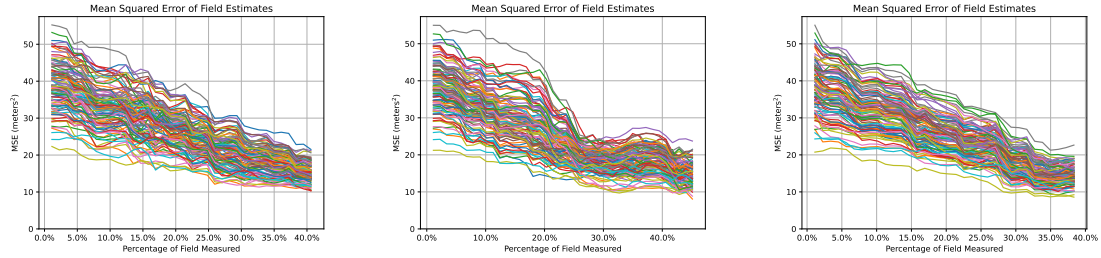
(a) **(b)** **(c)**

Figure 8.28: The paths (starting in the bottom left corner) of three separate ASVs after exploring $\sim 40\%$ of a field. (a) is the Zig-zag path planner, (b) is the myopic path planner, and (c) is the HV Bellman Ford path planner.



(a) **(b)** **(c)**

Figure 8.29: All MSE signals versus the percent field measured of the three separate ASVs after exploring $\sim 40\%$ of a field. (a) is the Zig-zag path planner, (b) is the myopic path planner, and (c) is the HV Bellman Ford path planner. The spatial estimator was GPR. The field resolution $\mathrm{d}s = 5.0$ meters. Each field had 266 discrete points.



(a) **(b)** **(c)**

Figure 8.30: All MSE signals versus the percent field measured of the three separate ASVs after exploring $\sim 40\%$ of a field. (a) is the Zig-zag path planner, (b) is the myopic path planner, and (c) is the HV Bellman Ford path planner. The spatial estimator was PGPR. The field resolution $\mathrm{d}s = 5.0$ meters. Each field had 266 discrete points.

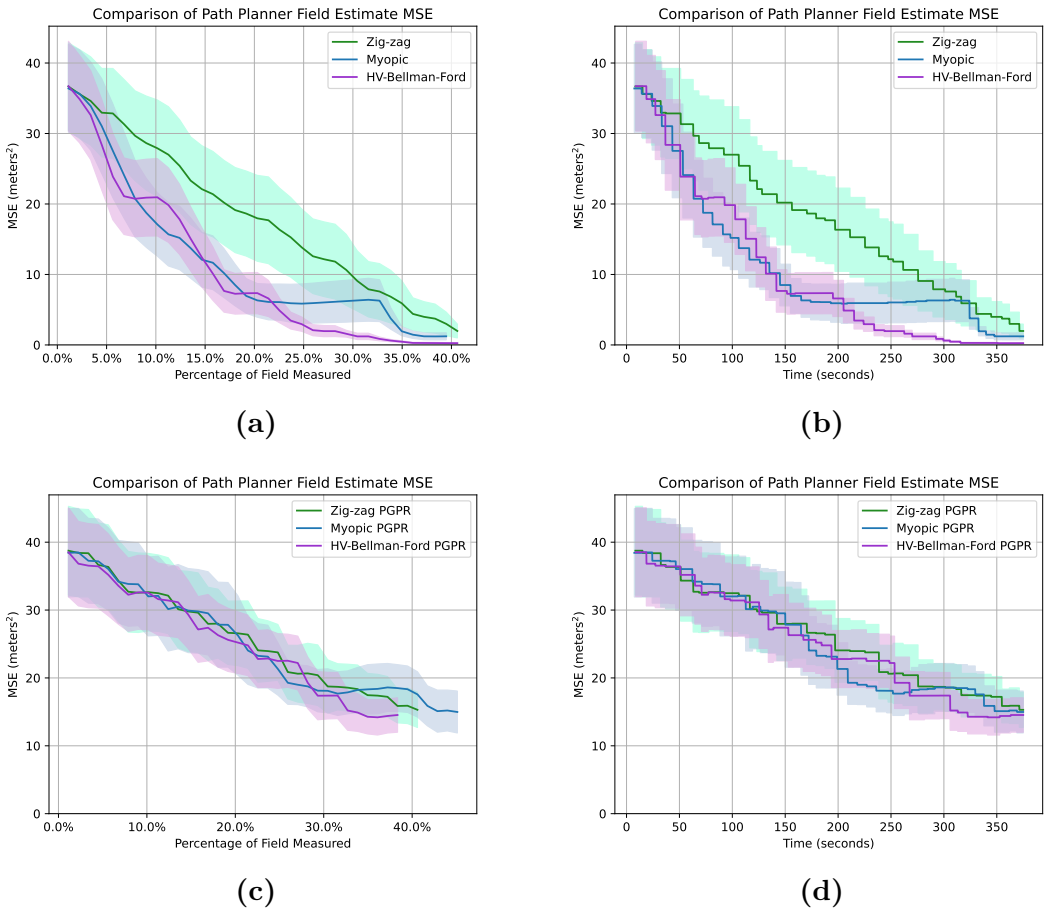


Figure 8.31: The MSE signals from Fig. 8.29 and Fig. 8.30 were averaged, and their standard deviations were computed. These signals were plotted as a function of percent area explored (a) and (c), and as a function of time (b) and (d). This is the result of running path planners on 100 separate GRFs with $ds = 5.0$ meters for a total of 266 discrete points per field. The lighter colors indicate one standard deviation from the mean. (a) and (B) correspond to GPR, and (c) and (d) correspond to PGPR.

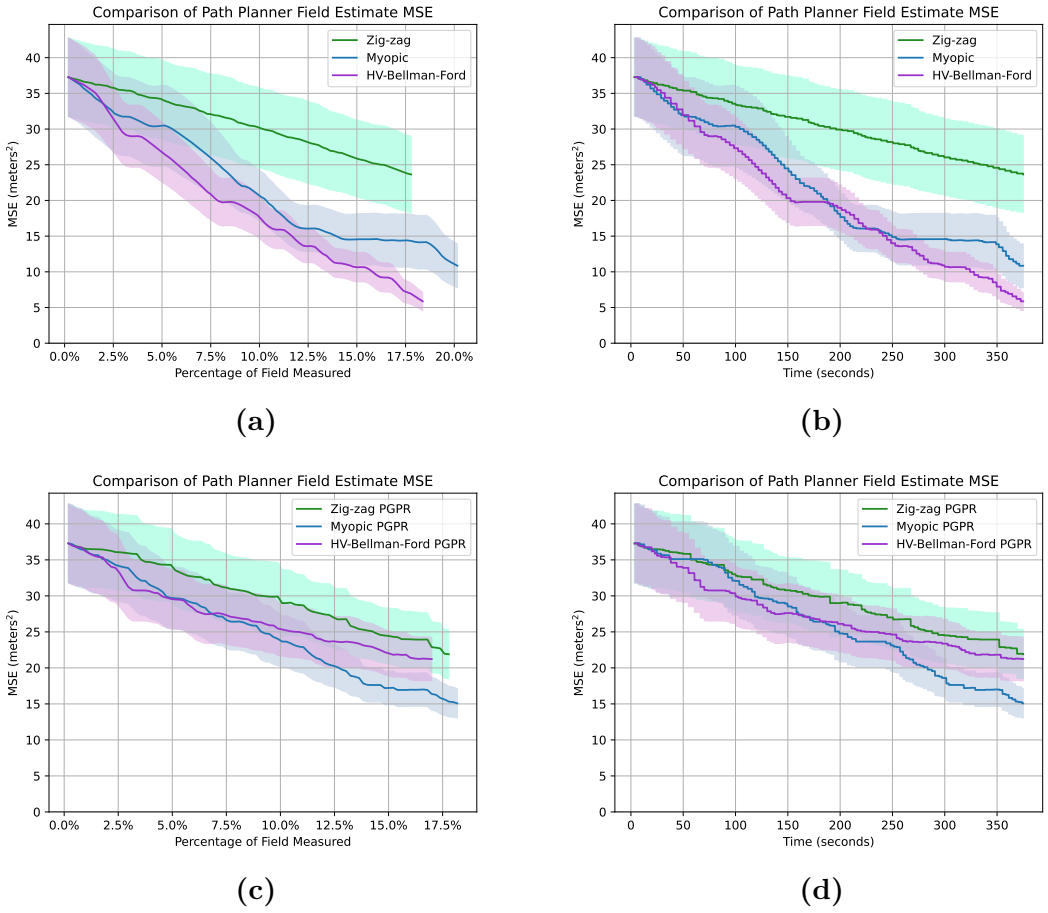


Figure 8.32: Path planner field estimate mean MSE comparisons versus percent area explored (a) and (c), and as a function of time (b) and (d). This is the result of running path planners on 100 separate GRFs with $\mathrm{d}s = 2.0$ meters for a total of 1,518 discrete points per field. The lighter colors indicate one standard deviation from the mean. (a) and (b) correspond to GPR, and (c) and (d) correspond to PGPR.

8.7.3 Conclusion and Caveats

This section introduced an optimal method to explore a field in terms of suppressing uncertainty by maximizing the variance along a path. The simulation results seem to indicate that the Alg. 9 outperforms both the myopic and zig-zag path planning algorithms. The differences are not overwhelming most likely because they are both a form of highest variance path planner. However, the key drawback to Alg. 9 is that it is computationally demanding compared to the other two path planners if it is informed by GPR or Ordinary Kriging. This is due to the computational cost of the Bellman Ford graph search algorithm, specifically when the DAG formed between the vehicle's current position and the new highest variance point is large.

One way around this problem is to tune the resolution of the DAG; as mentioned earlier, the number of nodes does not necessarily have to match the number of discrete points. The question then becomes; how do we decide the ratio of number of DAG nodes to the total number of discrete points or, how do we tune the size of the DAG? One way to make this choice is to base it off of the spatial auto-correlation of the estimated field thus far. That means that the variogram model's range could be used to proportionally tune the size of the DAG. Future work should explore this idea and other possible approximations of optimal paths with respect to variance along the path.

Another aspect for future work is to investigate the effects of the degree to which the vehicle explores the space between two points. This can be tuned depending on the weight of the variance versus the weight of the distance cost. This could be an adaptive feature based on the observed field over time, perhaps being proportional to the range of the variogram.

Lastly, the discussion of optimal exploration did not involve using multiple

vehicles and obstacle avoidance. Work by [74] outlines some methods for multi-vehicle exploration. Introducing this idea further complicates the problem of optimal exploration, but has obvious benefits in reducing overall exploration time. Obstacle avoidance is important for autonomous vehicles, especially if multiple vehicles are deployed and their paths depend on the surrounding environment. Future work should focus on incorporating obstacle avoidance that are application specific.

Chapter 9

Experimental Results

This chapter examines experimental results of applying Ordinary Kriging and POK on real-world data collected by the Slug 3 ASV. Their performance in terms of depth field estimation accuracy and computation time versus the number of points measured is discussed. This chapter also examines two custom-made autonomous surface vehicles, the Slug 2 and Slug 3 along with results from a number of experimental test-runs. Sec. 9.6 shows the autonomous waypoint tracking for the Slug 3. It had multiple successful test-runs, which show that the guidance algorithm and subsequent simulations from Ch. 7 can be considered validated to a high degree of confidence. Sec. 9.7 shows the effectiveness of the EKF for position estimation for the Slug 2 and Slug 3. Sec. 9.8 shows a speed estimate versus time of the Slug 2, also using the EKF. Sec 9.9 shows the heading angle estimate versus time of the Slug 3, comparing the estimate with the COG angle from the onboard GPS receiver.

9.0.1 Note on Experimental Results

The reader will note that this chapter does not hold *all* experimental results. Recall that throughout this thesis, there have been both simulation and experimental results shared within each chapter. There are yet more experimental results in the appendix of this thesis along with other supplemental material. This chapter highlights key findings within this research and the related experimental results.

9.1 Experimental Results

Work done by [83] compares various path planners informed by the Ordinary Kriging method. This section uses the field estimates to inform basic path planners. It should be noted that neither [83] nor [80] used real-world data to test their methods. Here we apply these same methods on real-world data to further the case of their applicability for online updates for autonomous vehicle exploration. Specifically, we seek to meet strict time requirements that are representative of real-world constraints (such as system battery life).

The goal is to develop a robust and real-time method for spatial estimation to inform various path planners used by the ASV.

9.1.1 ASV System Block Diagram

To measure the depth of the pond, an onboard micro-controller and raspberry pi are used for hardware peripheral interfacing, data logging, and the ability for autonomous navigation based on more computationally demanding spatial estimation and path planning. However, though the micro-controller interfaces with the servos, electronic speed controllers (ESCs), GPS, and the inertial measure-

ment unit (IMU), the depth sensor is connected to the raspberry pi directly, to simplify depth data logging. MAVLink messages are exchanged between the micro-controller and the raspberry pi. Messages include additional vehicle status information for logging, including attitude with respect to the LTP, GPS position, and other useful information. See Ch. 2 for the full design of the experimental platform.

9.1.2 Experimental Procedure

Collecting Real-World Data

Depth data was collected from a local pond with a depth echo-sounder onboard the ASV. The collected depth measurements were associated with GPS coordinates of the ASV's location. The data are presented in terms of the LTP, with the x and y-axes representing the East and North directions in meters respectively.

A portion of the data collected by the ASV was via remote operation, and other data was collected autonomously. The intent was to collect depth data along a pseudo-random path created arbitrarily by the operator, so as to validate the quality of two spatial estimation methods. If a spatial estimation method were to depend on a specific hard-coded path it might be susceptible to disturbances. Ideally, small disturbances while navigating should not - by themselves - significantly impact the quality of the spatial estimator. A pseudo-random path could be considered similar to the random waypoint path planner discussed in Ch. 8. Such a path is a useful first step for collecting data simply to check that a spatial estimation algorithm can work with real data.

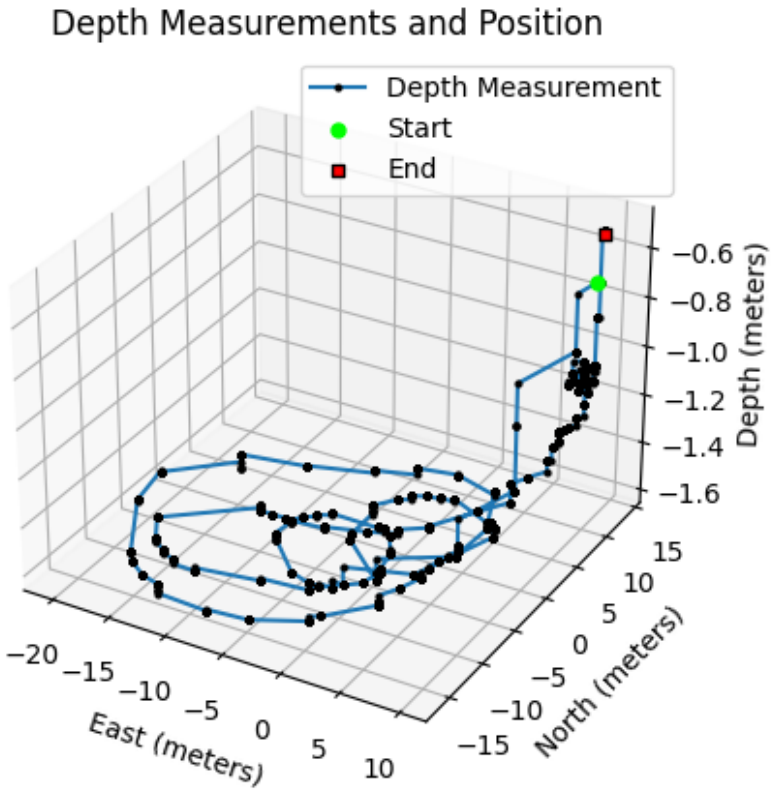


Figure 9.1: An example of real depth-sensed data corresponding to GPS location of the ASV, projected onto the LTP. The start (green) and stop (red) markers indicate where the ASV initially launched and returned. The black dots represent discrete depth measurements taken by the ASV using the onboard ping echo-sounder

Validation Metrics for OK versus POK

A subset of all depth measurements and corresponding GPS positions is used to form the first variograms. For instance, 25% of the total number of points measured may be used to initially determine the range, nugget, and sill of the variograms. This is not explicitly required, but can generally allow for more accurate starting predictions by both Ordinary Kriging and POK. Work by [83] and [74] also use an initial set of data points to form the starting variograms and covariance matrices respectively. A point that is estimated is compared to a direct measurement in the same location. The measured points are considered "true" points. True points come from separate data sets. The true depth measurements - in reality - have some noise (see Ch. 3). Comparing two measurements at the same point is avoided; the number of true points decreases if a new measurement occupies the same location as a true point. The reason for this is to avoid the case of comparing two measurements at the same point, because this would *not* facilitate determining the accuracy of the field estimate at that point.

Measurements are incrementally added as input to both Ordinary Kriging and POK. These are referred to as "scanned" points, having depth distance and East and North distance in meters. As the number of scanned points increase, the MAE is expected to decrease. An important factor is the rate at which the MAE converges towards zero, as the number of scanned points increases. True points

9.1.3 Results

Below are the comparisons between Ordinary Kriging, and POK in MAE and computation time, with real depth measurement data as input.

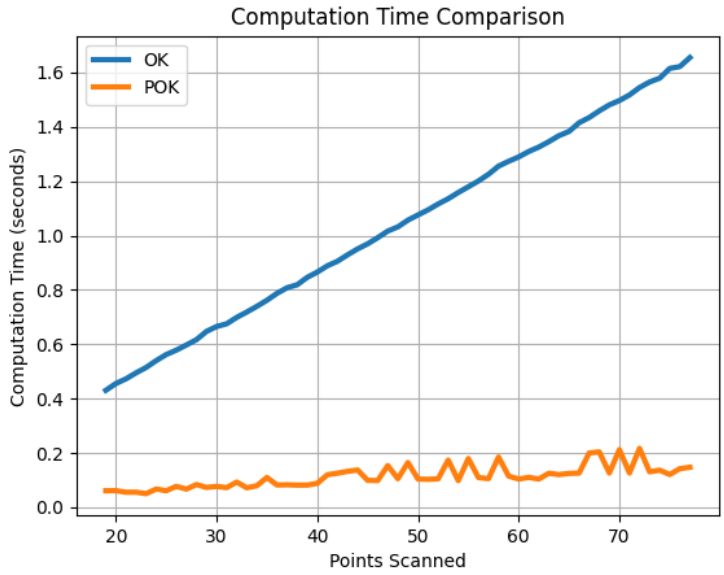


Figure 9.2: A comparison of computation time of Ordinary Kriging, and POK versus than number of depth measurements.

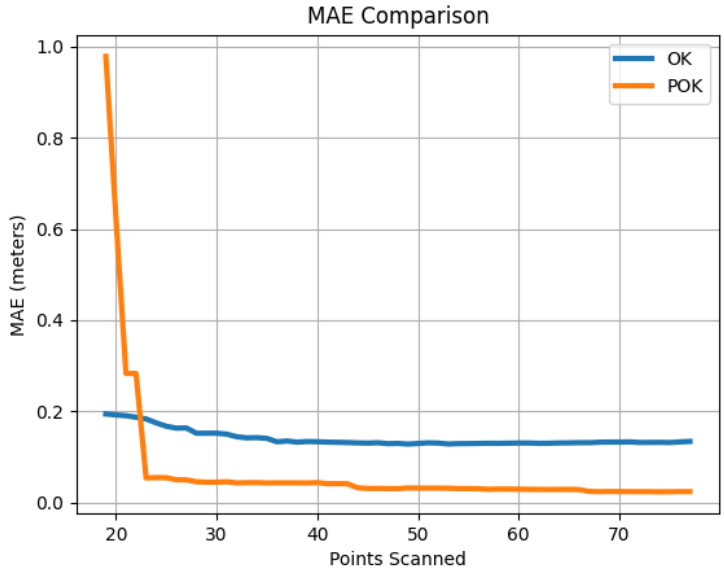


Figure 9.3: A comparison of MAE of Ordinary Kriging, and POK versus then number of depth measurements.

The same procedure resulting in Fig. 9.2 and Fig. 9.3 was repeated for other separate sets of depth measurement data, taken at different times and locations. Three separate data-sets are shown: (a), (b), and (c). Both Ordinary Kriging, and POK are run on each data-set independently, shown in Fig. 9.4 and Fig. 9.5.

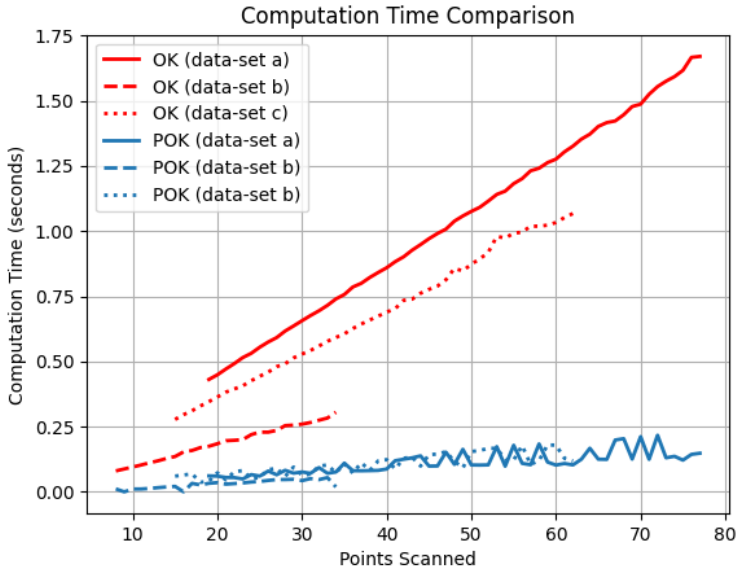


Figure 9.4: A comparison of computation time of ordinary krging, and POK versus then number of depth measurements for multiple separate sets of depth measurement data.

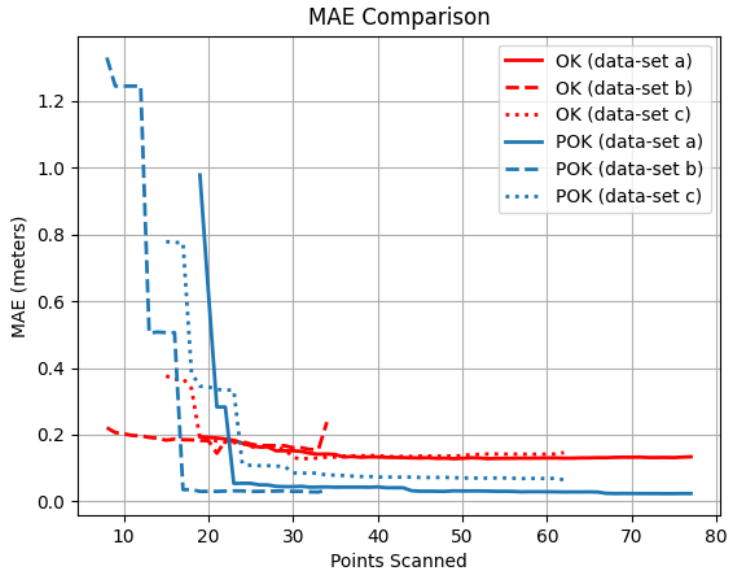


Figure 9.5: A comparison of MAE of ordinary krging, and POK versus then number of depth measurements for multiple separate sets of depth measurement data.

9.1.4 Remarks on POK

The POK method was demonstrated to possess favorable computation time compared to Ordinary Kriging, using real environmental data. The MAE comparison showed an initially higher error for POK, but after more data points are collected, POK appears to have lower MAE. The computation time is also favorable for online exploration.

9.2 GPR Experimental Field Reconstruction

Experimental depth measurements collected by the Slug 3 ASV were used as input to GPR to reconstruct the true depth field of a small body of water. Fig. 9.6 shows an example of a sparsely-measured depth field of a small body of water.

One method to reconstruct the depth field is to interpolate the depth mea-

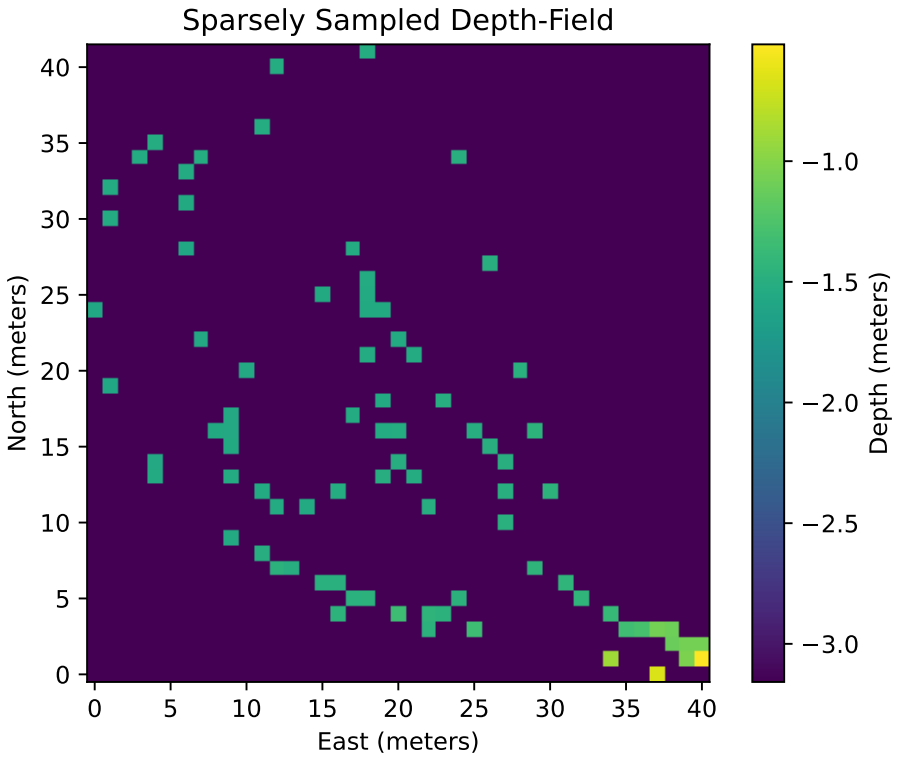


Figure 9.6: An example of a sparsely measured depth field of a small body of water

surement points in space. This can be done with Ordinary Kriging or GPR and their respective variations discussed in the Ch. 8. Fig. 9.6 shows the reconstructed depth field using GPR. The hyper-parameters were based on the empirical mean and standard of the data collected.

Fig. 9.6 and Fig. 9.7 show the depth measured by the ASV as it moved from shallow water (indicated by the brighter colors) to deeper water. The ASV moved in a large loop and then took measurements within that loop. The areas toward the edges of Fig. 9.7 are estimated to be similar to the measurements taken in the shallower water. Measurements taken near the center of the pond are deeper.

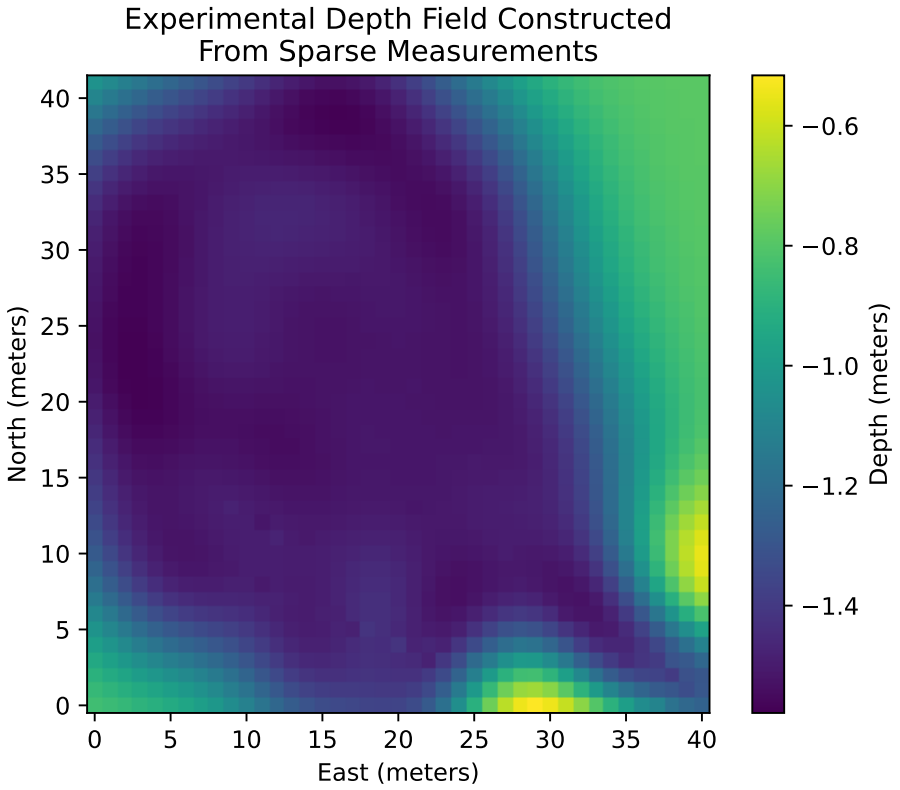


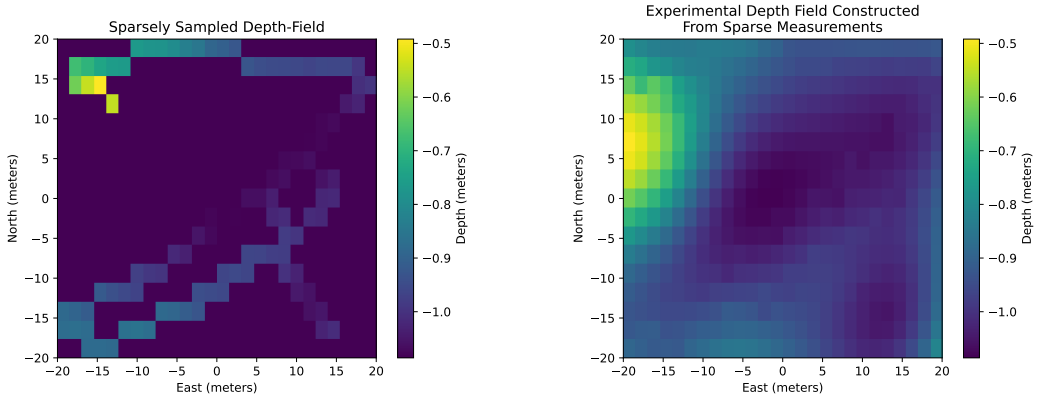
Figure 9.7: An experimental depth field reconstructed from depth measurements collected by the Slug 3 ASV. GPR was used to interpolate depth measurements in space. The hyper-parameters were based on the empirical mean and standard deviation of the collected data.

9.3 Experimental Field Reconstruction and Path Planning

Two path planners were used to guide the Slug 3 ASV, the Zig-zag and HV-Bellman-Ford path planner. The former is a static path planner, and the latter is dynamic.

9.3.1 Depth Measurements with Zig-zag Path Planner

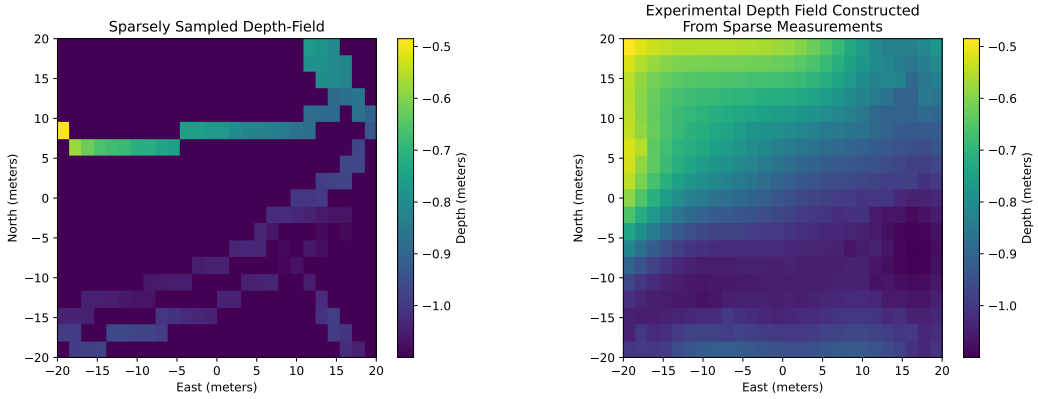
Below are depth field measurements (associated with the position of the Slug 3 ASV using the Zig-zag path planner and navigating autonomously) and depth field reconstructions based on these depth measurements, shown in Fig. 9.8 and Fig. 9.9



(a) Depth measurements collected autonomously using the Zig-zag path planner

(b) Reconstructed depth field from depth measurements using GPR

Figure 9.8: An experimental depth field reconstructed from depth measurements (left) collected by the Slug 3 ASV while navigating autonomously using the Zig-zag path planner. GPR was used to interpolate depth measurements in space (right) on board the Slug 3 during run-time. The hyper-parameters were based on the empirical mean and standard deviation of the collected data.



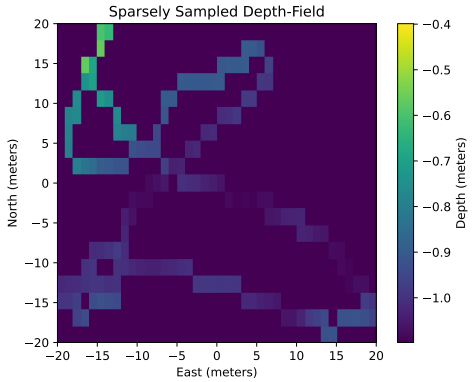
(a) Depth measurements collected autonomously using the Zig-zag path planner

(b) Reconstructed depth field from depth measurements using GPR

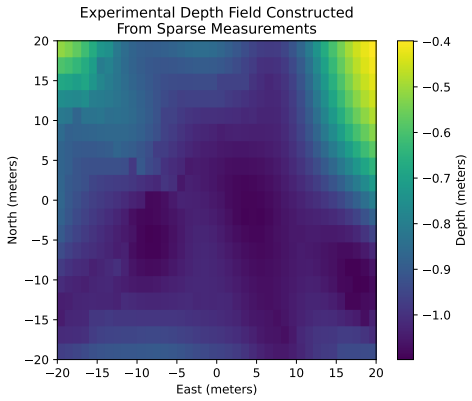
Figure 9.9: A second experimental depth field reconstructed from depth measurements (left) collected by the Slug 3 ASV while navigating autonomously using the Zig-zag path planner. GPR was used to interpolate depth measurements in space (right) on board the Slug 3 during run-time. The hyper-parameters were based on the empirical mean and standard deviation of the collected data.

9.3.2 Depth Measurements with HV-Bellman-Ford Path Planner

Below are depth field measurements (associated with the position of the Slug 3 ASV using the HV-Bellman-Ford path planner and navigating autonomously) and depth field reconstructions based on these depth measurements, shown in Fig. 9.10 and Fig. 9.11

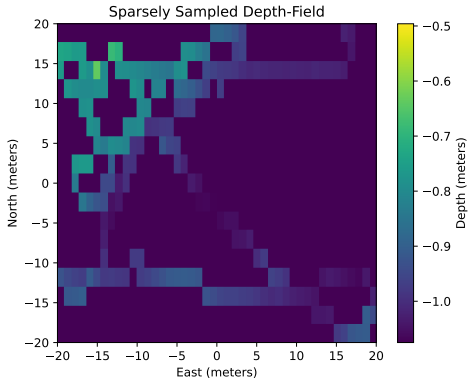


(a) Depth measurements collected autonomously using the HV-Bellman-Ford path planner

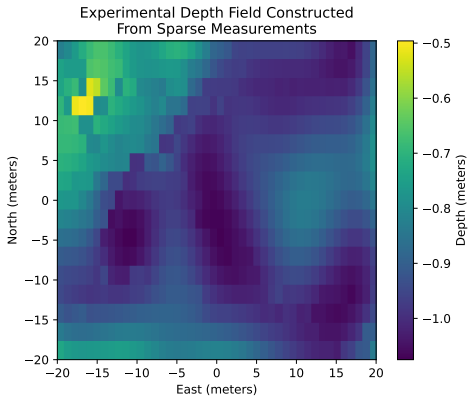


(b) Reconstructed depth field from depth measurements using GPR

Figure 9.10: An experimental depth field reconstructed from depth measurements (left) collected by the Slug 3 ASV while navigating autonomously using the HV-Bellman-Ford path planner. GPR was used to interpolate depth measurements in space (right) on board the Slug 3 during run-time. The hyper-parameters were based on the empirical mean and standard deviation of the collected data.



(a) Depth measurements collected autonomously using the HV-Bellman-Ford path planner



(b) Reconstructed depth field from depth measurements using GPR

Figure 9.11: A second experimental depth field reconstructed from depth measurements (left) collected by the Slug 3 ASV while navigating autonomously using the HV-Bellman-Ford path planner. GPR was used to interpolate depth measurements in space (right) on board the Slug 3 during run-time. The hyper-parameters were based on the empirical mean and standard deviation of the collected data.

9.4 MSE Spatial Estimation Comparison of Zig-zag and HV-Bellman-Ford

The Slug 3 ASV system was used to autonomously collect depth measurements of the Los Gatos Creek County Park pond number 2. Spatial estimation algorithms and path planners were paired, applied in autonomous exploration, and then compared. The idea was to compare a sub-set of the combinations of path planners and spatial estimators based on the simulation results Ch. 8. A procedure was created for comparing the spatial estimations made during run-time.

9.4.1 Procedure

This subsection outlines the procedure to reproduce this experiment. The first step was to make sure all systems and subsystems of the Slug 3 were functional by reviewing the pre-launch checklist, and conducting a quick remote controlled test to ensure that the vehicle could be recovered remotely in case of a malfunction. The second step was to re-launch the Slug 3, switching it to autonomous exploration mode, to explore the field using the Zig-zag path planner and a spatial estimator, such as GPR. The third step was to re-launch the Slug 3 again, still in autonomous exploration mode, and explore the field using the HV-Bellman-Ford path planner (Alg. 9). Estimates were made at run-time for both autonomous test-runs. The ground truth was a separate set of depth measurements from other test-runs taken in the same location and combined together. The depth-measurements and other data were recorded. Fig. 9.12 shows the high-level steps used to collect the experimental results.

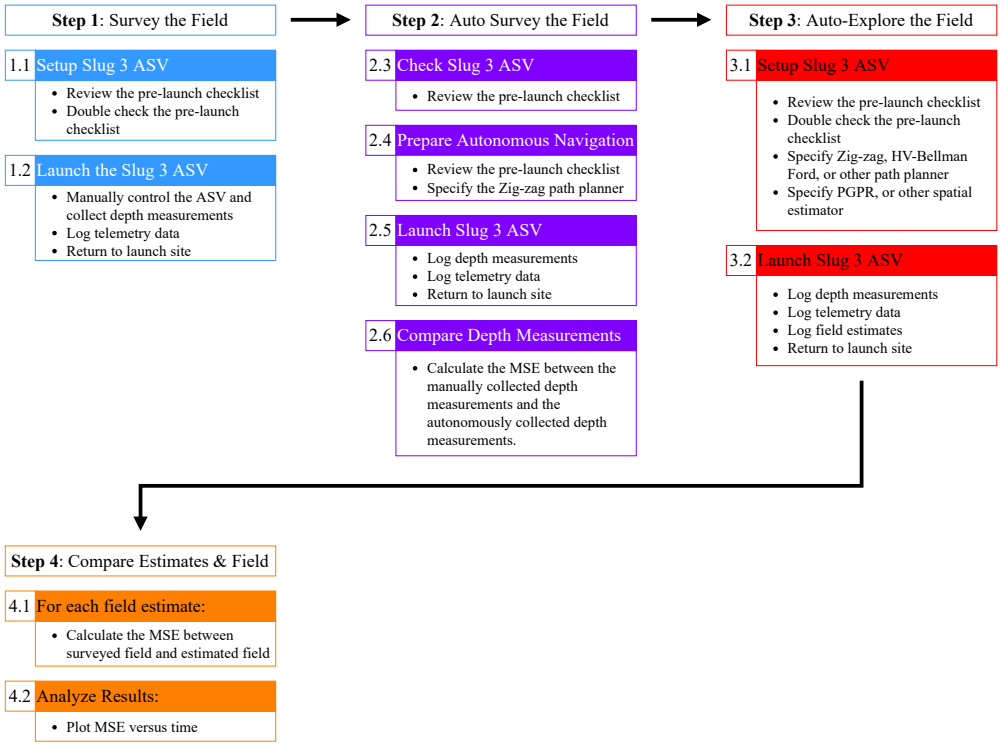


Figure 9.12: The high-level overview of the experimental procedure.

9.4.2 Experimental Results

Fig. 9.13 shows the comparison of two MSE signals from two separate autonomous test-runs of the Slug 3 ASV. It is clear that the HV-Bellman-Ford path planner paired with the PGPR spatial estimator had a more consistent rate of decreasing MSE. Fig. 9.14 shows the true depth field. It is a combination of a number of separate depth-data collected form separate test-runs of the Slug 3. Fig. 9.15 shows the estimate of the true depth field using the PGPR estimator paried with the HV-Bellman-Ford path planner. The HV-Bellman-Ford path planner and Zig-zag path planner were chosen because the former seemed like the most complicated, novel, and most capable path planner (based on the simulations in Ch. 8 and the latter seemed like the most simple and practical path planner to implement. In short, they both seemed to be on either end of

spectrum of complexity vs simplicity. The POK and PGPR algorithms were chosen because they both highlight one of the key points of this thesis; partitioning the computationally demanding aspects of certain spatial estimation algorithms can help extend their use in real-time autonomous vehicle applications. Essentially, both algorithms were chosen because they scale better with the number of measurements (see Ch. 8).

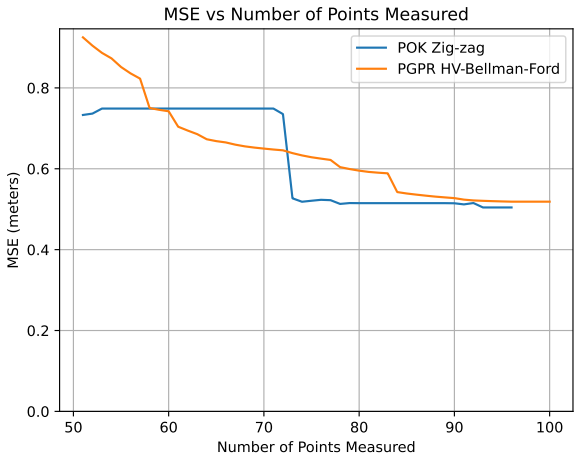


Figure 9.13: MSE versus number comparison between two autonomous test-runs of the Slug 3. One test run used the Zig-zag path planner (static) and the other used the HV-Bellman-Ford path path planner (dynamic).

For the sake of brevity a pairing between an estimation method and a path planner will be referred to as an exploration method. The comparison in Fig. 9.13 shows that the PGPR HV-Bellman-Ford exploration method starts with an initially higher MSE signal, but decreases at a more consistent rate than the POK Zig-zag method. The two exploration methods have a comparable final MSE. Interestingly the POK Zig-zag method shows an abrupt drop in MSE. This might be due to the POK Zig-zag combination finding a particularly informative (or "lucky") measurement that the PGPR HV-Bellman-Ford happened to miss. However, it may be the case that given a smaller or larger field the MSE performance

outcome could vary. Future work should explore the contribution of the scale of the field in relation to the exploration performance. Conceivably a larger field or larger number of field measurements might point towards better performance from the PGPR HV-Bellman-Ford method. Furthermore POK may partition the field differently than PGPR if the input measurements differ which would imply that the range of the variograms also differed. If the partitions are different it is possible for a local field estimate to be less or more accurate. Large drops in MSE were observed in simulation.

True Depth Field vs Final Estimated Field

The measurement-only depth field is shown in Fig. 9.14 and the estimated field is shown in Fig. 9.15. This is meant as visual indicator to help show that the measured depth field visually corresponds to the final PGPR estimate; the MSE signals do a reasonable job to reflect the degree of similarity between true and estimated depth fields.

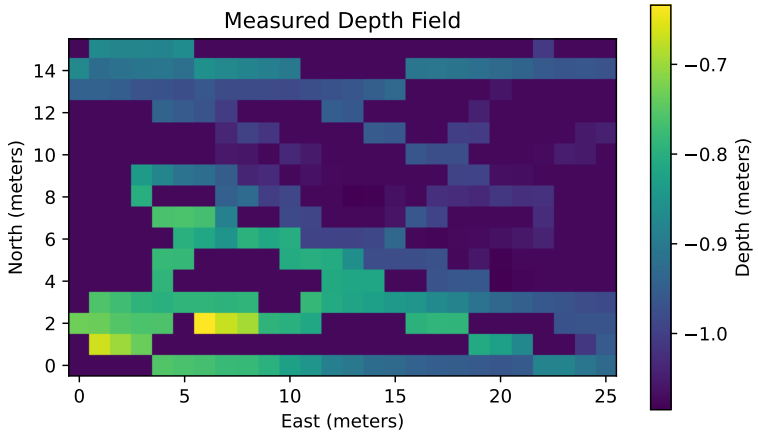


Figure 9.14: The ”true“ depth field used to compare the spatial estimates at run-time. This is a combination of different separate test-run depth data.

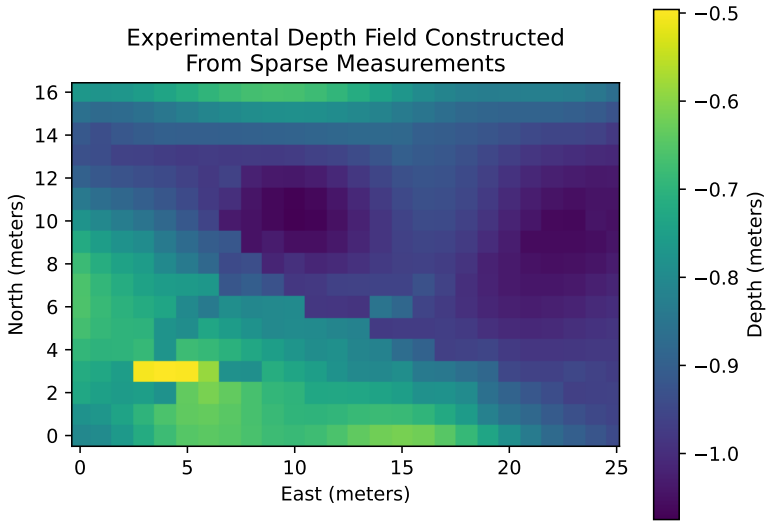


Figure 9.15: An estimate of the true depth field using the PGPR estimator paried with the HV-Bellman-Ford path planner. There is a clear resemblance to the measurement-only depth field shown in Fig. 9.14; the shallow lower left region shows good agreement, and the center and center right regions are of similar lower depth.

This section highlighted the ability of the Slug 3 ASV to use the HV-Bellman-Ford path planner and the PGPR spatial estimator to estimate the depth field of the Los Gatos Creek County Park pond number 2. The MSE signals shown in this section agree with those shown in simulation in Ch. 8. While this is a useful comparison, it is a small-scale experiment. A larger area for exploration and more comparisons between path planners and spatial estimators is required for more accurate insights.

9.5 Comparing all Combinations of Path Planners and Spatial Estimators

All combinations of path planners and spatial estimators were compared experimentally (see Ch. 8 for simulation comparison results). The same procedure, shown in Fig. 9.12, was applied. The Slug 3 was used to measure the depth at different points within a specific area in the Los Gatos Creek County Park pond number 2. The raw depth measurements associated with using different path planners (Zig-zag, Myopic, and HV-Bellman-Ford) is shown in Fig. 9.16. Data including the depth measurements, field estimates, and other information were recorded for each experimental test-run of the Slug 3. Fig. 9.17 shows the MSE for all combinations of path planners and spatial estimators.

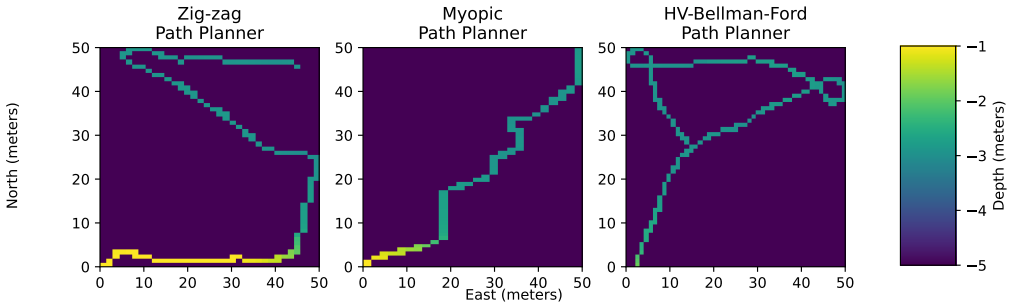


Figure 9.16: Experimental depth data recorded by the Slug 3 ASV.

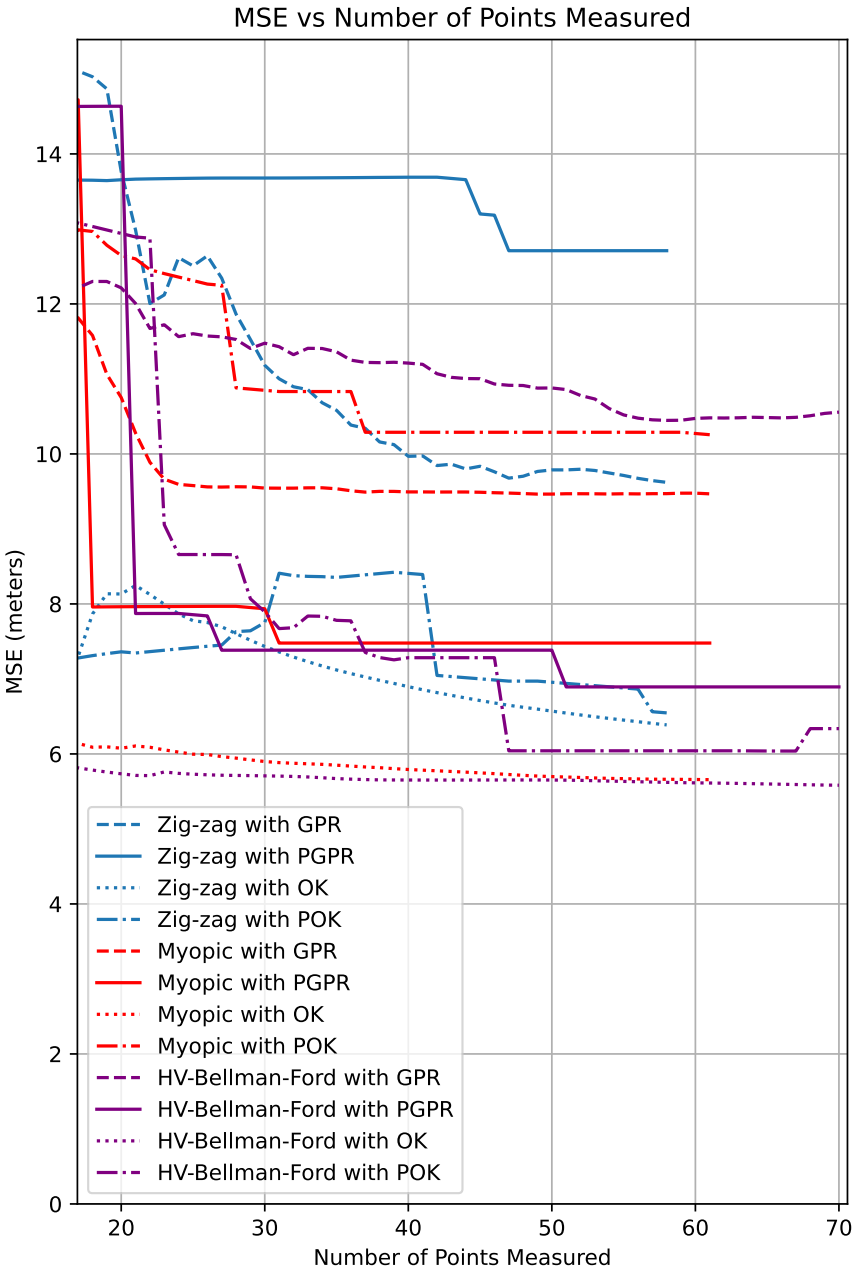


Figure 9.17: The MSE of all path planner and spatial estimator combinations from experimental data collected by the Slug 3.

Table 9.1 shows the final MSE values for all combinations of path planners and spatial estimators. It appears that there is a difference between Fig. 9.5 and Fig. 9.17. This is likely due to the fact that the field used in the former was relatively flat (see Fig. 9.7), whereas the field explored in the latter had more variation (Fig. 9.15). This implies that field partitions that have yet to be estimated may, by default, possess values that happens to results in a low field estimate error. Note that this is not the case for a field with higher variation, which is evident in the difference in estimate error in Fig. 9.13.

Table 9.1: Final MSE for Path Planner and Spatial Estimator Pairs

	GPR	PGPR	OK	POK
Zig-zag	9.65	12.71	6.40	6.57
Myopic	9.48	7.50	5.67	10.27
HV-Bellman-Ford	10.06	6.90	5.61	6.33

Final Considerations

In terms of MSE as a function of the number of measurements made, OK performs best when paired with HV-Bellman-Ford and the Myopic path planners. OK does not perform as well when paired with the Zig-zag path planner. POK shows a large decrease in MSE when paired with HV-Bellman-Ford. It appears that the POK and Zig-zag combination initially has a low MSE, but then increases after 30 measurements, and finally decreases close to OK and Zig-zag at approximately 57 measurements. This is likely due to the first estimates being initially "lucky" with very few measurements. It was shown in Ch. 8 that the computation time for OK scales poorly compared to the other spatial estimators (see Fig. 8.14). Note that the overall MSE in Fig. 9.17 is greater than Fig. 9.13. This is due to the fact that the Fig. 9.17 is showing estimate results for a larger

area and with comparable fewer measurements.

Considering both computation time and final MSE, the best choice for a combination of path planner and spatial estimator appears to be HV-Bellman-Ford paired with POK or PGPR. HV-Bellman-Ford and OK show the lowest final MSE value, but OK scales poorly with many measurements (see Ch. 8). Both HV-Bellman-Ford paired with POK or PGPR show significant drops in MSE, scale well with many measurements, and have final MSE values that are comparable with OK. The POK and Myopic combination and the PGPR and Zigzag combination both have higher final MSE values. Between choosing between HV-Bellman-Ford paired with POK or PGPR, the final best choice is the combination of HV-Bellman-Ford paired with POK. This is because POK inherently requires calculating the variogram, whereas PGPR makes use of the variogram as an *additional* computation step. The best path planner and spatial estimator is HV-Bellman-Ford paired with POK, based on these results.

9.6 Autonomous Waypoint Tracking

The Slug 3 ASV system was used in extensive testing in the Los Gatos Creek County Park pond number 2 (it did not have a name at the time of this writing). This location was chosen based on a number of factors: 1) it was close to the author's home, thus reducing personal cost of transportation, 2) it was one of the few locations that specifically allowed autonomous and remote controlled boating via permit and inspection from a County Park official, and 3) it was large enough ($\sim$ 75m $\times$ 120m) to facilitate and be representative of a real-world environment that the Slug 3 could be deployed in. Note that the size of the pond would vary slightly depending on water level, rainfall, the water reserves in the park water treatment facility (located nearby), and the amount of pond scum. Pond scum

in particular posed a problem because it could become so dense that it could stall the propeller motors and in one instance one of the onboard electronic speed controllers exploded in a cloud of smoke due to sinking too much current while stalling. As such, testing was conducted on days and in areas with little to no pond scum.

This location was repeatedly visited for testing autonomous waypoint tracking for both static and dynamic path planners; a zig-zag pattern and the HV-Bellman-Ford algorithm (Alg. 9). The mean and standard deviation associated with the cross-track error for the waypoint-tracking controller (Alg. 1) is shown and discussed in this section. The waypoint-tracking controller was used with the Zig-zag path planner as well as the HV-Bellman-Ford path planner. Recall that the waypoint-tracking controller is agnostic of which path planner is used.

The results of this section show a mean cross-track error for all autonomous test-runs that is less than 0.9 meters with a standard deviation less than 2.5 meters.

9.6.1 Zig-zag Path Planner Waypoint Tracking

The position of the Slug 3 ASV during experimental test-runs was recorded. Fig. 9.18 shows the position of the Slug 3 navigating autonomously, using the Zig-zag path planner. The cross-track error is presented as histograms in Fig. 9.20 and Fig. 9.21 to articulate the consistency of the waypoint-tracking controller when informed by the Zig-zag path planner. Fig. 9.19 shows a Zig-zag path with more graph edges and a corresponding smaller node separation.

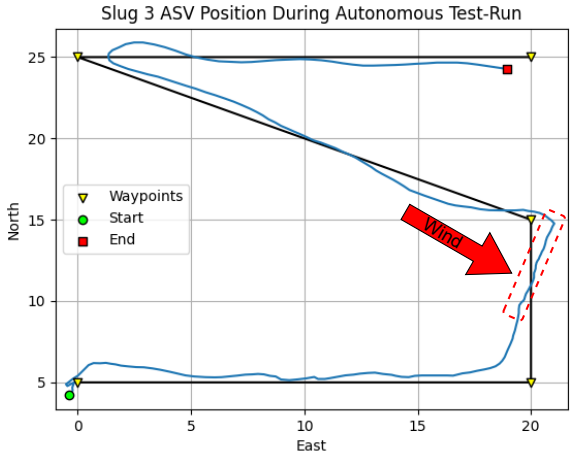


Figure 9.18: A position plot of the Slug 3 ASV (blue) as it autonomously navigates the desired Zig-zag waypoints (yellow) with a minimum node separation of 7ds. This is test-run **6a** for the Slug 3. Note that a gust of wind (red arrow) began to force the Slug 3 off course near the 3rd waypoint, resulting in the linear drift.

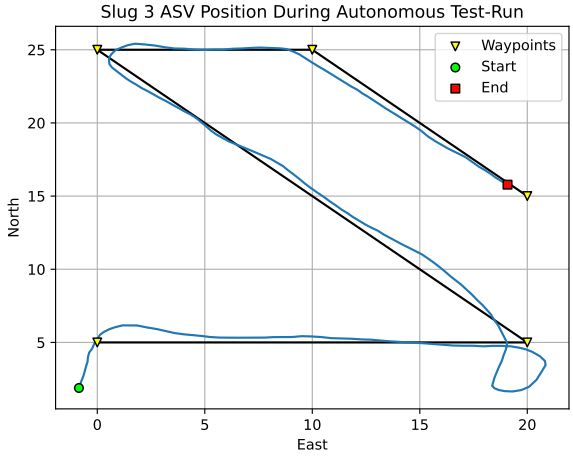


Figure 9.19: A position plot of the Slug 3 ASV (blue) as it autonomously navigates the desired Zig-zag waypoints (yellow) with a minimum node separation of 5ds. Note that the first circular arc-turn shows a loop instead of an arc; this is a fail-safe feature to re-acquire the desired path if a waypoint transition results in a cross-track error that is too high. This is test-run **6e** for the Slug 3.

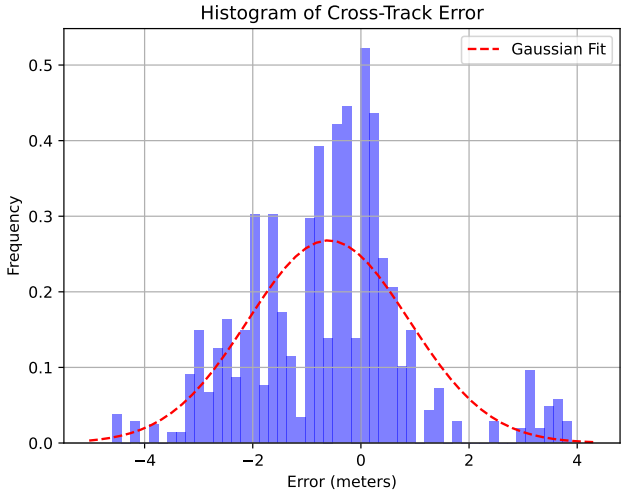


Figure 9.20: Histogram and Gaussian fit of the cross-track error of the Slug 3 ASV system during an experimental test-run (test-run **6a**) with the waypoint-tracking controller informed by the Zig-zag path planner. This data test-run was done in the Los Gatos Creek County Park pond number 2.

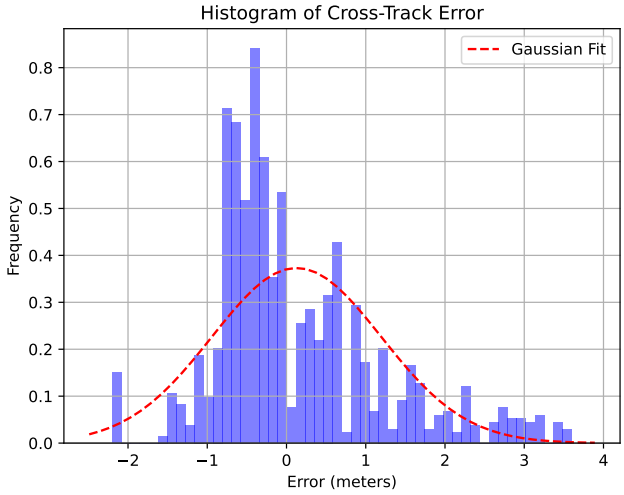


Figure 9.21: Histogram and Gaussian fit of the cross-track error of the Slug 3 ASV system during an experimental test-run (test-run **6e**) with the waypoint-tracking controller informed by the Zig-zag path planner. This data test-run was done in the Los Gatos Creek County Park pond number 2.

9.6.2 HV-Bellman-Ford Path Planner Waypoint Tracking

Fig. 9.22 shows the position of the Slug 3 navigating autonomously, using the HV-Bellman-Ford path planner. The cross-track error is presented as histograms in Fig. 9.24 and Fig. 9.23.

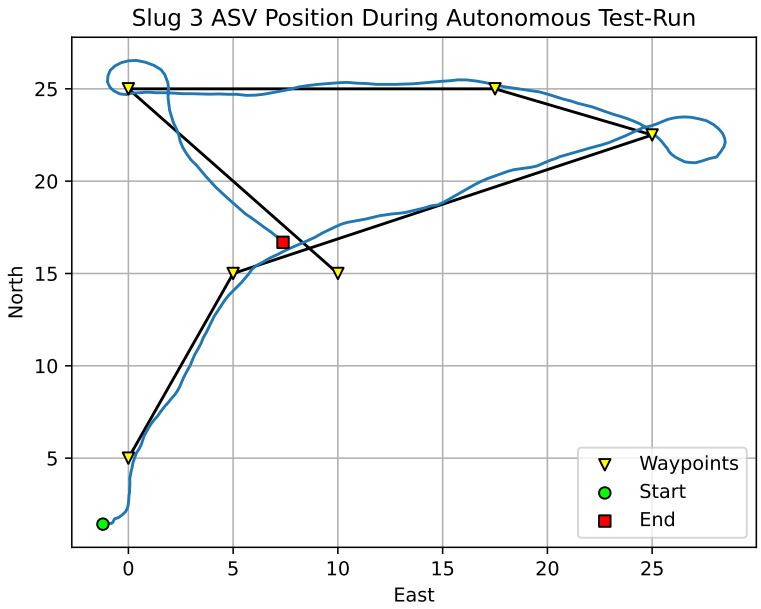


Figure 9.22: Another position plot of the Slug 3 ASV (blue) as it autonomously navigates the desired HV-Bellman-Ford determined waypoints (yellow).

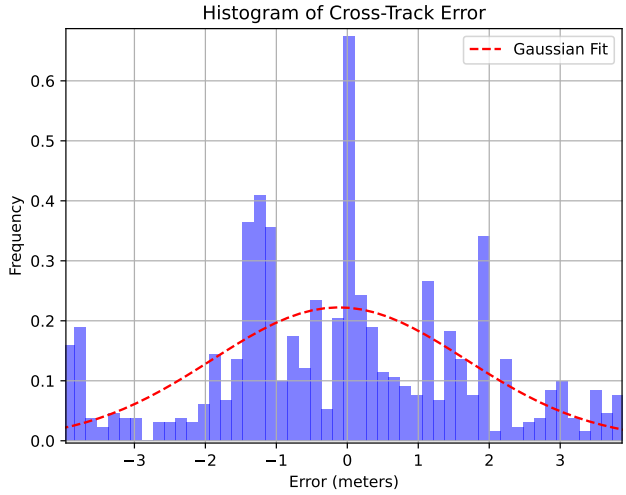


Figure 9.23: Histogram and Gaussian fit of the cross-track error of the Slug 3 ASV system during an experimental test-run (test-run **6d**) with the waypoint-tracking controller informed by the HV-Bellman-Ford path planner. This data test-run was done in the Los Gatos Creek County Park pond number 2. Note that the histogram excludes the looping segments of the position plot when acquiring the next line segment.

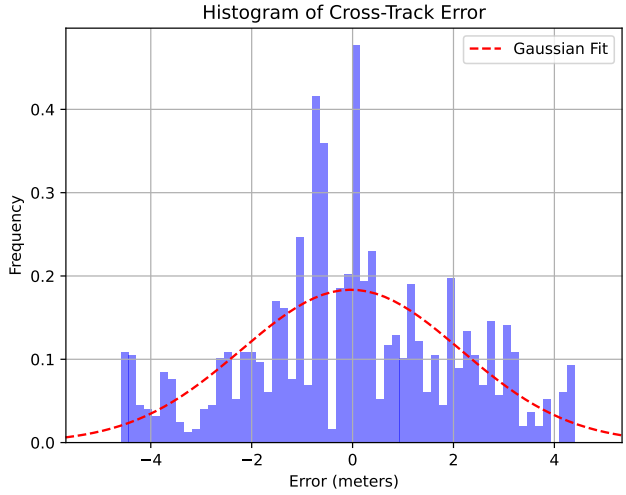


Figure 9.24: Histogram and Gaussian fit of the cross-track error of the Slug 3 ASV system during an experimental test-run (test-run **5g**) with the waypoint-tracking controller informed by the HV-Bellman-Ford path planner. This data test-run was done in the Los Gatos Creek County Park pond number 2.

Table 9.2 shows that the Slug 3 ASV is capable of a mean cross track error less than 0.9 meters, a median less than 0.3 meters, and a standard deviation of approximately 2.459 meters.

Table 9.2: Mean and standard deviation of cross-track error

	μ (Meters)	σ (Meters)
Zig-zag (test-run **6a**)	-0.599	1.488
Zig-zag (test-run **6e**)	0.127	1.069
HV-Bellman-Ford (test-run **6d**)	-0.116	1.794
HV-Bellman-Ford (test-run **5g**)	-0.190	2.332

Remarks on Waypoint Tracking

The static Zig-zag waypoint tracking showed relatively good performance. The tracking quality seems to change depending on the space between vertices. Smaller spacing results in poorer tracking. It should be noted that these test runs were susceptible to inconsistent environmental disturbances (i.e.: wind). Ideally testing was done at times with lower wind speeds, but many tests were conducted with wind speeds ranging between 4 and 12 miles per hour. All Zig-zag paths were tracked within a standard deviation of < 1.5 meters. This is $\sim 50\%$ better than the accuracy of the onboard GPS unit ($\sigma = 2.5$ meters).

The dynamic tracking with the HV-Bellman-Ford path planner was less accurate. This is likely due to the abrupt waypoint generation that is inherent to the path planner. In Fig. 9.22 there are sharp acute angles and the space between vertices can be very small.

9.7 Position Estimation

The Slug 3 had its position estimated using the EKF described in Ch. 6 and Ch. 7. Fig. 9.25 show data logs of the Slug 3's GPS position along with the EKF position estimates.

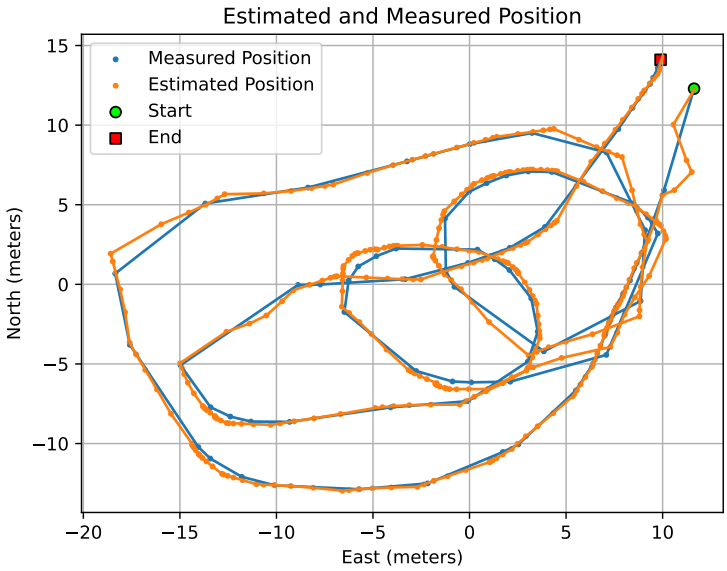


Figure 9.25: GPS position (blue) and EKF position estimates (orange) of the Slug 3

9.8 Speed Estimation

Using the EKF described in Ch. 6 and Ch. 7 the speed of the Slug 2 was estimated. Fig. 9.26 shows the estimated speed over time.

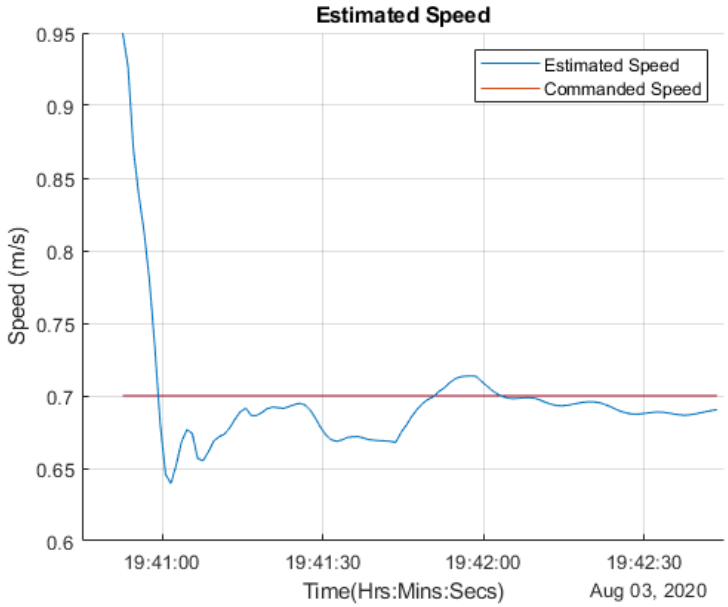


Figure 9.26: The estimated speed of the ASV using the EKF described in Ch. 7.

9.9 Heading Angle Estimation

Again using the EKF described in Ch. 6 and Ch. 7, the heading angle of the Slug 3 was also estimated. Fig. 9.27 and Fig. 9.28 show the estimated heading angle over time.

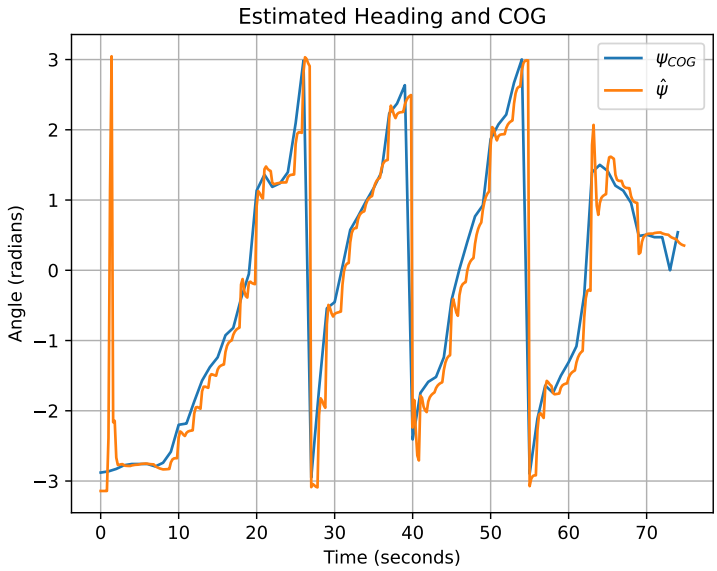


Figure 9.27: The estimated heading angle generated from the EKF described in Ch. 7 compared to the COG angle from the GPS receiver.

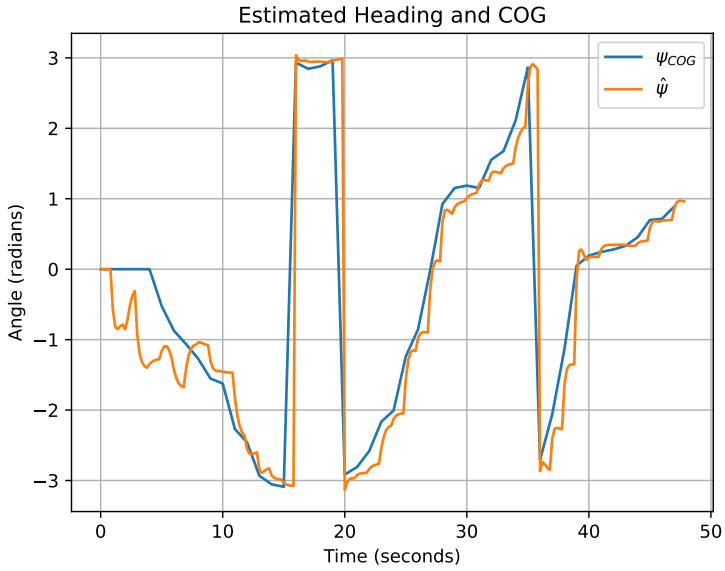


Figure 9.28: The estimated heading angle generated from the EKF described in Ch. 7 compared to the COG angle from the GPS receiver.

9.10 Conclusion and Caveats

This chapter highlighted the data and experimental results associated with the Slug 2 and Slug 3 ASVs. Ordinary Kriging and POK were compared on experimental depth measurements from three different data sets of the same body of water. The choice of using three separate data sets was meant to increase the confidence of the performance comparisons between Ordinary Kriging and POK. The depth measurements collected by the Slug 3 were also used to reconstruct the true depth field (see Fig. 9.6). GPR was chosen for interpolating the depth measurements between positions, because the hyper-parameters could be determined from the data empirically. Autonomous waypoint tracking was shown for the Slug 3 with multiple successful test runs. As stated before, the results of this section show a mean cross-track error for all autonomous test-runs that is less than 0.6 meters with a standard deviation less than 2.3 meters. The EKF discussed in Ch. 6 and Ch. 7 were used successfully for position estimation of both the Slug 2 and Slug 3. Additionally, the speed of the Slug 2 was estimated using the EKF. The course over ground angle was compared to the estimated heading angle of the vehicle. These results further validate the effectiveness of the EKF and the Nomoto ship steering model.

9.10.1 Connection to the Overarching Theme

POK was shown to be a fast method of estimating a field compared to Ordinary Kriging. The results show that POK scales better in terms of computation time versus the number of measurements for input. In terms of MAE, the results for three different data sets of the same body of water showed POK can outperform Ordinary Kriging. This shows that POK can be used onboard vehicles to help estimate a field during run-time. This could be very useful for autonomous vehicles

for other research domains or for ocean pollutant cleanup missions. For example POK could be used to quickly estimate a field, and help inform researchers of areas of interest. Similarly, it could be used as part of another algorithm for maximizing ocean cleanup; it could estimate high densities of ocean pollutants, such as plastic waste and help inform a fleet of ASVs where to clean up the most polluted areas efficiently during runtime.

Chapter 10

Conclusion

10.1 Discussion

This work culminated in the creation of two low-cost autonomous surface vehicles, capable of autonomous waypoint-to-waypoint tracking and intelligent exploration based on Gaussian Process Regression in the form of Ordinary Kriging and Partitioned Ordinary Kriging. An in-depth analysis and comparison was conducted to determine the most well-suited attitude heading and reference system algorithm. Simulations and experimental procedures were performed to observe AHRS computation times, error characteristics, and implementation variations. A number of different trajectory generation algorithms were introduced for exploration; different types of vehicle models were simulated and compared. A number of different controllers were designed, tuned and were implemented in code for real-world tests. An optimal method of exploration via uncertainty suppression subject to vehicle energy constraints was introduced.

All of this work represents a holistic approach to designing an efficient, low-cost autonomous surface vehicle. It represents a demonstration for an autonomous surface vehicle for exploration based on off-the-shelf-components that is readily

accessible and implementable. The performance analysis for online trajectory planning showed that even inexpensive computation platforms such as the PIC32 microcontroller and Raspberry Pi can be used to implement optimal exploration and control. Furthermore, a number of different vehicle models were discussed and compared. Considering vehicles that are not boats that can move in all three dimensions, the Newtonian model in Section 5.2 was shown to be able to describe vehicle motion in three axes. The Newtonian model in particular could even be used to describe the motion of a rocket, plane, or quad-copter, because it accounts for fluid medium drag forces and other disturbances.

10.1.1 Novel Contributions

The main high-level contributions and achieved goals of this work are as follows:

1. Autonomous exploration algorithms for static *and dynamic* path planning based on Ordinary Kriging, or Gaussian Process Regression.

2. Modernized architecture, software design, and sensor fusion for marine surface data collection.

3. Experimentally demonstrated waypoint line-tracing to sub-meter mean cross-track error with existing hardware; developed a simulation environment for SIL testing of the algorithms.

4. Designed, built, and tested a novel and affordable AHRS validation apparatus based on laser-cut MDF platform and a hall effect encoder for truth validation. The developed system is able to calibrate and assess performance of various types of attitude estimation algorithms at low-cost and without specialized external references.

5. Developed framework to compare performance and computational costs of various attitude estimation schemes including Extended Kalman Filter, TRIAD and Complementary Filter; generated trade-off plots of computation performance versus attitude accuracy.

6. Developed online trajectory generation and optimized sample point generation for maximal entropy suppression of a priori field. optimal sample point generation using Kriging.

7. Developed a novel speed-improved spatial estimation algorithm.

The software contributions of this work include:

1. Two novel hardware ASV implementations (low-level drivers and high-level code).

2. A software toolbox for autonomous exploration.

3. A full software stack for simulating and collecting real-time data from the experimental platform.

This work showed the comparison of different AHRS algorithms in simulation; contributions include the creation of a novel attitude and heading reference system validation apparatus. It was capable of comparing accumulated encoder angles about a body-fixed axis of rotation to the estimated orientation angle of a complementary filter. An embedded explicit, quaternion-based complementary filter was created and tested in the real world. AHRS performance was even compared to additional "ground truth" sources including the course-over-ground angle from a GPS module.

For optimal exploration, additional contributions includes the algorithms for forming a local directed a-cyclic graph for Bellman-Ford optimization. The di-

rected a-cyclic graph algorithm was shown to be versatile in terms of implementation; it could be adjusted to different field resolutions, and could be tuned to account for vehicle maneuvering limitations, such as minimum turning radius. A solution to the variance maximization problem was introduced by summing the negative of the variance along a path. This optimization was shown to be able to be performed during run-time of an autonomous surface vehicle. This includes successive optimizations during run-time and not just a single computation based on training data.

In terms of spatial estimation, a novel field partitioning scheme was introduced in Section 8.5. Results showed that this computation speedup by partitioning still maintained comparable mean absolute error for field estimates, and could (in certain situations) outperform Ordinary Kriging. Three path planners and four spatial estimators were compared with experimental data. This resulted in Fig. 9.17 and Table. 9.1 which showed that the best path planner and spatial estimator combination, after considering computation time, is HV-Bellman-Ford paired with POK.

10.2 Future Work

The next steps of this research could take many different paths. A few potential considerations are discussed here.

10.2.1 Variance and Distance Weighting

During this research, it was realized that the distance and variance may be differently weighted when applying the Bellman-Ford graph search algorithm. These weights could potentially change over time depending the current infor-

mation available. A potential next step could be to experiment with weighting the variance at nodes or along a path as well as the distance in relation to either the variogram, or some other criteria during exploration. For example Fig. 10.1 shows an example of weighting the distance 10× the variance.

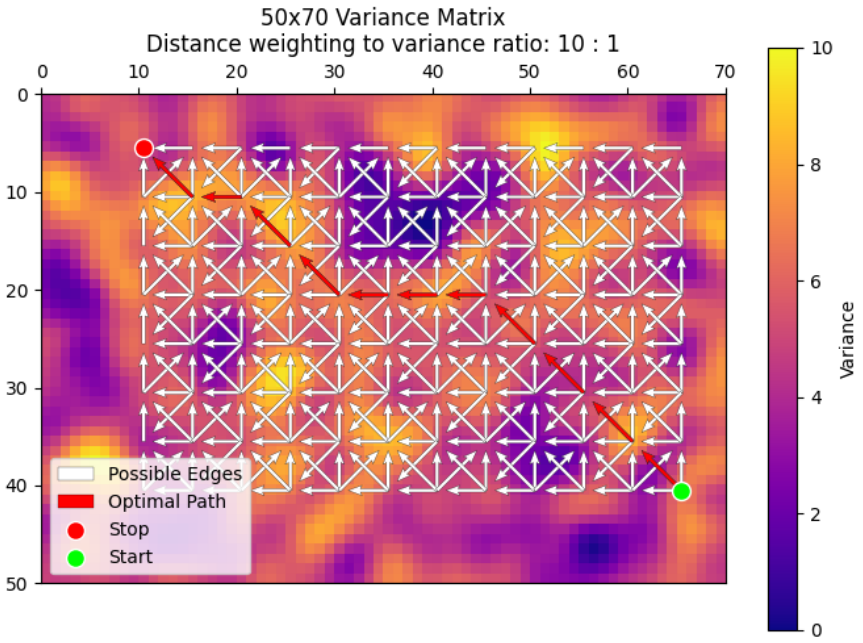


Figure 10.1: Example of a variance and distance path (red arrows) between a start point (green) and a stop point (red). The distance weighting to variance ratio is 10:1. The possible edges of the DAG are represented by the white arrows.

10.2.2 Multi-Disciplinary Optimization (MDO)

One potential next step for this research is to explore multi-disciplinary optimization for the optimal ASV design. Some potential variables to consider include: 1) the set of tasks for the vehicle, 2) the size, 3) the shape, 4) the battery size, 5) the number of motors, 6) the types of motors, 7) the propulsion force range 8) the computation platform (e.g.: a custom PCB that combines a Raspberry Pi

and microcontroller), 9) energy scavenging mechanisms, and 10) ease of manufacturing. All of these variables are quantifiable. If an optimal ASV design exists (one that is capable of performing the most oceanography or similar exploratory task for a fixed cost) the design should factor in all or most of these variables.

10.2.3 Aerospace Extension

As this research was conducted the author found many fundamental theoretical connections between ASV modeling and control design and designing and controlling aircraft and spacecraft. For example, a rocket and boat have much in common; they both are non-holonomic, have a form of thruster vectoring control, posses a fuel or energy constraints, move in a fluid medium with drag forces and disturbances, use embedded computation architectures (microcontrollers), use attitude and position estimation (Extended Kalman Filter), use radio telemetry, and require controllers for trajectory tracking. A potential next step of this research could be the design and control of a launch vehicle (e.g.: a rocket) to explore asteroids using GPR or OK to search for places to mine rare-Earth materials. Even the spatial estimation component of this research could be applied to a landing camera to locate and track the landing site for a rocket booster. In this example the field could be estimated from images taken by a video camera. GPR could be used to interpolate the images' pixels because the resolution at a distance would be a critical limiting factor in landing zone position estimation relative to the rocket lander-booster. It would be worth while to further test the HV-Bellman-Ford and POK combination in this context.

10.2.4 Field Estimate Normalization

An observation that was made after analyzing experimental depth data was that a sensitive aspect of Ordinary Kriging, that is not often discussed in the literature, is normalizing the field estimate matrix $\hat{\mathbf{Z}}$. This can be done based on known dimensions of the field *a priori*, however such information might not be available. It might be useful to devise an algorithm to aid in the field estimate normalization if there are no prior known maximum field attribute values. It was observed that Ordinary Kriging specifically could yield inaccurate field estimates if the normalization was based on poor assumptions of field attribute bounds.

10.2.5 Covariance Function Research

The covariance function and variogram models could be extended and further developed. The are many different types of covariance functions as noted in [74], but it is not clear if an adaptive covariance function exists, or if we are limited to case-specific functions *a priori*. The potential benefit could be to help form covariance matrices that are smaller than the number of measurements (and take less time to invert), but still accurately describe the spatial auto-correlation.

10.2.6 Covariance Kernel Optimization

Another method to reduce the computational load of spatial estimation, besides partitioning the field, is to modify the covariance kernel, as noted in [38]. This work did not explore different methods of kernel resizing, modification, augmentation, or optimization. Perhaps a dynamic covariance kernel could make for a rewarding next step in this research.

10.2.7 Combining Computational Loads

It was realized that there are similar code loops between creating a field estimate and generating a directed a-cyclic graph for an optimal graph search. It may be possible to combine the graph creating with the spatial estimation step, such that the nested matrix loops can fulfill two different purposes simultaneously. This could speed up online trajectory generation. Further research in this area is highly suggested as a potentially promising area.

Appendix A

Additional Simulation Results and Other Material

This appendix has GNC simulation results, variogram fitting graph comparisons, as well as AHRS validation apparatus CAD models, and other similar material.

A.0.1 Optimal Control

When possible, and depending on the system model an optimal controller should be considered, such as a linear quadratic regulator (LQR) or linear quadratic Gaussian (LQG). [33] outlines a steady state solution for an optimal controller. It is briefly and partially repeated here for completeness. If the model is non-linear and time-varying things become complicated. This is because the discrete Riccati equation,

$$\underline{\mathrm{S}}(k) = \underline{\Phi}^{\mathrm{T}}[\underline{\mathrm{S}}(k+1) - \underline{\mathrm{S}}(k+1)\underline{\Gamma}\underline{\mathrm{R}}^{-1}\underline{\Gamma}^{\mathrm{T}}\underline{\mathrm{S}}(k+1)]\underline{\Phi} + \underline{\mathrm{Q}}_1 \tag{A.1}$$

where $\underline{\mathrm{R}}$ may defined as

$$\mathrm{R} = \underline{\mathrm{Q}}_2 + \underline{\Gamma}^{\mathrm{T}}\underline{\mathrm{S}}(k+1)\underline{\Gamma} \tag{A.2}$$

may have to be solved as frequently as the smallest possible time step. Alternatively, it common to linearize the system about fixed points. This is due to the fact that the dynamics matrix $\underline{\Phi}$ will change as a function of time. However, in the case of the error model of the vehicle in the Serret-Frenet reference frame, the dynamics matrix is linear and time-invariant.

Assuming an infinite time horizon and seeking an optimal steady state gain vector, we can use the discrete state space formulation from Eq. (5.8) along with two weighting matrices $\underline{Q}_1$ and $\underline{Q}_2$ as part of the *Sweep* method by [11] (also shown in [33]). The weighting matrices are symmetric and non-negative definite. Given a vector of Lagrangian multipliers $\underline{\lambda}$, we assume there exists a matrix $\underline{\mathrm{S}}$ that translates the state vector $\underline{\mathrm{x}}$ to $\underline{\lambda}$, such that

$$\underline{\lambda}(k) = \underline{\mathrm{S}}(k)\underline{\mathrm{x}}(k) \tag{A.3}$$

This leads to the algebraic Riccati equation,

$$\underline{\mathrm{S}}_\infty = \underline{\Phi}[\underline{\mathrm{S}}_\infty - \underline{\mathrm{S}}_\infty\underline{\Gamma}\mathrm{R}^{-1}\underline{\Gamma}^{\mathrm{T}}\underline{\mathrm{S}}_\infty]\underline{\Phi} + \underline{\mathrm{Q}}_1 \tag{A.4}$$

While it is possible to find an analytical solution for equation (A.4), in most situations a numerical solution is necessary. One such method is eigenvector decomposition. This uses Hamilton's equations, also called the Euler-Lagrange equations,

$$\begin{bmatrix}\underline{\mathrm{x}} \\ \underline{\lambda}\end{bmatrix}_{k+1} = \begin{bmatrix}\underline{\Phi} + \underline{\Gamma}\underline{\mathrm{Q}}_2\underline{\Gamma}^{\mathrm{T}}\underline{\Phi}^{-\mathrm{T}}\underline{\mathrm{Q}}_1 & -\underline{\Gamma}\underline{\mathrm{Q}}_2\underline{\Gamma}^{\mathrm{T}}\underline{\Phi}^{-\mathrm{T}} \\ -\underline{\Phi}^{-\mathrm{T}}\underline{\mathrm{Q}}_1 & \underline{\Phi}^{-\mathrm{T}}\end{bmatrix}\begin{bmatrix}\underline{\mathrm{x}} \\ \underline{\lambda}\end{bmatrix}_k \tag{A.5}$$

The Hamiltonian matrix is

$$\underline{\mathcal{H}}_c = \begin{bmatrix} \underline{\Phi} + \underline{\Gamma}\underline{\mathrm{Q}}_2\underline{\Gamma}^{\mathrm{T}}\underline{\Phi}^{-\mathrm{T}}\underline{\mathrm{Q}}_1 & -\underline{\Gamma}\underline{\mathrm{Q}}_2\underline{\Gamma}^{\mathrm{T}}\underline{\Phi}^{-\mathrm{T}} \\ -\underline{\Phi}^{-\mathrm{T}}\underline{\mathrm{Q}}_1 & \underline{\Phi}^{-\mathrm{T}} \end{bmatrix} \tag{A.6}$$

At this time, we refer the reader to [33] for more information, and carry on with the general case in finding a constant optimal gain solution. We may now diagonalize the Hamiltonian matrix,

$$\underline{\mathcal{H}}_c^* = \begin{bmatrix} \underline{\mathrm{E}}^{-1} & \underline{0} \\ \underline{0} & \underline{\mathrm{E}} \end{bmatrix} \tag{A.7}$$

where $\underline{\mathrm{E}}$ and $\underline{\mathrm{E}}^{-1}$are a diagonal matrices possessing the unstable and stable roots respectively. The diagonalized Hamlitonian matix comes from the following similarity transform.

$$\underline{\mathcal{H}}_c^* = \underline{\mathrm{W}}^{-1}\underline{\mathcal{H}}_c\underline{\mathrm{W}} \tag{A.8}$$

$\underline{\mathrm{W}}$ is a matrix of eigenvectors of the Hamiltonian matrix, such that

$$\underline{\mathrm{W}} = \begin{bmatrix} \underline{\mathrm{X}}_I & \underline{\mathrm{X}}_O \\ \underline{\Lambda}_I & \underline{\Lambda}_O \end{bmatrix} \tag{A.9}$$

where the left column is the matrix of eigenvectors with corresponding roots that exist inside of the unit circle, and the right column with roots outside of the unit circle. Again, a more in-depth analysis of this method is done by [33]. The end result is

$$\underline{\mathrm{S}}_\infty = \underline{\Lambda}_I\underline{\mathrm{X}}_I^{-1} \tag{A.10}$$

And finally, the optimal gain matrix can be calculated and used for optimal full-

state feedback.

$$\underline{\mathrm{K}}_{\infty} = (\underline{\mathrm{Q}}_2 + \underline{\Gamma}^{\mathrm{T}}\underline{\mathrm{S}}_{\infty}\underline{\Gamma})^{-1}\underline{\Gamma}^{\mathrm{T}}\underline{\mathrm{S}}_{\infty}\underline{\Phi} \tag{A.11}$$

$$\underline{\mathrm{u}}_k = \underline{\mathrm{K}}_{\infty}(\underline{\mathrm{x}}_{\mathrm{des},k} - \underline{\mathrm{x}}_k) \tag{A.12}$$

A.1 B-Spline Generation

It can be useful to generate smooth paths for a vehicle to follow, rather than strict straight line segments. This can involve interpolating points of interest that the ASV is desired to pass over or next to. A popular method is to use splines, of which there many different kinds, including cubic, centripetal, Bezier, B-splines, rational and non-rational. This section discusses B-spline, as they have a number of useful properties, including guaranteed interpolation, smoothness conservation at segments or knots, and posses the convex hull property, such that their control points form a bounding perimeter around a B-spline sub-segment.

B-splines are calculated using a basis function, parameter vector, control points, and subdivisions of the parameter vector called knots. An important property of B-splines is that they may be used to do global interpolation of spatial coordinates. For example, suppose there exists a 2-dimensional plane and a matrix of x, y coordinates corresponding to points on the plane.

$$\mathbf{d} = \begin{bmatrix} x_0 & y_0 \\ x_1 & y_1 \\ x_2 & y_2 \\ \vdots & \vdots \\ x_n & y_n \end{bmatrix} \tag{A.13}$$

B-splines are defined using a basis function, as shown in [70]. The specific implementation used in this thesis and in [70] is repeated here for completeness. The basis function is

$$N_{i,0}(v) = \begin{cases} 1 & \text{if } u_i \leq v < u_{i+1} \\ 0 & \text{otherwise} \end{cases} \tag{A.14}$$

$$N_{i,p}(v) = \frac{v - u_i}{u_{i+p} - u_i} N_{i,p-1}(v) + \frac{u_{i+p+1} - v}{u_{i+p+1} - u_{i+1}} N_{i+1,p-1}(v) \tag{A.15}$$

where $v \in \mathbb{R} : [0, 1]$ and whose value corresponds to a point on the curve to be generated. The knots that subdivide the interval $[0, 1]$, are shown in eq A.14 and eq A.15 as u_i, u_{i+1}, etc. The basis coefficient, $N_{i,p}$ is calculated based on some iteration, i, and order, p. Using the basis coefficients, the following summation is used to calculate a point on the spline curve,

$$\mathbf{c}(v) = \sum_{i=0}^{n} N_{i,p}(v)\mathbf{p}_i \tag{A.16}$$

where $\mathbf{c}$ is a point on the curve, $\mathbf{p}_i$ is a control point. The control points may be found by solving the linear system of equations represented by

$$\mathbf{d}_k = \mathbf{c}(v_k) = \sum_{i=0}^{n} N_{i,p}(v_k)\mathbf{p}_i \tag{A.17}$$

where $\sum_{i=0}^{n} N_{i,p}(v_k)$ can be rewritten in matrix form as a basis coefficient matrix.

$$N = \begin{bmatrix} N_{0,p}(v_0) & N_{1,p}(v_0) & \dots & N_{n,p}(v_0) \\ N_{0,p}(v_1) & N_{1,p}(v_1) & \dots & N_{n,p}(v_1) \\ \vdots & \vdots & \ddots & \\ N_{0,p}(v_n) & N_{1,p}(v_n) & \dots & N_{n,p}(v_n) \end{bmatrix} \tag{A.18}$$

After rewriting eq A.18, the relation between the spatial coordinates and the control points can represented as $\mathbf{d} = N\mathbf{p}$. This can be used to solve for $\mathbf{p}$ by augmenting eq A.18 with the respective columns of the spatial coordinate matrix. The control points necessary to generate a spline that will pass through the desired spatial coordinates are found by using standard Gaussian elimination on the augmented coefficient matrix.

$$\begin{bmatrix} p_{0,i} \\ p_{1,i} \\ \vdots \\ p_{n,i} \end{bmatrix} = \left[\begin{array}{cccc|c} N_{0,p}(v_0) & N_{1,p}(v_0) & \dots & N_{n,p}(v_0) & d_{0,i} \\ N_{0,p}(v_1) & N_{1,p}(v_1) & \dots & N_{n,p}(v_1) & d_{1,i} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ N_{0,p}(v_n) & N_{1,p}(v_n) & \dots & N_{n,p}(v_n) & d_{n,i} \end{array}\right] \tag{A.19}$$

So long as the coordinates are distinct from each other the matrix is guaranteed to be non-singular. Note that each column of the control point matrix is calculated by solving the coefficient matrix augmented with each respective column of the spatial coordinates matrix. This means the number of times the linear system must be solved is equal to the dimension of the spline. In the 2-dimensional case, the system is solved twice. There exist other solving methods, such as QR factorization. Gaussian elimination is chosen as the means of solving the system because its implementation on an embedded system is more intuitive.

Recall that parameter generation is required to calculate the basis coefficients in eq A.18. There are various methods for selecting the b-spline parameters,

such as uniformly spaced, chord-length, or centripetal method. In this work the chord-length method is used for it's simplicity in implementation.

Let s_k be the ratio of the length between two spatial coordinates and the total length l_t of all segments between spatial coordinates.

$$s_k = \frac{1}{l_t} \sum_{i=1}^{k} ||d_i - d_{i-1}|| \tag{A.20}$$

Eq A.20, only applies to finding the parameters between 0 and 1. The first and last parameters are be defined as $s_0 = 0$ and $s_n = 1$ to complete the parameter vector.

The last component needed to form the b-splines is the knot vector $\mathbf{u}$. This is done using the following equation,

$$u_{j+p} = \frac{1}{p} \sum_{i=j}^{j+p+1} s_i \tag{A.21}$$

where $j \in \mathbb{Z} : [1, 2, ..., n - p]$. It should be noted that the number of knots $m = n + p + 1$. There are other methods for generating the knot vector, however some will result in a singular N when used with the chordal method. Fortunately, the method described above does not have this issue, as noted in [70].

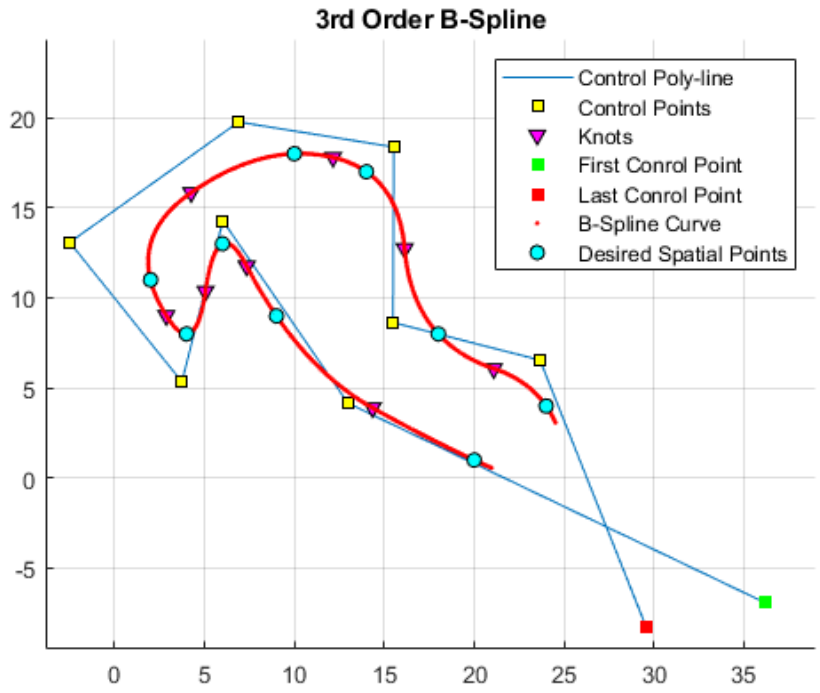


Figure A.1: A 2-dimensional, 3rd order B-spline generated on the PIC32MX795F512L.

The first and last p elements of the knot vector are 0's and 1's respectively. The reason for this lies in the multiplicity of the knots. In order to have a clamped B-spline where the first and last spatial coordinates connect to the curve, there must exists sufficient multiplicity. In practice, the above knot generation method will produce a spline that connects its ending spatial coordinates if an extra set of coordinates that are zero are added to the beginning and end of the spatial coordinates matrix. This effectively increases the multiplicity of the knot vector elements such that the b-spline connects end points. The curve passes through all spatial coordinates, excluding the necessary zero end points and the interpolates the desired points correctly. After applying the above methods, a 2-dimensional, 3rd order b-spline is created, as shown in fig A.1.

A.2 Fitting the Variogram with Least-Squares

Figs. A.2, A.3, A.4, and A.5 show the effects of the variations of the number of terms used for the Taylor expansion approximation for the Gaussian variogram model, as well as the effects of different tolerances, and lag sizes (or increments). The importance of these figures is highlighting the accuracy of fitting the variogram depending on the maximum lag element value h_{max} in the lag vector. This is essentially the furthest distance we may consider within the lag vector when applying least-squares to fit the variogram model. For instance, Fig. A.4 shows that for a complex gradient (having many "hills") the variogram fit is more accurate if h_{max} is less (11.m) than the maximum empirical variogram distance.

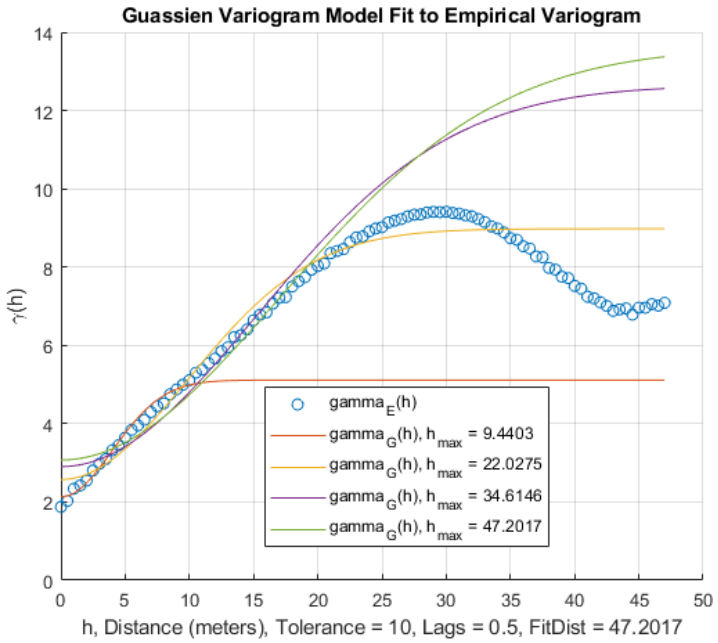


Figure A.2: An empirical variogram fit using least-squares with varying number of terms for the Taylor expansion

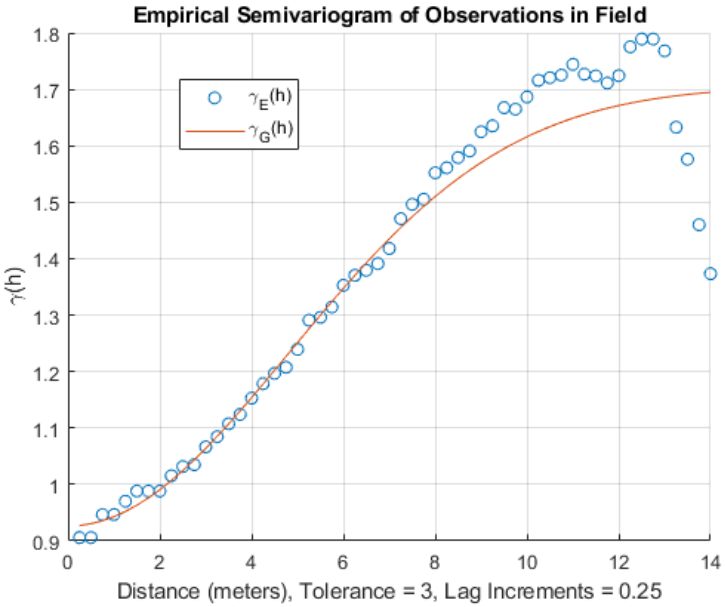


Figure A.3: An empirical variogram fit using least-squares with varying number of terms for the Taylor expansion

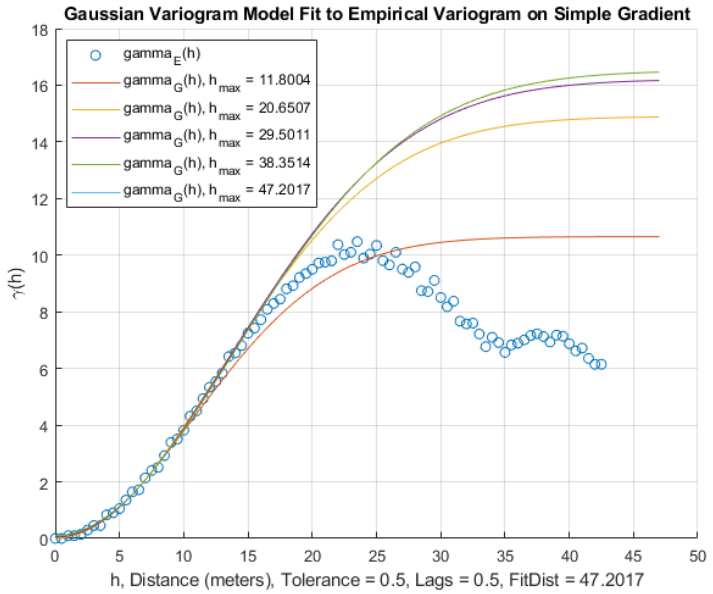


Figure A.4: An empirical variogram fit using least-squares with varying number of terms for the Taylor expansion

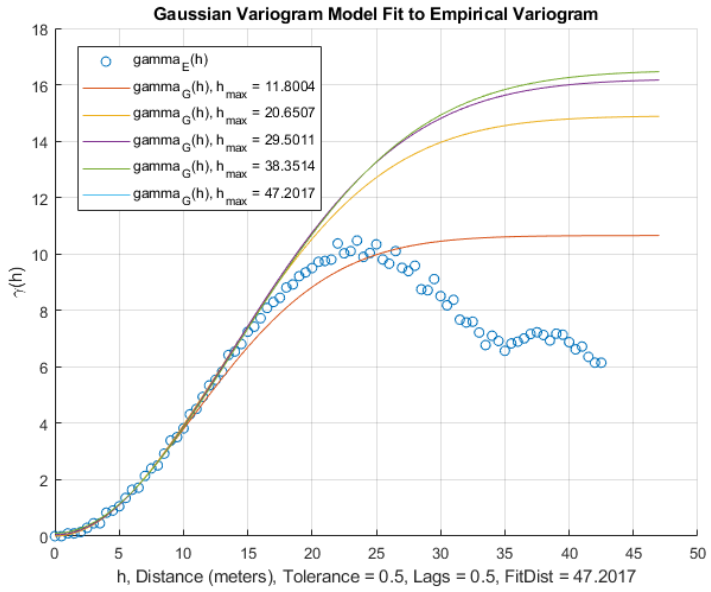


Figure A.5: An empirical variogram fit using least-squares with varying number of terms for the Taylor expansion

A.3 CAD

This appendix section has computer assisted designs (CAD) for things like the AHRS validation apparatus. Figs. A.6, A.7, and A.8 show the various CAD drawings. The CAD files can be found at `https://github.com/PavloGV/AHRS_Validation_Apparatus_CAD.git`.

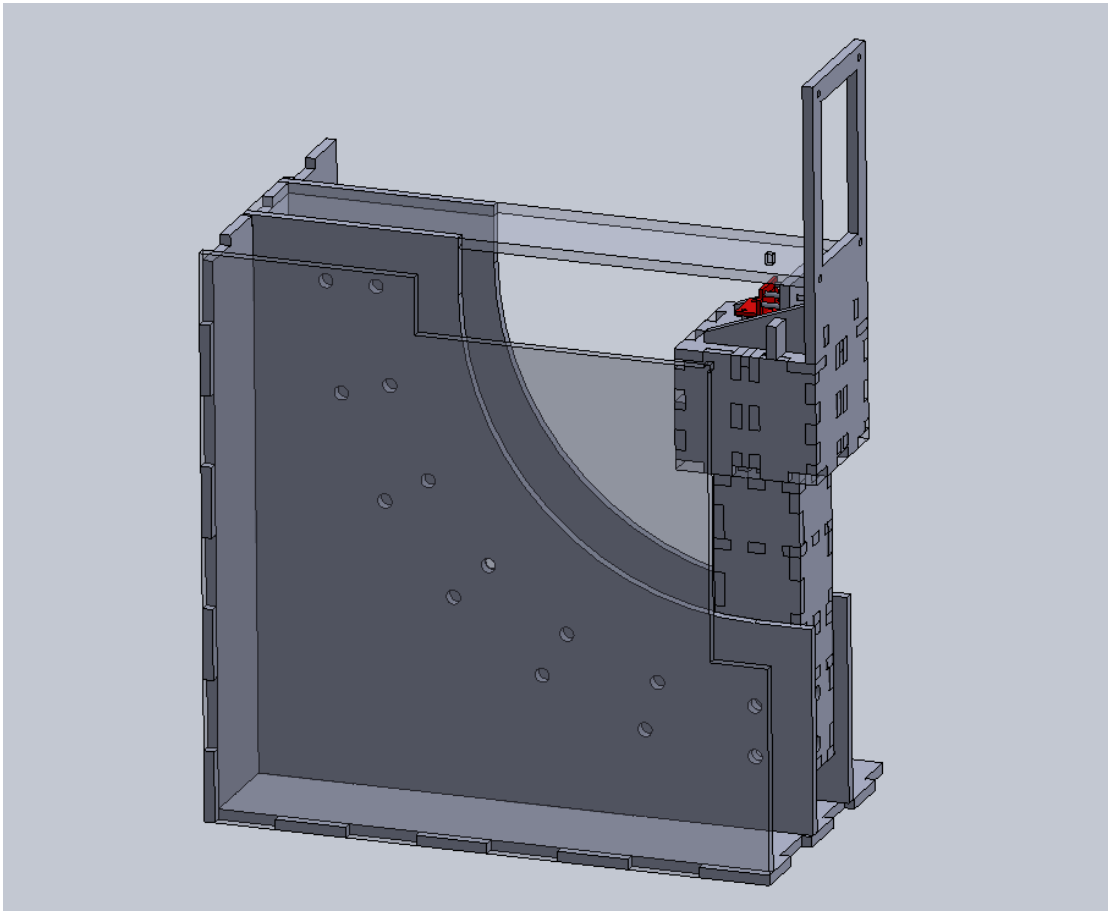


Figure A.6: An image of the 3D CAD model for the AHRS validation apparatus

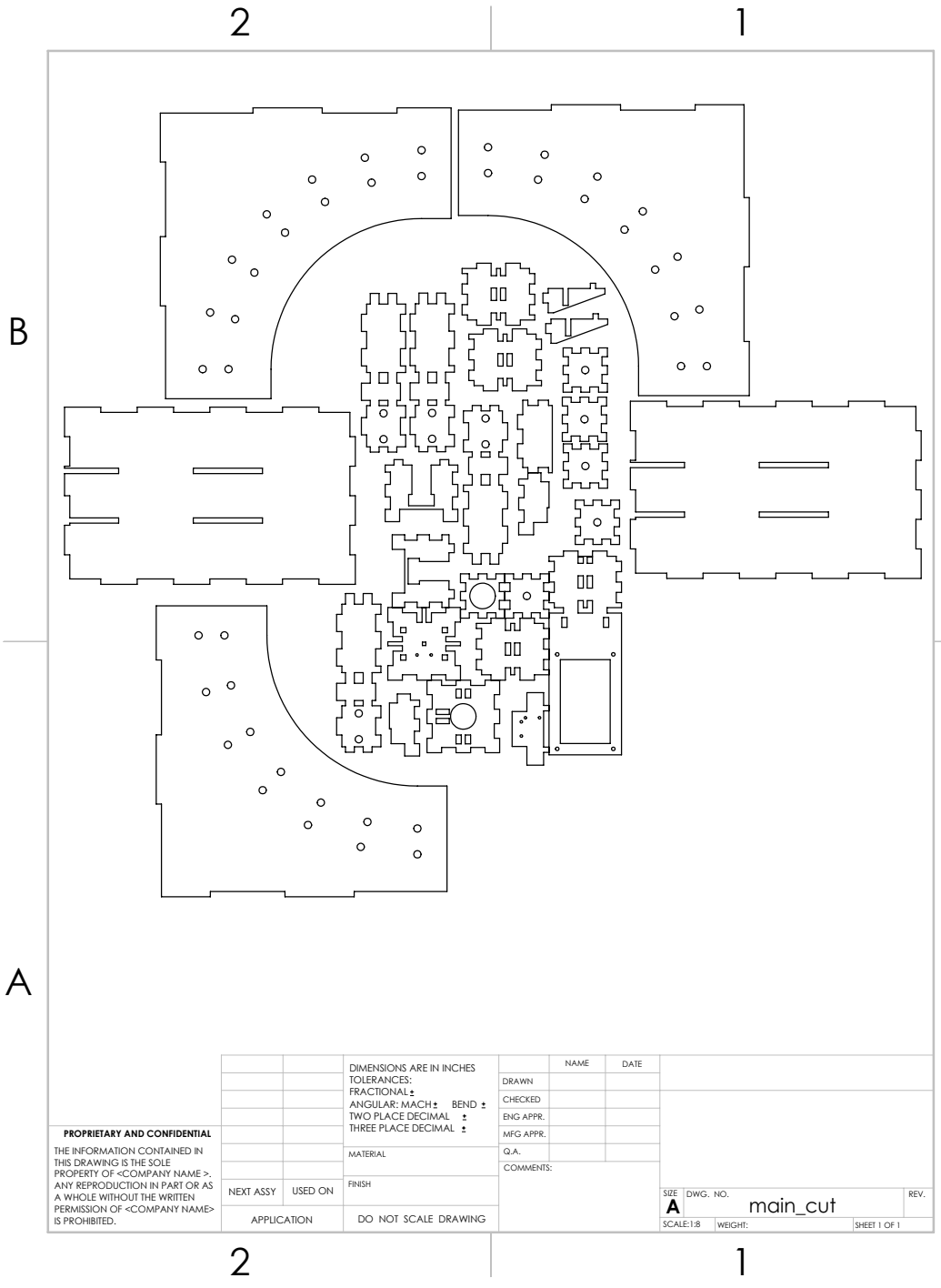


Figure A.7: A CAD drawing of the AHRS validation apparatus

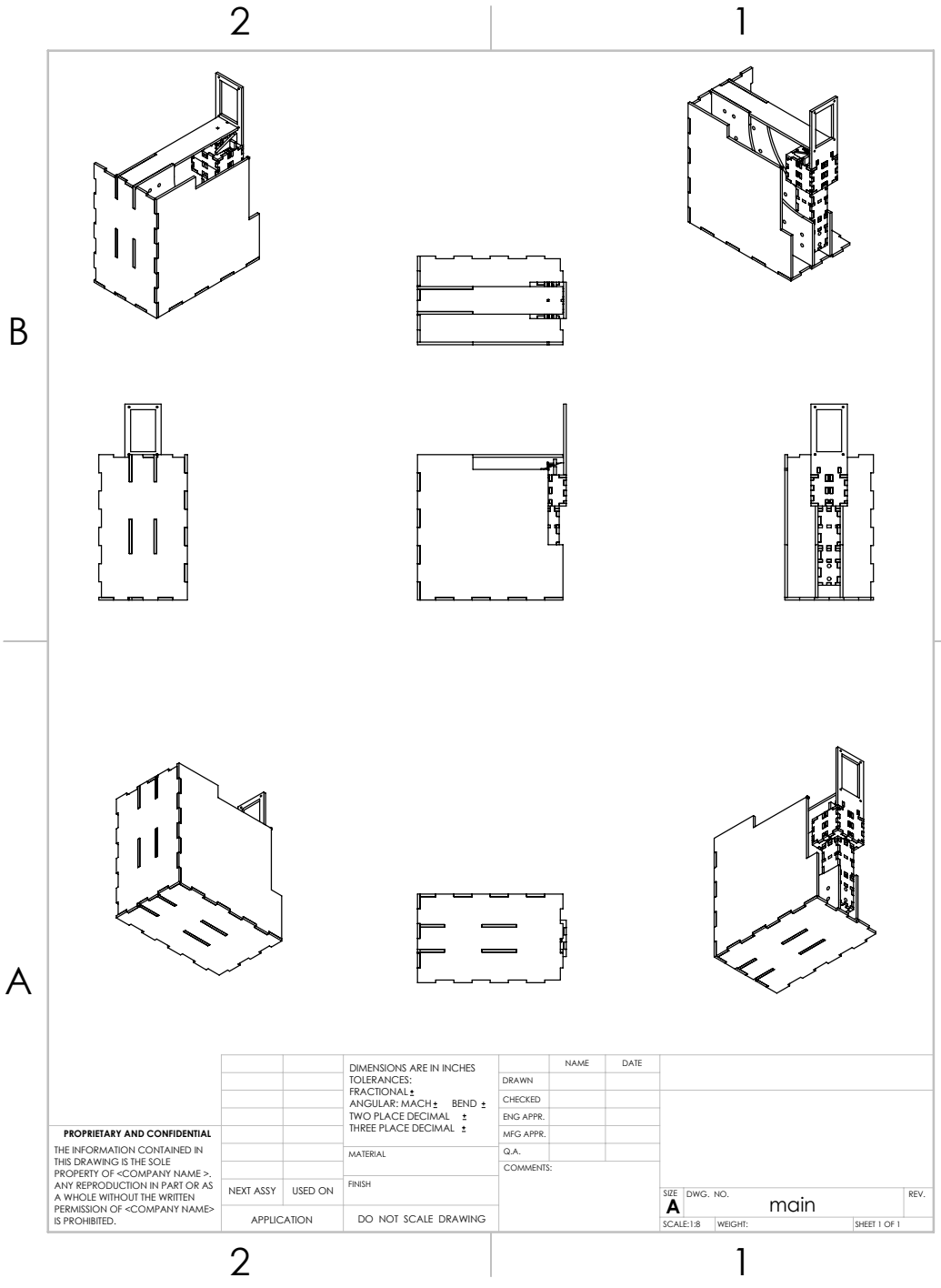


Figure A.8: A CAD drawing with multiple views of the AHRS validation apparatus

A.4 Hardware

This section has some additional photos of the hardware used by the Slug 2 and Slug 3 ASVs.

Figure A.9: A side shot of the Slug 3 ASV

Figure A.10: A front shot of the Slug 3 ASV

Figure A.11: A back shot of the Slug 3 ASV

Figure A.12: An image of the Turnigy Plush 40A ESC used by both the Slug 2 and Slug 3

Figure A.13: An image of the 3500kV Radient Reaktor brushless motor used on the Slug 2

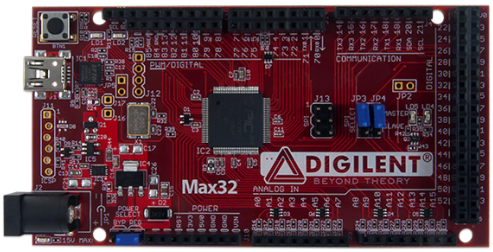


Figure A.14: An image of the Max32 microcontroller used on the Slug 2 and Slug 3

Figure A.15: An image of the Sik 925MHz 3DR radio module used to transmit telemetry from the Slug 2 to the ground control station (latptop).

Figure A.16: An image of the 5V SunFounder Metal Gear Digital RC Servo.

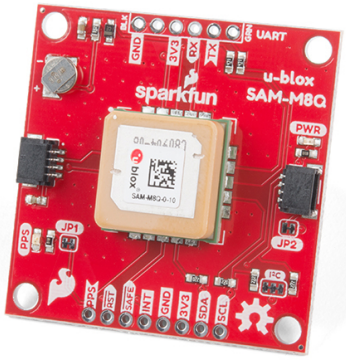


Figure A.17: An image of the Sparkfun breakout board for the SAM-M8Q GPS module used on the Slug 2 and Slug 3.

Bibliography

[1] Hyo-Sung Ahn and Seon-Ho Lee. Gyroless attitude estimation of sun-pointing satellites using magnetometers. *IEEE Geoscience and Remote Sensing Letters*, 2(1):8–12, January 2005. Conference Name: IEEE Geoscience and Remote Sensing Letters.

[2] Mohammad K. Al-Sharman, Yahya Zweiri, Mohammad Abdel Kareem Jaradat, Raghad Al-Husari, Dongming Gan, and Lakmal D. Seneviratne. Deep-Learning-Based Neural Network Training for State Estimation Enhancement: Application to Attitude Estimation. *IEEE Transactions on Instrumentation and Measurement*, 69(1):24–34, January 2020. Conference Name: IEEE Transactions on Instrumentation and Measurement.

[3] Livia Albeck-Ripka. The 'Great Pacific Garbage Patch' Is Ballooning, 87,000 Tons of Plastic and Counting. *The New York Times*, mar 2018.

[4] Carlos Almeida, Tiago Franco, Hugo Ferreira, Alfredo Martins, Ricardo Santos, Jose Miguel Almeida, Joao Carvalho, and Eduardo Silva. Radar based collision detection developments on USV ROAZ II. In *OCEANS 2009-EUROPE*, pages 1–6, May 2009.

[5] Michael S. Andrle and John L. Crassidis. Geometric Integration of Quaternions. *Journal of Guidance, Control, and Dynamics*, 36(6):1762–1767, 2013.

[6] Juli Berwald. Ocean-studying satellite 'no longer recoverable', 2011.

[7] James Bishop. Autonomous Observations of the Ocean Biological Carbon Pump. *Oceanography*, 22(2):182–193, June 2009.

[8] R. Bohlin and L.E. Kavraki. Path planning using lazy PRM. In *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*, volume 1, pages 521–528 vol.1, April 2000. ISSN: 1050-4729.

[9] S.A. Bortoff. Path planning for UAVs. In *Proceedings of the 2000 American Control Conference. ACC (IEEE Cat. No.00CH36334)*, volume 1, pages 364–368 vol.1, June 2000. Issue: 6 ISSN: 0743-1619.

[10] Andrew W. Browning. A mathematical model to simulate small boat behaviour. *SIMULATION*, 56(5):329–336, May 1991.

[11] Arthur Bryson, Y.-C Ho, and George Siouris. Applied Optimal Control: Optimization, Estimation, and Control. *Systems, Man and Cybernetics, IEEE Transactions on*, 9:366–367, July 1979.

[12] Joseph Carsten, Arturo Rankin, Dave Ferguson, and Anthony Stentz. Global Path Planning on Board the Mars Exploration Rovers. In *2007 IEEE Aerospace Conference*, pages 1–11, March 2007. ISSN: 1095-323X.

[13] Tongtong Chen, Bin Dai, Ruili Wang, and Daxue Liu. Gaussian-Process-Based Real-Time Ground Segmentation for Autonomous Land Vehicles. *Journal of Intelligent & Robotic Systems*, 76(3-4):563–582, December 2014.

[14] XiaoDiao Chen, Yin Zho, Zhenyu Shu, and Hua Su. Improved Algebraic Algorithm on Point projection for B´eziercurves. In *Second International Multi-Symposiums on Computer and Computational Sciences (IMSCCS 2007)*, pages 158–163, Iowa City, IA, USA, August 2007. IEEE.

[15] Ji-wung Choi, Renwick Curry, and Gabriel Elkaim. Path planning based on bézier curve for autonomous ground vehicles. In *Advances in Electrical and Electronics Engineering - IAENG Special Edition of the World Congress on Engineering and Computer Science 2008*, pages 158–166. IEEE.

[16] Ji-wung Choi, Renwick E Curry, and Gabriel Hugh Elkaim. Continuous Curvature Path Generation Based on B´ezier Curves for Autonomous Vehicles. page 12, 2010.

[17] Sophie Jeong CNN, Susanna Capelouto and Nicole Chavez. A cargo ship leaking tons of oil off the Mauritius coast has split in two.

[18] John Connors and Gabriel Elkaim. Analysis of a Spline Based, Obstacle Avoiding Path Planning Algorithm. In *2007 IEEE 65th Vehicular Technology Conference - VTC2007-Spring*, pages 2565–2569, Dublin, Ireland, April 2007. IEEE. ISSN: 1550-2252.

[19] Noel Cressie. Fitting variogram models by weighted least squares. *Journal of the International Association for Mathematical Geology*, 17(5):563–586, July 1985.

[20] Renwick Curry, Mariano Lizarraga, and Gabriel Elkaim. The Design of Rapidly Reconfigurable Filters for Attitude and Position Determination. In *AIAA Infotech@Aerospace 2010*. American Institute of Aeronautics and Astronautics, 2010.

[21] Hector Garcia de Marina, Fernando J. Pereda, Jose M. Giron-Sierra, and Felipe Espinosa. UAV Attitude Estimation Using Unscented Kalman Filter and TRIAD. *IEEE Transactions on Industrial Electronics*, 59(11):4465–4474, November 2012. Conference Name: IEEE Transactions on Industrial Electronics.

[22] Dmitri Dolgov, Sebastian Thrun, Michael Montemerlo, and James Diebel. Path Planning for Autonomous Vehicles in Unknown Semi-structured Environments. *The International Journal of Robotics Research*, 29(5):485–501, April 2010. Publisher: SAGE Publications Ltd STM.

[23] Scott C. Doney, Victoria J. Fabry, Richard A. Feely, and Joan A. Kleypas. Ocean Acidification: The Other CO $_2$ Problem. *Annual Review of Marine Science*, 1(1):169–192, January 2009.

[24] Eric Dorveaux, David Vissière, Alain-Pierre Martin, and Nicolas Petit. Iterative calibration method for inertial and magnetic sensors. In *Proceedings of the 48h IEEE Conference on Decision and Control (CDC) held jointly with 2009 28th Chinese Control Conference*, pages 8296–8303, December 2009. ISSN: 0191-2216.

[25] Matthew Dunbabin, Alistair Grinham, and James Udy. An Autonomous Surface Vehicle for Water Quality Monitoring. page 6, 2009.

[26] Mark Euston, Paul Coote, Robert Mahony, Jonghyuk Kim, and Tarek Hamel. A complementary filter for attitude estimation of a fixed-wing UAV. In *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 340–345, September 2008. ISSN: 2153-0858, 2153-0866.

[27] P. G. Falkowski. Biogeochemical Controls and Feedbacks on Ocean Primary Production. *Science*, 281(5374):200–206, July 1998.

[28] Yaron A. Felus, Alan Saalfeld, and Burkhard Schaffrin. Delaunay Triangulation Structured Kriging for Surface Interpolation. *Surveying and Land Information Science*, 65(1):27–36, mar 2005.

[29] Hugo Ferreira, C. Almeida, A. Martins, J. Almeida, N. Dias, A. Dias, and E. Silva. Autonomous bathymetry for risk assessment with ROAZ robotic surface vehicle. In *OCEANS 2009-EUROPE*, pages 1–6, May 2009.

[30] Paolo Fiorini and Zvi Shiller. Motion Planning in Dynamic Environments Using Velocity Obstacles. *The International Journal of Robotics Research*, 17(7):760–772, July 1998.

[31] Gregory Foltz. An Unprecedented View Inside a Hurricane, May 2022.

[32] Trygve Olav Fossum, Cédric Travelletti, Jo Eidsvik, David Ginsbourger, and Kanna Rajan. Learning excursion sets of vector-valued Gaussian random fields for autonomous ocean sampling. *The Annals of Applied Statistics*, 15(2):597–618, June 2021. Publisher: Institute of Mathematical Statistics.

[33] Gene F. Franklin. *Digital Control of Dynamic Systems 3rd Edition*. 1998.

[34] Gene F. Franklin, J. David Powell, and Abbas Emami-Naeini. *Feedback Control of Dynamic Systems (7th Edition)*. Pearson, 2014.

[35] Bryan A. Franz, Sean W. Bailey, P. Jeremy Werdell, and Charles R. McClain. Sensor-independent approach to the vicarious calibration of satellite ocean color radiometry. *Applied Optics*, 46(22):5068–5082, August 2007.

[36] Arthur Gelb. *Applied Optimal Estimation*. The MIT Press, 1974.

[37] C. L. Gentemann, Joel P. Scott, Piero L. F. Mazzini, Cassia Pianca, Santha Akella, Peter J. Minnett, Peter Cornillon, Baylor Fox-Kemper, Ivona Cetinić, T. Mike Chin, Jose Gomez-Valdes, Jorge Vazquez-Cuervo, Vardis Tsontos, Lisan Yu, Richard Jenkins, Sebastien De Halleux, Dave Peacock, and Nora Cohen. Saildrone: Adaptively Sampling the Marine Environment. *Bulletin of the American Meteorological Society*, 101(6):E744–E762, June 2020. Publisher: American Meteorological Society Section: Bulletin of the American Meteorological Society.

[38] Arjan Gijsberts and Giorgio Metta. Real-time model learning using Incremental Sparse Spectrum Gaussian Process Regression. *Neural Networks*, 41:59–69, May 2013.

[39] William W. Hager. Updating the Inverse of a Matrix | SIAM Review | Vol. 31, No. 2 | Society for Industrial and Applied Mathematics.

[40] J. T. Hardy. The sea surface microlayer: Biology, chemistry and anthropogenic enrichment. 11(4):307–328.

[41] He He and Wan-Chi Siu. Single image super-resolution using Gaussian process regression. In *CVPR 2011*, pages 449–456, June 2011. ISSN: 1063-6919.

[42] Roger Hine, Scott Willcox, Graham Hine, and Tim Richardson. The wave glider: A wave-powered autonomous marine vehicle. In *OCEANS 2009*, pages 1–6. ISSN: 0197-7385.

[43] Kevin Judd and Timothy McLain. Spline based path planning for unmanned air vehicles. In *AIAA Guidance, Navigation, and Control Conference and Exhibit*. American Institute of Aeronautics and Astronautics.

[44] Dongwon Jung and Panagiotis Tsiotras. On-Line Path Generation for Unmanned Aerial Vehicles Using B-Spline Path Templates. *Journal of Guidance Control and Dynamics*, 36, November 2013.

[45] R. E. Kalman. A New Approach to Linear Filtering and Prediction Problems. *Journal of Basic Engineering*, 82(1):35–45, March 1960.

[46] JACK B. KUIPERS. *Quaternions and Rotation Sequences: A Primer with Applications to Orbits, Aerospace and Virtual Reality*. Princeton University Press, 1999.

[47] Yu-lei Liao, Ming-jun Zhang, and Lei Wan. Serret-Frenet frame based on path following control for underactuated unmanned surface vehicles with dynamic uncertainties. *Journal of Central South University*, 22(1):214–223, January 2015.

[48] Yang Liu, Jingfa Li, Shuyu Sun, and Bo Yu. Advances in Gaussian random field generation: a review. *Computational Geosciences*, 23(5):1011–1047, October 2019.

[49] Lennart Ljung. *System Identification Theory for the User*. Prentice Hall PTR, One Lake Street, Upper Saddle River, NJ 07458, second edition, 1999.

[50] R. Mahony, T. Hamel, and J. Pflimlin. Complementary filter design on the special orthogonal group SO(3). In *Proceedings of the 44th IEEE Conference on Decision and Control*, pages 1477–1484, December 2005.

[51] Bryant Mairs and Gabriel Elkaim. SeaSlug: A low-cost, long-duration mobile marine sensor platform for flexible data-collection deployments. page 7.

[52] Justin Manley and Scott Willcox. The Wave Glider: A persistent platform for ocean science. In *OCEANS'10 IEEE SYDNEY*, pages 1–5, May 2010.

[53] J.L. Marins, Xiaoping Yun, E.R. Bachmann, R.B. McGhee, and M.J. Zyda. An extended Kalman filter for quaternion-based orientation estimation using MARG sensors. In *Proceedings 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems. Expanding the Societal Role of Robotics in the the Next Millennium (Cat. No.01CH37180)*, volume 4, pages 2003–2011 vol.4, October 2001.

[54] Claire Martin. Vanishing Marine Algae Can Be Monitored From a Boat With Your Smartphone | Science | Smithsonian Magazine.

[55] Kevin P. Murphy. *Machine learning: a probabilistic perspective*. Adaptive computation and machine learning series. MIT Press, Cambridge, MA, 2012.

[56] Daiju Narita, Katrin Rehdanz, and Richard S. J. Tol. Economic costs of ocean acidification: a look into the impacts on global shellfish production. *Climatic Change*, 113(3):1049–1063, August 2012.

[57] NASA. Electrical Power, January.

[58] Masahiro Ono, Thoams J. Fuchs, Amanda Steffy, Mark Maimone, and Jeng Yen. Risk-aware planetary rover operation: Autonomous terrain classification and path planning. In *2015 IEEE Aerospace Conference*, pages 1–10, March 2015. ISSN: 1095-323X.

[59] John E. O'Reilly, Stéphane Maritorena, B. Greg Mitchell, David A. Siegel, Kendall L. Carder, Sara A. Garver, Mati Kahru, and Charles McClain. Ocean color chlorophyll algorithms for SeaWiFS. *Journal of Geophysical Research: Oceans*, 103(C11):24937–24953, October 1998.

[60] Chiwoo Park and Daniel Apley. Patchwork Kriging for Large-scale Gaussian Process Regression. page 43.

[61] Jonghoon Park and Wan-Kyun Chung. Geometric integration on Euclidean group with application to articulated multibody systems. *IEEE Transactions on Robotics*, 21(5):850–863, October 2005.

[62] Tristan Perez. *Mathematical Ship Modeling for Control Applications*. 2002.

[63] Sean Potter. NASA Ingenuity Mars Helicopter Prepares for First Flight, March 2021.

[64] S.H. Pourtakdoust and H. Ghanbarpour Asl. An adaptive unscented Kalman filter for quaternion-based orientation estimation in low-cost AHRS. *Aircraft Engineering and Aerospace Technology*, 79(5):485–493, September 2007.

[65] I. Rhee, S. Park, and C. Ryoo. A tight path following algorithm of an UAS based on PID control. In *Proceedings of SICE Annual Conference 2010*, pages 1270–1273, 2010.

[66] Melissa Rice and Briony Horgan. NASA Perseverance Rover's First Major Successes on Mars – An Update From Mission Scientists, October 2021. Section: Space.

[67] A. M. Sabatini. Quaternion-based strap-down integration method for applications of inertial sensing to gait analysis. *Medical & Biological Engineering & Computing*, 43(1):94–101, February 2005.

[68] Angelo Maria Sabatini. Kalman-Filter-Based Orientation Determination Using Inertial/Magnetic Sensors: Observability Analysis and Performance Evaluation. *Sensors*, 11(10):9182–9206, October 2011. Number: 10 Publisher: Molecular Diversity Preservation International.

[69] Kevin G. Sellner, Gregory J. Doucette, and Gary J. Kirkpatrick. Harmful algal blooms: causes, impacts and detection. 30(7):383–406.

[70] C.-K. Shene. B-spline curves: Computing the coefficients, cs3621 introduction to computing with geometry notes, 2014.

[71] D. Shi, Y. Xu, B. M. Hopkinson, and F. M. M. Morel. Effect of Ocean Acidification on Iron Availability to Marine Phytoplankton. *Science*, 327(5966):676–679, February 2010.

[72] M. D. SHUSTER and S. D. OH. Three-axis attitude determination from vector observations. *Journal of Guidance and Control*, 4(1):70–77, 1981. Publisher: American Institute of Aeronautics and Astronautics _eprint: https://doi.org/10.2514/3.19717.

[73] P. Smith and M. Dunbabin. High-fidelity autonomous surface vehicle simulator for the maritime RobotX challenge. pages 1–10.

[74] Sisi Song. *Trajectory Planning for Autonomous Vehicles for Optimal Exploration of Spatial Processes*. PhD thesis, UC Santa Cruz, 2019.

[75] Dave Steitz. Terra: Flagship of the Earth Observation System.

[76] Ching-Yaw Tzeng and Ju-Fen Chen. FUNDAMENTAL PROPERTIES OF LINEAR SHIP STEERING DYNAMIC MODELS. *Journal of Marine Science and Technology*, 7(2):10, 1999.

[77] National Oceanic and Atmospheric Administration US Department of Commerce. Can we clean up, stop, or end harmful algal blooms?

[78] Jur van den Berg, Ming Lin, and Dinesh Manocha. Reciprocal Velocity Obstacles for real-time multi-agent navigation. In *2008 IEEE International Conference on Robotics and Automation*, pages 1928–1935, May 2008. ISSN: 1050-4729.

[79] Pavlo Vlastos. Low-Cost Validation for Complmentary Filter-Based ARHS. *Proceedings of IEEE/ION PLANS 2020*, September 2020.

[80] Pavlo Vlastos. Partitioned Gaussian Process Regression for Online Trajectory Planning for Autonomous Vehicles. *International Conference on Controls, Automation, and Systems*, October 2021.

[81] George G. Waldbusser, Burke Hales, Chris J. Langdon, Brian A. Haley, Paul Schrader, Elizabeth L. Brunner, Matthew W. Gray, Cale A. Miller, and Iria Gimenez. Saturation-state sensitivity of marine bivalve larvae to ocean acidification. *Nature Climate Change*, 5(3):273–280, March 2015.

[82] S. Willcox, C. Meinig, C. L. Sabine, N. Lawrence-Slavas, T. Richardson, R. Hine, and J. Manley. An autonomous mobile platform for underway surface carbon measurements in open-ocean and coastal waters. In *OCEANS 2009*, pages 1–8.

[83] Sargis Yonan, Renwick Curry, and Gabriel Elkaim. Uncertainty suppression methods for the exploration of sparsely sampled fields. pages 2480–2511, 10 2019.

[84] F. Zhao and B. G. M. van Wachem. A novel Quaternion integration approach for describing the behaviour of non-spherical particles. *Acta Mechanica*, 224(12):3091–3109, December 2013.

[85] Xu Zhong, Allison Kealy, and Matt Duckham. Stream Kriging: Incremental and recursive ordinary Kriging over spatiotemporal data streams. *Computers & Geosciences*, 90:134–143, may 2016.