UNIVERSITY OF CALIFORNIA, SAN DIEGO

The Development and Implementation of a Low-Cost, Ball-Catching Robot

A Thesis submitted in partial satisfaction of the
requirements for the degree
Master of Science

in

Engineering Sciences (Mechanical Engineering)

by

Kaylee Feigum

Committee in charge:

Thomas Bewley, Chair
Mark Anderson
Mauricio de Oliveira

2017

The Thesis of Kaylee Feigum is approved, and it is acceptable in quality and form for publication on microfilm and electronically:

_____

_____

_____

Chair

University of California, San Diego

2017

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I would like to acknowledge my advisor, Dr. Bewley, for the project idea and guidance. I would also like to acknowledge my lab-mates in the Coordinated Controls Lab for their advice and suggestions.

ABSTRACT OF THE THESIS

The Development and Implementation of a Low-Cost, Ball-Catching Robot

by

Kaylee Feigum

Master of Science in Engineering Sciences (Mechanical Engineering)

University of California, San Diego, 2017

Thomas Bewley, Chair

As the number of robotic toys on the market increases, so does the difficulty of differentiating a product from the competition. One approach to this problem is to develop robots that can interact with the user in novel ways. To this end, a relatively small, ball-catching robot was developed with a focus on low-cost components. The two main features that keep the cost low are a mobile inverted pendulum robot design, and low-cost cameras. The focus of this paper is overcoming the challenges that these two features add the the ball-catching task. Linear regression was used for estimating and predicting the ball trajectory using low-cost cameras. Three strategies for intercepting the ball with a non-holonomic

robot were compared. The best strategy for the application used nonlinear model predictive control to calculate paths for the robot. A prototype was built and it successfully caught a thrown ball, but only within a limited area. Finally, improvements are suggested to increase the ball catching area of the robot, and to improve catching accuracy.

Chapter 1

Introduction

With the increasing number of robotic toys on the market, to remain competitive a designer must come up with new ways of engaging the consumer, which may include developing new ways for the robot to interact with the user. One interaction that has not been explored much in the commercial market is the game of catch, although many different ball catching systems have been developed for non-commercial purposes. Often, the systems with the most impressive results are those which use an industrial arm and cameras that are located off of the robot such as seen in [1] which is able to catch a ball thrown by a human and bounce it on a paddle repeatedly. Another example can be seen in [2] which is able to catch softballs thrown by many different people in a net on the end of the arm with about a %66 catch rate. In [3] a basketball was caught on a plate at the end of the arm. While these results are compelling, having the cameras off of the robot adds a large amount of complexity in processing and sending results to the robot. In [4] and [5], an industrial arm is used with only a monocular camera in the hand of the robot. Both of these examples actuate the arm to keep the ball in view of the camera.

While industrial arms allow complex strategies to be designed and tested, they are

extremely expensive and are very far from the consumer toy level. A mobile robot is presented in [6], but this one intercepts balls that are on the ground and not in the air. The project that is most similar to the goal of this paper is [7] in which a monocular vision system on a mobile robot is used to catch balls. This paper does not have low-cost as a goal, although it was most likely cheaper than an industrial arm. While mobile prototypes are cheaper than the industrial arms, there are not as many of this type, and most development seems focused on the more expensive arms. The high cost of these current prototypes might explain the lack of a commercially available ball-catching robotic toy. To get closer to a commercially viable catch-playing robot, a ball-catching prototype was developed, with a focus on keeping costs low.

Designing a low-cost prototype presents extra challenges not present in a more expensive build. Most low-cost cameras have a frame rate that is too slow to capture the fast motion of a thrown ball. They also generally have significant image distortion, making it difficult to locate the ball precisely. To keep costs low, one must also use a low-cost computer or micro-controller to control the robot and process the data, limiting the computations that can be done in real time.

In this paper, the physical prototype is described, as well as the equations of motion governing the system. A method for calculating the ball location between camera frames is detailed. Then, a strategy for determining an optimal ball-interception path is presented and compared to two different interception strategies. Finally, the conclusions drawn from the experiment are presented, as well as suggestions for improvement upon the design.

Chapter 2

Robotic Platform

2.1 Introduction

For the prototype catching robot, a mobile inverted pendulum (MIP) design was chosen for the robot. A MIP generally consists of a body and two wheels. The robot is inherently unstable, and must be actively stabilized by the wheels, which both control the angle of the body and drive the robot. The MIP design eliminates the need for an actuated catching arm, reducing complexity and cost; the arm is rigidly attached to the body allowing it to pivot with the rest of the body. This will allow the robot to change the arm angle to catch the ball as well allowing future iterations to throw the ball back to the user [8], creating a more convincing game of catch.

2.2 Hardware

The body of the robot was based off of a previous design developed in the Coordinated Robotics Lab called iFling [8]. All parts besides electronics, wheel treads, and motor

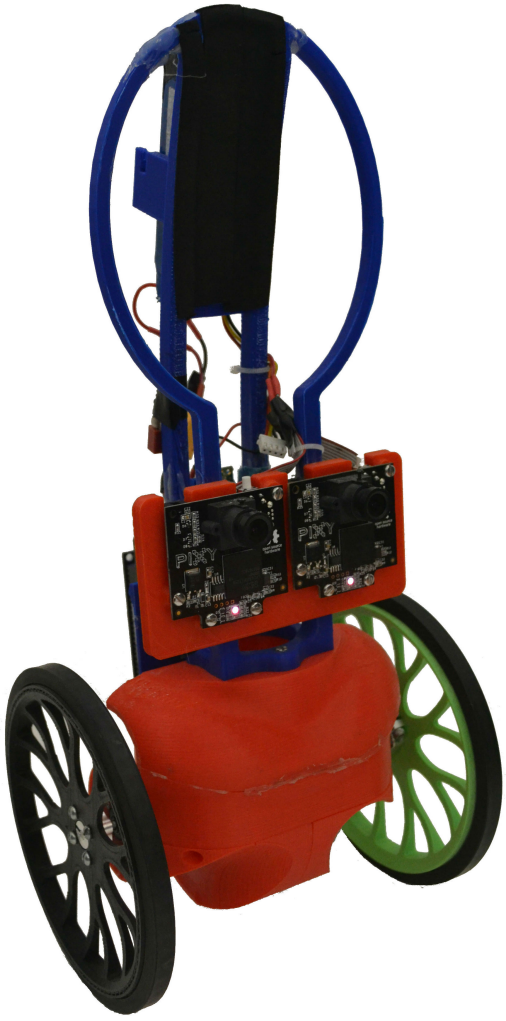


Figure 2.1: The completed protoype

hubs are 3D printed with polylactic acid (PLA) plastic. The lower body is modified from the previous iFling project, but is updated to hold new motors and a new controller. The arm on top of the robot has been redesigned to optimize for catching instead of throwing: a cage has been added to trap the ball in the lower arm section, as well as to provide a larger catching area in the top section. A mount for two small cameras has also been added to the front of the robot. The robot is controlled by a BeagleBone Black (BBB) with a Robotics Cape produced by Strawson Design. The motors are ServoCity 624 RPM Premium Planetary

Gear Motors. The physical parameters of the robot used for modeling the dynamics can be found in Table 2.1, and the motor properties are located in Table 2.2.

Table 2.1: Physical parameters of robot platform

Property	Variable Name	Value
Mass of body	m_b	$0.9kg$
Mass of wheel	m_w	$67g$
Wheel radius	r	$73mm$
Length to center of mass from motor axis	L_b	$16mm$
Length between wheels	L_w	$74mm$
Moment of inertia of body about the x axis	I_1	$3g*m^2$
Moment of inertia of body about the y axis	I_2	$6g*m^2$
Moment of inertia of body about the z axis	I_2	$6g*m^2$
Moment of inertia of wheel about the x axis	I_{w_1}	$8.93e-5kg*m^2$
Moment of inertia of wheel about the y axis	I_{w_2}	$1.79e-4kg*m^2$

Table 2.2: Motor parameters

Property	Variable	Value
Nominal Voltage	V	$12V$
Motor Resistance	Ω	3.4Ω
Torque Constant	k	$0.014N*m/A$
Motor Inertia	J	4.6e-5 $kg*m^2$
Gear Ratio	Γ	$19:1$
Stall Torque	s	$2.9kg*m$
No Load Current	i_0	$0.19A$

2.3 Modeling

To develop a control system and path planner for this platform, a model of the system must first be developed. Lagrange's equation is used to do this

$$\frac{d}{dt}(\frac{\partial \boldsymbol{L}}{\partial \dot{q}_i}) - \frac{\partial \boldsymbol{L}}{\partial q_i} = Q_i, \qquad i = 1,2,...,n \tag{2.1}$$

where $\boldsymbol{L} = K - U$, U is the potential energy of the system, K is the kinetic energy of the system, Q_i are the external forces applied to the system, and q_i are the generalized coordinates:

$$\boldsymbol{q} = \begin{bmatrix} \theta(t) \\ \phi_1(t) \\ \phi_2(t) \end{bmatrix} \tag{2.2}$$

where ϕ_1 and ϕ_2 are the angle of the left and right wheel about the y-axis measured from the z-axis and θ is the angle of the body about the y-axis as measured from the z-axis. The energy of the system can be separated into the energy contribution of the body and the energy contribution of each wheel. The coordinate system used for this model can be seen in Figure 2.2.

The kinetic energy of each component can be calculated from its linear and angular velocity. The linear velocity of the body is

$$v_b = \begin{bmatrix} v_{b_1} \\ v_{b_2} \\ -L_b \sin(\theta(t))\theta'(t) \end{bmatrix} \tag{2.3}$$

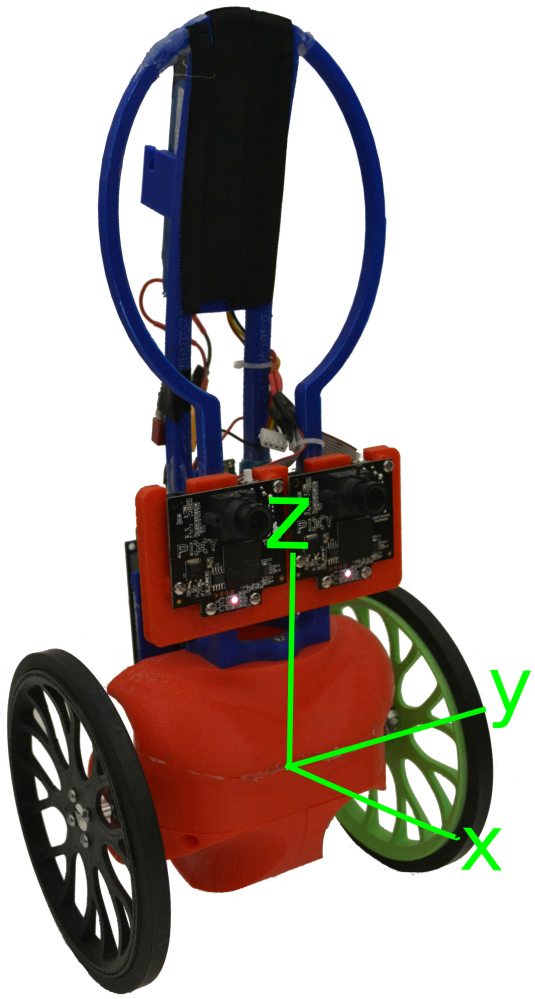


Figure 2.2: The coordinate system used to model the dynamics of the robot

$$v_{b_1} = \cos\left(\frac{r(\phi_1(t)-\phi_2(t))}{L_w}\right)\left(\frac{1}{2}R(\phi_1'(t)+\phi_2'(t)) + L_b\cos(\theta(t))\theta'(t)\right) - \frac{L_bR\sin(\theta(t))\sin\left(\frac{r(\phi_1(t)-\phi_2(t))}{L_w}\right)(\phi_1'(t)-\phi_2'(t))}{L_w} \quad (2.4)$$

$$v_{b_2} = sin\left(\frac{r(\phi_1(t)-\phi_2(t))}{L_w}\right)\left(\frac{1}{2}R(\phi_1'(t)+\phi_2'(t)) + L_b\cos(\theta(t))\theta'(t)\right) - \frac{L_bR\sin(\theta(t))\cos\left(\frac{r(\phi_1(t)-\phi_2(t))}{L_w}\right)(\phi_1'(t)-\phi_2'(t))}{L_w} \quad (2.5)$$

The angular velocity of the body is

$$\omega_b = \begin{bmatrix} -\sin(\frac{r(\phi_1(t)-\phi_2(t))}{L_w})\theta'(t) \\ -\cos(\frac{r(\phi_1(t)-\phi_2(t))}{L_w})\theta'(t) \\ \frac{r(\phi_1'(t)-\phi_2'(t))}{L_w} \end{bmatrix} \tag{2.6}$$

The kinetic energy from each wheel is

$$v_{w_1} = \begin{bmatrix} r\cos(\frac{r(\phi_1(t)-\phi_2(t))}{L_w})\phi_1'(t) \\ r\sin(\frac{r(\phi_1(t)-\phi_2(t))}{L_w})\phi_1'(t) \\ 0 \end{bmatrix} \tag{2.7}$$

$$v_{w_2} = \begin{bmatrix} r\cos(\frac{r(\phi_1(t)-\phi_2(t))}{L_w})\phi_2'(t) \\ r\sin(\frac{r(\phi_1(t)-\phi_2(t))}{L_w})\phi_2'(t) \\ 0 \end{bmatrix} \tag{2.8}$$

and the angular velocity of each wheel is

$$\omega_{w_1} = \begin{bmatrix} -\sin(\frac{r(\phi_1(t)-\phi_2(t))}{L_w})\phi_1'(t) \\ \cos(\frac{r(\phi_1(t)-\phi_2(t))}{L_w})\phi_1'(t) \\ \frac{r(\phi_1'(t)-\phi_2'(t))}{L_w} \end{bmatrix} \tag{2.9}$$

$$\omega_{w_2} = \begin{bmatrix} -\sin(\frac{r(\phi_1(t)-\phi_2(t))}{L_w})\phi_2'(t) \\ \cos(\frac{r(\phi_1(t)-\phi_2(t))}{L_w})\phi_2'(t) \\ \frac{r(\phi_1'(t)-\phi_2'(t))}{L_w} \end{bmatrix} \tag{2.10}$$

The potential energy of each wheel is zero, and the potential energy of the body is,

$$U_b = \begin{bmatrix} 0 \\ 0 \\ gL_b m_b \cos(\theta(t)) \end{bmatrix} \tag{2.11}$$

where g is the acceleration due to gravity. The external forces, Q, for this system are the torque from each motor.

$$Q = \begin{bmatrix} -\tau_1 - \tau_2 \\ \tau_1 \\ \tau_2 \end{bmatrix} \tag{2.12}$$

$$\tau_1 = \frac{V}{\Omega k} u_1 + \frac{1}{\Omega k^2}(\theta'(t) - \phi_1'(t)) \tag{2.13}$$

$$\tau_2 = \frac{V}{\Omega k} u_2 + \frac{1}{\Omega k^2}(\theta'(t) - \phi_2'(t)) \tag{2.14}$$

Using these equations and the equation for kinetic energy, $K = \frac{1}{2}\sum_{i=1}^{n} m_i v_i^2$ where m_i is the mass of each energy component and v_i is the velocity of each energy component, the Lagrangian L can be calculated, and is located in appendix A as A.1. The Lagrangian is next used in equation 2.1 to calculate the equations of motion, which are then rearranged to fit the form,

$$E\frac{d\boldsymbol{X}}{dt} = N(\boldsymbol{X}, \boldsymbol{u}) \tag{2.15}$$

$\boldsymbol{X}$ is the state vector and has been chosen to be,

$$\boldsymbol{X} = \begin{bmatrix} \theta(t) \\ \phi_1(t) \\ \phi_2(t) \\ \theta'(t) \\ \phi_1'(t) \\ \phi_2'(t) \\ x(t) \\ y(t) \\ \gamma(t) \end{bmatrix} \tag{2.16}$$

$x(t)$ and $y(t)$ are the robot's position along the x- and y-axis respectively. $\gamma(t)$ is the robot's heading as measured from the x-axis. The equations that define these variables are as follows

$$x'(t) = \frac{R}{2}\cos(\gamma(t))(\phi_1'(t) + \phi_2'(t)) \tag{2.17}$$

$$y'(t) = \frac{R}{2}\sin(\gamma(t))(\phi_1'(t) + \phi_2'(t)) \tag{2.18}$$

$$\gamma'(t) = \frac{R}{2L_w}(\phi_1'(t) - \phi_2'(t)) \tag{2.19}$$

N and E are given in Appendix A as Equations A.17 and A.6 respectively.

Chapter 3

Ball Trajectory Measurement and Prediction

3.1 Introduction

To measure the position of the ball relative to the robot while the ball is in the air, a pair of cameras is used. The desire to minimize the price of the prototype drove the camera selection, but using a cheaper camera brought many extra challenges not present when using more expensive cameras. The cameras needed to be calibrated to reduce distortion, and the lack of a controllable shutter on the cameras meant that the measurements from the two cameras are not synchronized. To get around this problem, a method was devised for estimating measurements between camera frames.

Because the cameras are rigidly attached to the robot body, when the robot changes pose, the scene the cameras are able to capture also changes. While it could be possible to adjust the position and angle of the robot to keep the ball in frame, that is not possible in this application because the angle of the body of the robot also controls the speed of the robot,

and the speed needs to be the maximum possible to catch the ball before it hits the ground. Therefore, the location of the ball must be predicted using only the initial measurements of the cameras before the robot needs to start moving. This prediction process is done by using the equations of motion for the ball to calculate its future trajectory.

3.2 Cameras

3.2.1 Camera Selection

The choice of cameras was limited by three factors: the size of the robot, the cost of the cameras, and the speed of the ball. As the robot is designed to be low-cost it is necessarily small, meaning that the cameras must also be similarly small, which eliminates many popular off-the-shelf vision solutions for robots. The goal of reducing costs also excludes small high-end cameras specifically designed for computer vision. Because the ball quickly flies through the air when it is thrown and the robot must intercept it before it lands, the camera must capture multiple frames and send them back to the computer well before the ball hits the ground. A low frame rate will also cause significant blur in the image of the ball if a typical rolling shutter is used in the camera. This makes it impossible to determine the exact location in the ball in the frame, because its image will be blurred over a large area of its motion.

The camera selected to fit these three criteria is the Pixy (CMUcam5) created by Carnegie Mellon Robotics Institute and Charmed Labs [9]. The camera board is about $51mm$ tall and $50mm$ long, allowing it to easily fit on the robot. The camera operates at a maximum of 50 frames per second. It costs around $70, making it much cheaper than other cameras that have a similar frame rate. Because it can send data over serial interfaces, the timing of

the frames can be accurately measured, and two cameras can be used to get a measurement in 3 dimensions. The Pixy camera differs from typical cameras used in computer vision tasks. The camera does not send an image back to the computer, instead the unit has an on-board processor that performs the necessary calculations and only sends the location and size of the desired object back to the computer. This capability eliminates the need to use computer vision algorithms on the computer controlling the robot, significantly reducing the amount of processing power that the computer requires, which allows a cheaper and smaller board to be used to control the robot.

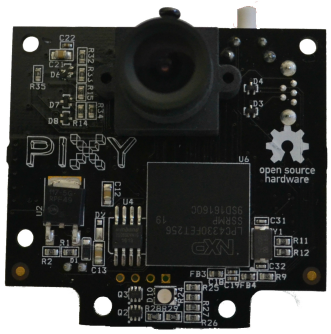


Figure 3.1: One of the Pixy cameras used in the prototype

For this prototype, two Pixys were mounted to the front of the robot spaced 70*mm* apart using a 3D printed mount. The two cameras communicate with the BeagleBone Black using the I^2C interface. To use the I^2C interface, the address of each camera was set using the PixyMon software. The color of the ball to be caught was selected and adjusted using the same software. When moving to new lighting situations, the color parameters often need to be readjusted before the ball will be correctly recognized. An example of the typical results from the Pixy in mediocre lighting conditions can be seen in Figure 3.2. As can be

seen, the entire ball is not recognized as the lighting quality causes dark shadows on the bottom side of the ball which cause the color to no longer register as red. The ball is reliably distinguished from the background and surrounding objects even in less than ideal lighting conditions, so it is deemed adequate for the goals of this prototype.

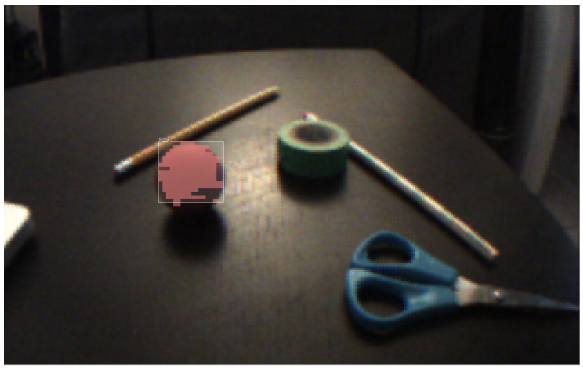


Figure 3.2: An example picture taken with the Pixy using the PixyMon software showing the detection of the red ball

The Pixy camera has a resolution of 320x200 pixels and a 75° view angle. The ball being used for the experiments is 40mm is diameter. If the smallest area to be detected is 8 pixels across, this result leads to a range of 1.0m from the front of the camera to detect the ball in ideal light conditions. In less than perfect lighting conditions, often part of the ball is not detected, leading to a effective range of about 0.8m. With the cameras mounted 70mm apart, the closest that an object can be detected by both cameras is 0.05m. The area over which the view of the cameras overlaps can be seen in Figure 3.3. By placing the cameras close together, most of the viewing area of each camera is visible to the other camera as well.

3.2.2 Calibrating the Cameras

As with many low-cost cameras, the Pixy exhibits significant radial and tangential distortion in the image that it captures. Radial distortion causes straight lines of images to

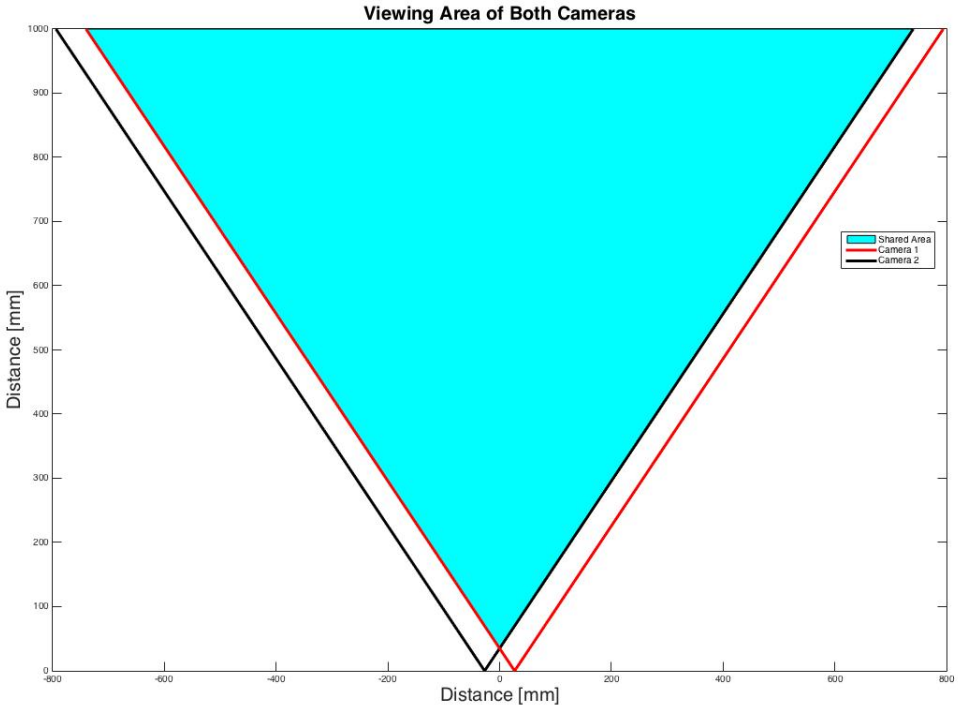


Figure 3.3: The area that is viewable by each camera

bow outward because the light rays bend more near the edges of the lens than the center. This distortion is described by the following equation [10],

$$\begin{bmatrix} \tilde{u}_{distorted} \\ \tilde{v}_{distorted} \end{bmatrix} = \begin{bmatrix} \tilde{u}(k_1 r^2 + k_2 r^4 + k_3 r^6) \\ \tilde{v}(k_1 r^2 + k_2 r^4 + k_3 r^6) \end{bmatrix} \tag{3.1}$$

where $r = \tilde{u}^2 + \tilde{v}^2$, $(\tilde{u}, \tilde{v})$ are the coordinates of the undistorted image in normalized image coordinates, $(\tilde{u}_{distorted}, \tilde{v}_{distorted})$ are the coordinates of the distorted image in normalized image coordinates, k_1, k_2, and k_3 are the radial distortion coefficients. The normalized image

coordinates are related to the image coordinates in pixels by the equation

$$\begin{bmatrix} \tilde{u} \\ \tilde{v} \end{bmatrix} = \begin{bmatrix} \frac{u-u_0}{f_u} \\ \frac{v-v_0}{f_v} \end{bmatrix} \tag{3.2}$$

where (u,v) are the coordinates of the undistorted image in pixels, (u_0,v_0) are the coordinates of the optical center in pixels, f_u and f_v are the focal lengths in pixels in the u and v directions respectively. The coordinate system of an image taken from a Pixy can be seen in Figure 3.4

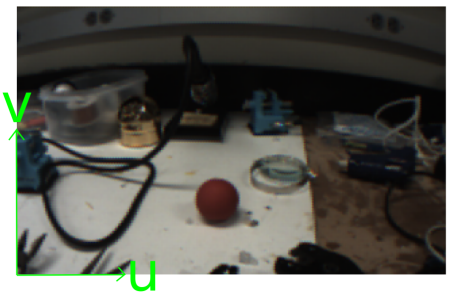


Figure 3.4: The coordinate system of a picture taken with a Pixy camera

Tangential distortion is caused when the camera sensor and lens are not parallel, and is represented by the following equation [10],

$$\begin{bmatrix} \tilde{u}_{distorted} \\ \tilde{v}_{distorted} \end{bmatrix} = \begin{bmatrix} \tilde{u} + 2p_1\tilde{u}\tilde{v} + p_2(r^2 + 2\tilde{u}^2) \\ \tilde{v} + p_1(r^2 + 2\tilde{v}^2) + p_2\tilde{u}\tilde{v} \end{bmatrix} \tag{3.3}$$

where $p_1 and p_2$ are the tangential distortion coefficients.

If the distortion is not accounted for, the measurement of the ball location will be inaccurate. To obtain the distortion parameters, OpenCV's calibration routine was used for both cameras [11]. This process returned the distortion coefficients as well as the intrinsic parameters of the cameras in the form

$$d = \begin{bmatrix} k_1 \\ k_2 \\ p_1 \\ p_2 \\ k_3 \end{bmatrix} \tag{3.4}$$

$$C = \begin{bmatrix} f_u & 0 & u_0 \\ 0 & f_v & v_0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.5}$$

These values are included in Appendix A as A.2-A.5. Using these parameters, an undistorted image can be produced as seen in Figure 3.5. The correction fixes the curved lines near the edges of the image, producing a more accurate image.

(a) The original image (b) The undistorted image

Figure 3.5: A distorted image and the same image after correcting for distortion

Generally, to produce an undistorted image, one first chooses a pixel location in the desired undistorted image, then using equations 3.1-3.3, one calculates which pixel from the distorted image corresponds to the new pixel. In this way, the undistorted image is gradually built up pixel by pixel. Because the Pixy only returns one pixel location, to determine its undistorted location, one would have to select a group of pixels where the

undistorted pixel might end up, then test each pixel in the group until the desired one is located. While intelligent group selection would reduce the number of calculations needed for this approach, this process is time consuming, and to guarantee a match a relatively large group would have to be selected. Instead, for this prototype it makes more sense to use a root-finding algorithm to find the undistorted location using equations 3.1-3.3. The Newton-Raphson method was used, which can be represented as

$$\boldsymbol{U}_{n+1} = \boldsymbol{U}_n - J^{-1}(\boldsymbol{U}_n)F(\boldsymbol{U}_n) \tag{3.6}$$

where $\boldsymbol{U}$ is the image location $(\tilde{u}, \tilde{v})$, $F(\boldsymbol{U}_n)$ is the equation

$$\begin{bmatrix} \tilde{u}_{distorted} \\ \tilde{v}_{distorted} \end{bmatrix} = \begin{bmatrix} \tilde{u} + 2p_1\tilde{u}\tilde{v} + p_2(r^2 + 2\tilde{u}^2) + \tilde{u}(k_1r^2 + k_2r^4 + k_3r^6) \\ \tilde{v} + p_1(r^2 + 2\tilde{v}^2) + p_2\tilde{u}\tilde{v} + \tilde{v}(k_1r^2 + k_2r^4 + k_3r^6) \end{bmatrix} \tag{3.7}$$

and $J(\boldsymbol{U}_n)$ is the Jacobian matrix of $F(\boldsymbol{U}_n)$. Equation 3.6 was used to find the undistorted pixel location by iterating until convergence. By selecting the distorted location as the initial guess for the solution, convergence generally happens in 2 or 3 steps because the undistorted location is never very far from the distorted location.

3.3 Ball Position Using Stereo Vision

After the calibrated location of the ball in the image of each camera is obtained, the measurements from the two cameras must be combined to get the ball's location in 3 dimensional space. Usually to accomplish this, two cameras are placed in the same plane and they take an image at the same instant. Then, the depth of the object from the camera

can be measured using the equation

$$x = \frac{f_u b}{u_2 - u_1} \tag{3.8}$$

where b is the distance between the two cameras, u_1 and u_2 are the undistorted pixel locations of each camera, and x is the distance in the x-direction using the coordinate system shown in figure 2.2. The following equations can then be used to obtain measurements in the y and z direction.

$$y = \frac{-u_1 x}{f_u} \tag{3.9}$$

$$z = \frac{v_1 x}{f_v} \tag{3.10}$$

Unfortunately, this procedure cannot be used with the Pixy cameras because they do not include a way to control the operation of the shutter. This means that the two cameras cannot be guaranteed to capture an image at the same instant. Instead, the two cameras operate independently, with a timing difference up to 10ms. To use this data for measuring depth, the ball location must be estimated between frames. To do this, a linear regression model is be used.

3.3.1 Linear Regression to Calculate Ball Position Between Frames

When the trajectory of the ball is captured by the cameras, the three-dimensional parabolic trajectory of the ball projected onto the camera image planes results in two parabolic paths. When only the u and v components of the trajectory are plotted versus capture time, the result is a parabola in the v coordinates and a roughly straight line in the u coordinates as seen in Figures 3.6 and 3.7. If the cameras are physically located on the same plane at the same height, the measured v coordinates of a fixed object should be the

same for the two cameras. Therefore, nothing needs to be done to estimate the position of the ball along the v axis between each camera frame, as the position from the other camera can be used. The position of the ball along the u axis is not the same between cameras, as is expected, and the u position must be estimated between each camera frame to get a correct depth estimate.

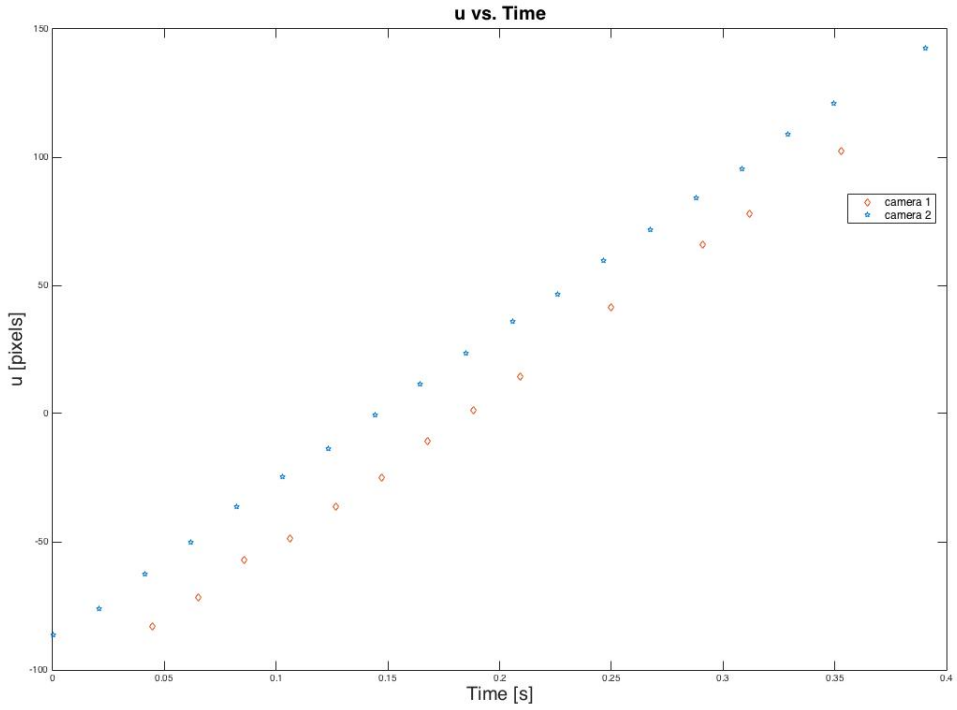


Figure 3.6: The u coordinates of each camera plotted against time for a sample throw

Because the plot of u position versus time is approximately linear, a simple linear regression model can be used. The equations to estimate the new u location in pixels u_{est} are

$$u_{est} = mt + b \tag{3.11}$$

$$m = \frac{\sum_{i=1}^{n}(t_i - \bar{t})(u_i - \bar{u})}{\sum_{i=1}^{n}(t_i - \bar{t})^2} \tag{3.12}$$

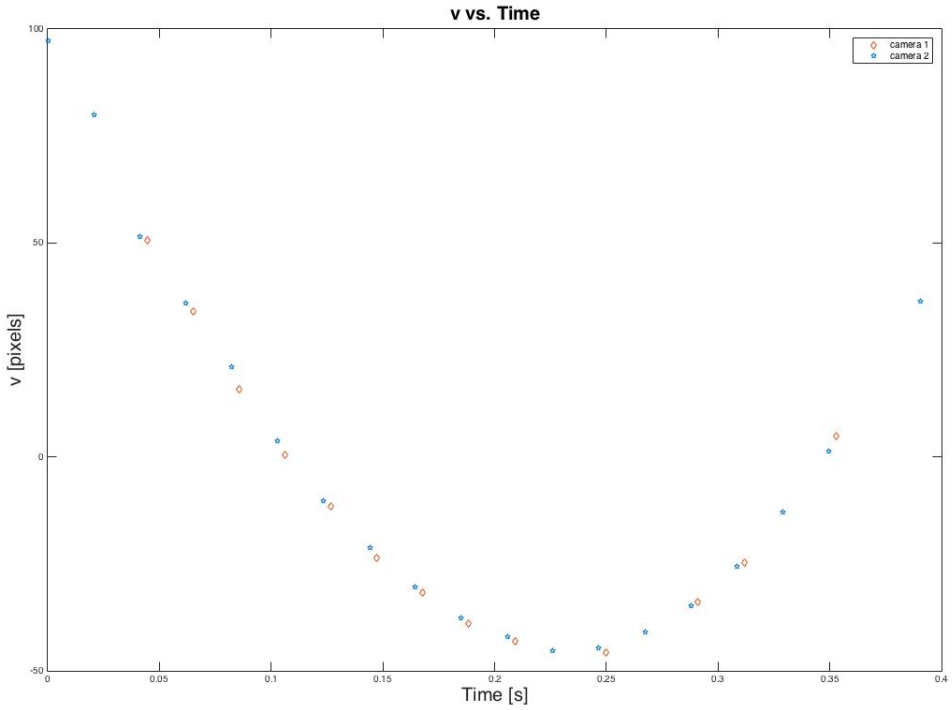


Figure 3.7: The v coordinates of each camera plotted against time for a sample throw

$$b = \bar{u} - m\bar{t} \tag{3.13}$$

where t is the time of the measurement, $\bar{u}$ and $\bar{t}$ are the average of u and t respectively. The lines resulting from using these equations on one set of data are shown in figure 3.8 For this set of data, both lines had a correlation coefficient of 0.999, showing that the equations can be used to accurately estimate the location of the ball in the u direction between frames for each camera.

Using the estimated u location values u_{est}, equations 3.8-3.10 can be used to reconstruct the trajectory of the ball. An example trajectory using the data from Figures 3.6 and 3.7 can be seen in Figure 3.9. The trajectory matches the expected parabolic trajectory, although it is not perfectly smooth. This roughness can be explained by the slight errors in

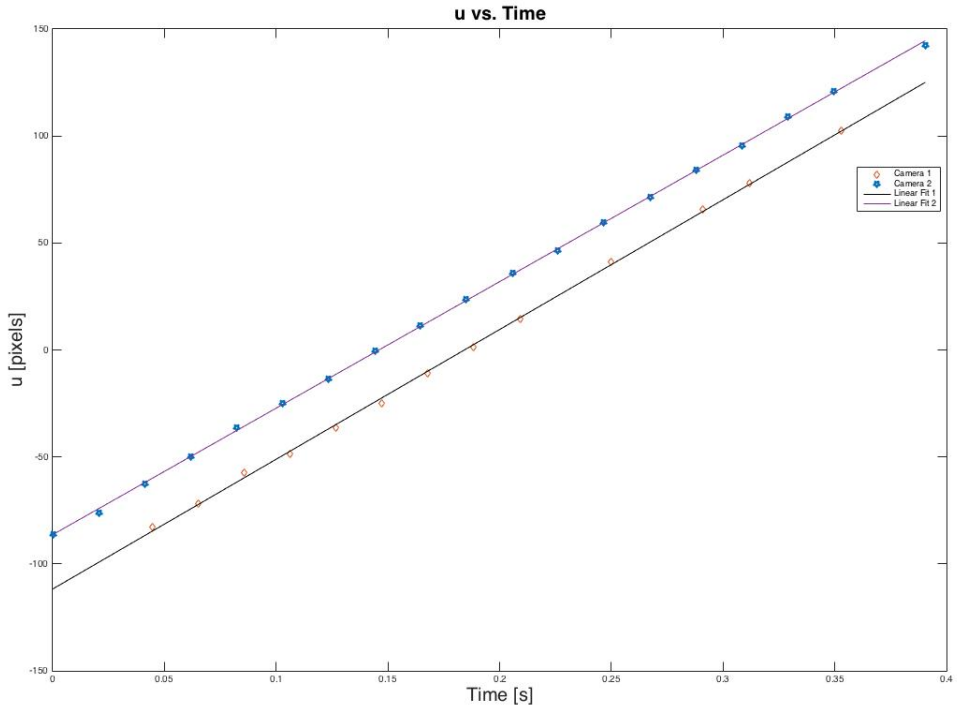


Figure 3.8: Linear regression of u versus time for both cameras

the measurement of the v location of the ball. Although the exact time-varying trajectory was not measured, the accuracy of the location estimates for a static object were measured, and proved to be good within one meter of the robot, with accuracy reducing after that. Because the trajectory looks as is expected and the measured locations of static objects is accurate, the measured trajectory using the linear regression model is assumed to be accurate enough for this application.

3.3.2 Predicting the Impact Point of the Ball

After getting a measurement of the ball's trajectory, the location where the ball will impact the robot needs to be calculated because the cameras will not detect the ball once the robot starts moving. To do this, the equations of motion of the ball must be used.

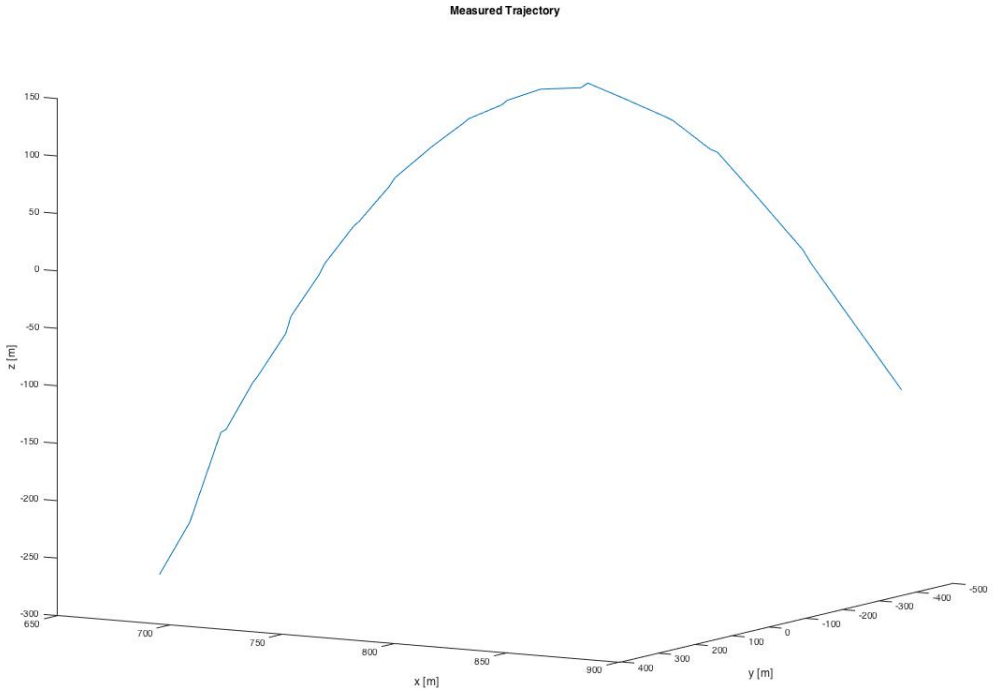


Figure 3.9: Measured trajectory using linear regression to estimate location between frames

The ball used is a red squash ball, with a diameter of $39mm$ and a mass of $13g$ (which is less than the typical squash ball, but the equations also hold for balls of typical weight). The ball follows a ballistic trajectory, and for throws of $5\frac{m}{s}$, the ball has a Reynolds number of approximately $1e4$. With this Reynolds number, the drag force is proportional to the squared velocity [2] resulting in the following equations of motion

$$\dot{\boldsymbol{v}} = \boldsymbol{g} - \alpha|\boldsymbol{v}|\,\boldsymbol{v} \tag{3.14}$$

$$\dot{\boldsymbol{x}} = \boldsymbol{v} \tag{3.15}$$

where $\boldsymbol{x}$ is the position of the ball in space, $\boldsymbol{v}$ is the velocity of the ball, $\boldsymbol{g}$ is the acceleration due to gravity, the ballistic coefficient $\alpha = \frac{C_d A \rho}{2m}$, the drag coefficient $C_d = 0.45$ for a sphere,

the air density at sea level $\rho = 1.29\frac{kg}{m^3}$. For the chosen ball, $\alpha = 0.11$so the drag force can therefore be ignored without much loss in accuracy for this purpose.

Using these equations, the ball location can be calculated using the last measurement for the beginning location and velocity. An example calculated using the first five measurements of the previous data is shown in figure 3.10. As can be seen, there is some error in the final predicted location. There is very little error in the angle of the trajectory, but the distance has an error of about 50mm, which is caused by errors in estimating the initial ball velocity. Although this is a significant error, it is less than half of the width or height of the catching arm and the robot should still be close enough to the actual ball location to catch the ball if the robot reaches the calculated location in time.

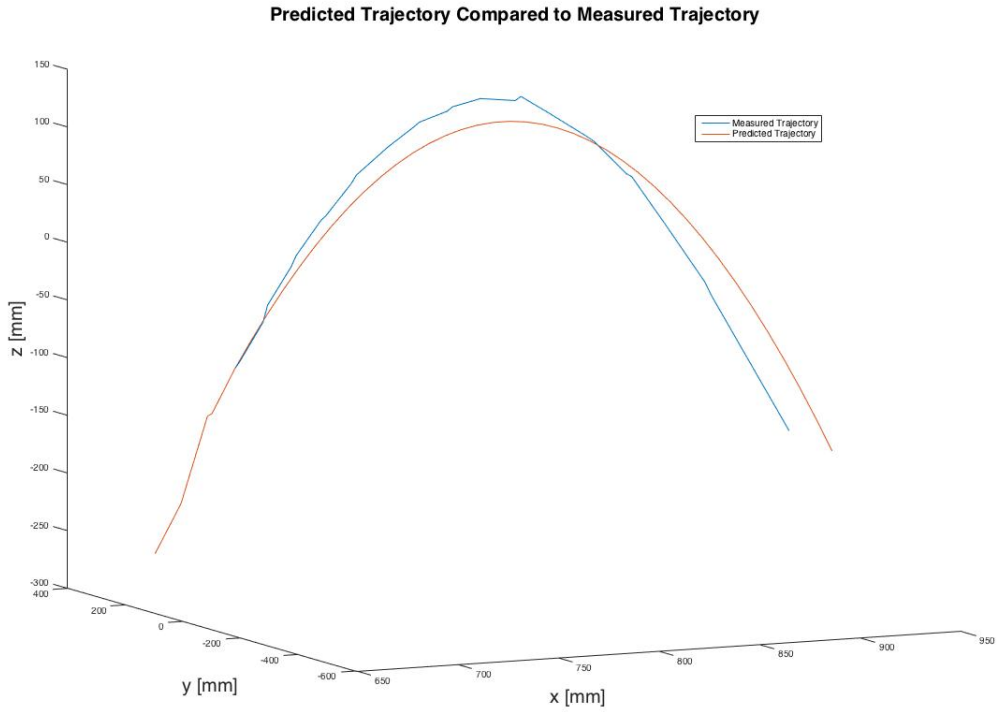


Figure 3.10: The ball trajectory predicted using the first five data points and the measured trajectory

Chapter 4

Control and Path Planning

4.1 Introduction

Once the location at which the ball will impact the robot is predicted, the robot must quickly drive to that spot to intercept the ball. Because of the choice to use a MIP platform, the robot cannot simply move directly to the desired spot while maintaining its heading if the ball is not directly in front of or behind the robot. Instead, the robot must turn towards the direction of the location, drive over, and turn back to face the ball. To get to the desired location and heading quickly, three different methods were tested: 1) using the desired end point to determine the heading angle of the robot and maintaining this angle until the end point is reached, 2) using the desired end point to determine the heading angle and updating the heading angle based on the distance of the robot to the the position, and 3) calculating a path to drive along using nonlinear model predictive control (NMPC). The first two methods are simple to implement, but the NMPC method requires some initial calculations.

4.2 NMPC Path Calculation

In nonlinear model predictive control, a nonlinear model of the system is used to simulate the behavior of the system in response to a given input u_k on the time period of interest, $t \in [0,T]$. A cost function $J(\boldsymbol{u})$ is used to penalize deviations of the system from the desired behavior. The gradient of the cost function,$\nabla J(u)$, is calculated using the adjoint, and is then used to find a new value of u_k. By iterating, a value of u_k can be found that minimizes the cost function $J(u)$. The general structure of the NMPC algorithm is as follows [12]:

1. Give an initial condition, and guess an initial control input $u_k(t)$ on $t \in [0,t]$
2. March the state equation forward until final time $t = T$
3. March the adjoint equation backward in time until initial time $t = 0$
4. Compute the gradient
5. Use the conjugate gradient method to update the control input
6. Repeat from step two until convergence

The state equation used is given by equation 2.15. The cost function chosen is

$$J(\boldsymbol{u}) = \frac{1}{2}\int_0^T (\boldsymbol{X}^H Q \boldsymbol{X} + \boldsymbol{u}^H R \boldsymbol{u}) dt + \frac{1}{2}\boldsymbol{X}(T)^H E^H Q_T E \boldsymbol{X}(T) \tag{4.1}$$

where Q, R, and Q_T are matrices chosen to produce the desired behavior. Q is used to penalize the trajectory of the system on the interval $t \in [0,T]$, and is used to keep the robot close to upright during simulation. R penalizes the control effort, and Q_T penalizes

deviations from the desired final state. The values of these matrices for this prototype are given in Appendix A. The adjoint is defined by

$$-E\frac{d\boldsymbol{r}}{dt} = A^H\boldsymbol{r} + Q\boldsymbol{X} \tag{4.2}$$

where E and A are defined by the tangent linear equation.

The tangent linear equation describes what happens to the system when small perturbations of the inputs, $\tilde{\boldsymbol{u}}(t)$ are applied to the system, resulting in small perturbations of the state $\tilde{\boldsymbol{X}}(t)$. It is found by linearizing the state equations about a trajectory $\boldsymbol{X}(\boldsymbol{u})$. This is done by taking $\boldsymbol{X} = \bar{\boldsymbol{X}} + \tilde{\boldsymbol{X}}$ and $\boldsymbol{u} = \bar{\boldsymbol{u}} + \tilde{\boldsymbol{u}}$ in equation 2.15. Then, a small angle approximation is performed, and only the terms that are linear in the perturbed quantities are kept. This results in an equation of the form

$$E\frac{d\tilde{\boldsymbol{X}}}{dt} = A\tilde{\boldsymbol{X}} + B\tilde{\boldsymbol{u}} \tag{4.3}$$

The values for A, and B are given in Appendix A. E has the same value as it does is equation 2.15

Using these equations, paths can be generated using the NMPC algorithm. A time of one second was chosen for the time interval. The initial values of $\boldsymbol{X}(t)$ were set equal to zero for all states, except for the position along the x and y axis. The position along the x and y axis can be chosen to be equal to the desired final location of the robot by shifting the coordinate system such that the final ball position is at the origin of the coordinate system. A sample generated path can be seen in Figure 4.1. The path has an initial backwards movement because the MIP requires that the robot move backwards initially before it is able to move forward stably. The path has an "s" shape, which matches what would be intuitively

expected for a non-holonomic platform such as the MIP.

Because NMPC is an iterative process, it can be time consuming to converge to a path; by the time a solution is found, the ball will have already landed. To get around this problem, a set of paths should be calculated offline, and stored on the robot for quick access. Before choosing the group of ball locations for which to calculate an store paths, the maximum area over which the robot can move needs to be calculated. The distance the robot moves in a straight line over the selected time period could be used, but this would produce a greater area than is physically possible, especially as the ball moves more to the side of the robot, resulting in a waste of time calculating impossible trajectories. Instead, the NMPC algorithm is used. First, a circle of points is generated that are guaranteed to be outside of the robots catching area. Next the NMPC algorithm is ran using each of these points in the initial state, then the distance the robot was actually able to travel was calculated for each of these points. This process resulted in Figure 4.2. With this area now defined, a grid of points is defined within the area, the NMPC algorithm is run using each of these points as an input, and the resulting trajectory of the robot is recorded.

Running the NMPC algorithm generates a set of motor inputs, which will theoretically move the robot along the exact calculated trajectory. In practice, it is quite difficult to model the system to the level of precision needed for this to work. A controller would also need to account for small imperfections in the model, as well as any small perturbations that take the robot off of its desired trajectory. Because of this limitation, instead of using the input calculated with the NMPC algorithm, a linear controller was designed to balance and drive the robot. This controller was used for all three driving strategies tested. The path that resulted from the NMPC algorithm was then used as a set of way-points for navigating the path.

4.3 Comparison of Driving Strategies

A successive loop closure approach was used for the balancing and driving of the robot. The inner loop controls the angle of the body, and the outer loop controls the position of the wheels. A lead-lag controller was used for both the inner and outer loops of the controller. The heading of the robot is governed by a PD controller that takes the output of the position controller and changes the relative movement of the two wheels to achieve the desired heading. The parameter values for these three controllers can be found in Appendix A as A.31 - A.34.

To compare the three driving strategies mentioned in section 4.1, the robot was placed on a clean wood floor. It balanced for five seconds, and then the a ramp signal of $9.8\frac{rad}{s}$ was used as input for the position controller which resulted in a gradual acceleration to $5.6\frac{rad}{s}$ over 2 seconds. After the position command was given, the heading input was determined by the driving strategy being tested to reach the desired end location of $(169mm, 204mm)$.

For the first strategy, called the basic heading strategy, the angle between the robot's current position and the desired end position was calculated from the robot's initial location, and this angle was used as the heading command for the entire test. The results from this test can be seen in Figure 4.3. The closest point the robot achieved to the desired location was off by $50.0mm$. The final heading angle was $0.72rad$, and this took path $2.37s$.

For the second strategy, called the updated heading strategy, the angle between the robot's current position and the desired end position was calculated from the robot's initial location. While the robot was driving, its position was calculated using equations 2.17 - 2.19. This position was used to find the actual angle between the robot and the desired end point, which was used to update the heading command at a rate of $200Hz$. The results from this test can be seen in Figure 4.4. The closest point the robot achieved to the desired location

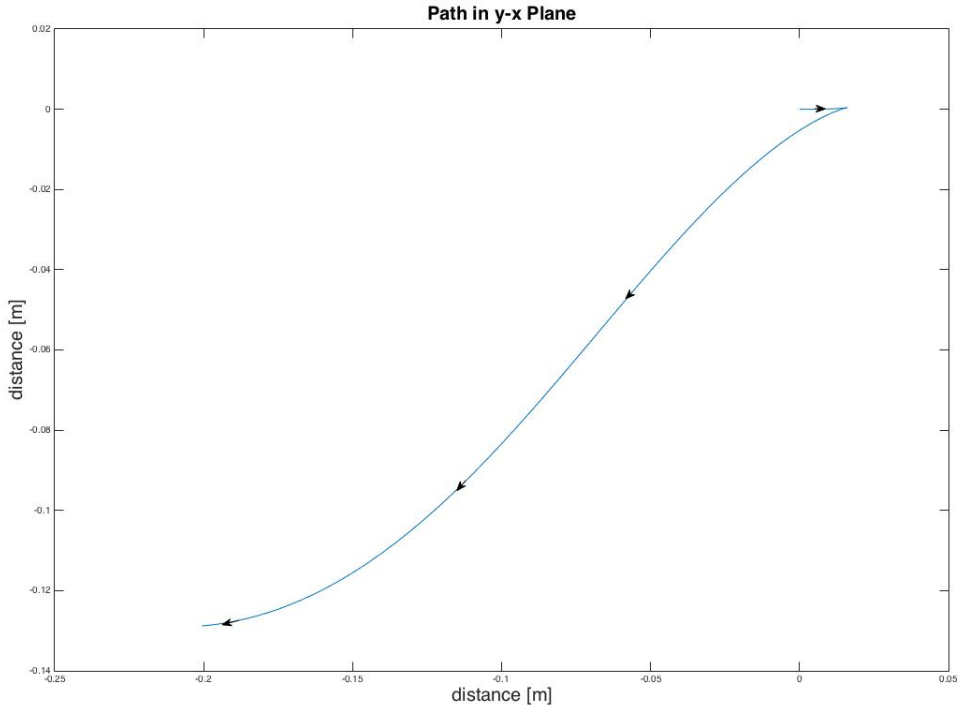
was off by 9.7mm. The final heading angle was 0.99rad, and this path took 2.36s.

The third strategy, called the NMPC strategy, a path was calculated to the desired end point using the NMPC algorithm detailed in section 4.2. The algorithm generates a large number of points, but it is unnecessary to use all of these points as way-points. Instead, a smaller subset should be taken. This size of this subset will depend on the robotic platform. Because the paths being taken are so short in comparison to the size of the robot, only four coordinates were used to get a good approximation of the path. With this method, the robot will not follow the exact path calculated by the NMPC algorithm and the path will therefore not be optimal. Figure 4.5 shows the path taken by the robot compared with the path calculated using the NMPC algorithm. There is some deviation from the desired path, but the robot still follows the s-shape of the NMPC path. The closest point the robot achieved to the desired location was off by 2.7mm. The final heading angle was 0.47rad, and this took 2.51s.

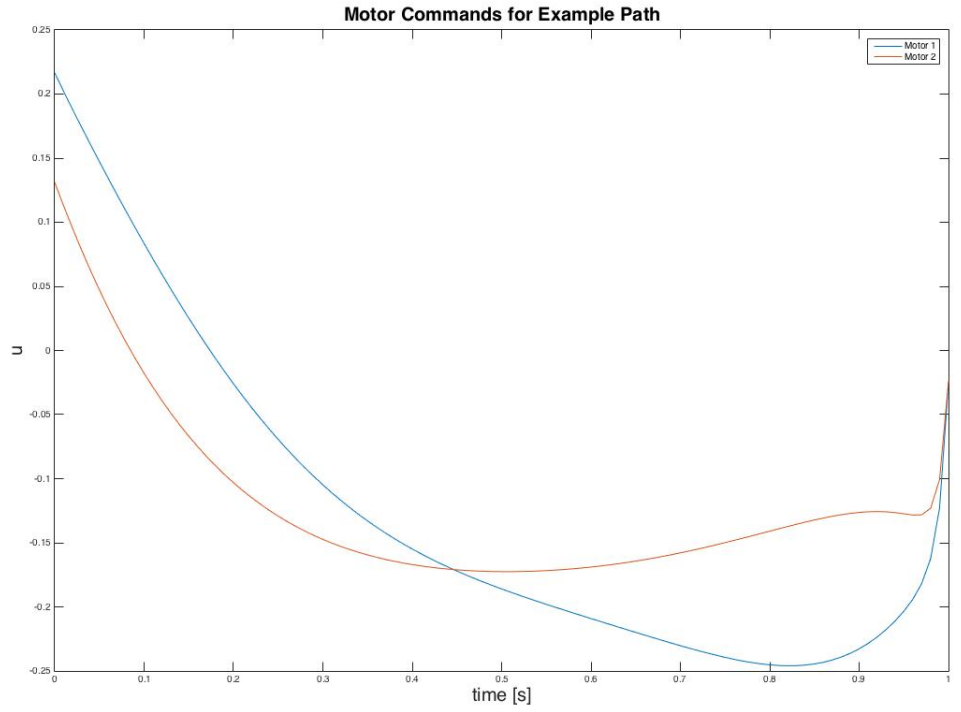
The NMPC strategy came in as the slowest of the three strategies. It was about nine percent slower than the other two strategies, but it was the most accurate in final location. All three strategies were slow to achieve the commanded heading as can be seen in Figure 4.3. It takes the robot about one second to turn one radian. This explains the large final position error of the basic heading strategy. Because this strategy does not update the heading angle, it will necessarily be inaccurate if the robot does not turn before starting to drive, and is greatly affected by the speed at which the robot can turn. The updated heading strategy also has its accuracy reduced by the slow turning response of the robot. The heading controller of the robot could be tuned more aggressively, but this would likely result in undesirable overshoot and oscillations, which would decrease the accuracy of the first strategy. The NMPC strategy ended with a heading angle closest to the desired heading of 0rad, and in

general has a greater ability to adjust the final heading of the robot without decreasing speed greatly. To end at a similar heading angle to the NMPC strategy, both the first and second strategies would require the robot to stop at the end location and turn around, which could take one second or more just to turn to the desired heading. Because the catching arm is rigidly mounted to the robot body, it is essential that the robot end up at an appropriate angle, or the ball will bounce off of the side of the arm instead of being caught.

Even though the NMPC strategy was a bit slower in this test, it is the best strategy for this prototype because it results in the robot being closest to the desired final heading. Because the NMPC strategy often results in a correct final heading, the robot is able to keep driving past the desired location, thus reducing the total time taking over the other two strategies, which would need to stop and correct heading. Driving past the desired location also allows the robot to maintain the correct body angle for the ball to be caught in the arm if the robot does not have time to stop at the location and then adjust its body angle.



(a) An example path generated by the NMPC algorithm



(b) Motor commands generated by the NMPC algorithm

Figure 4.1: Results from the NMPC algorithm

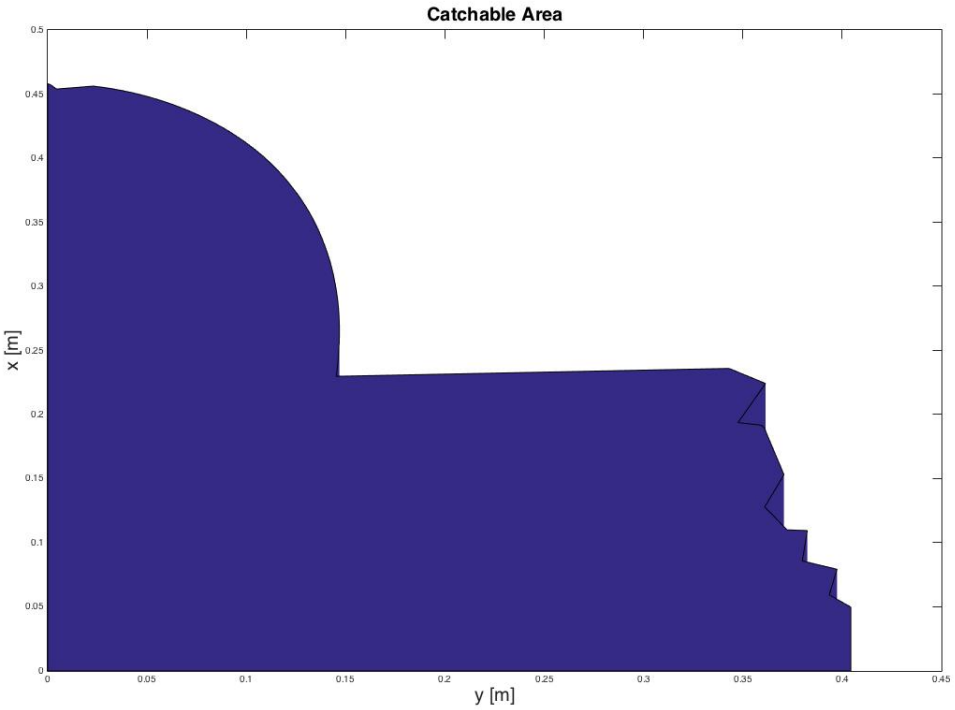
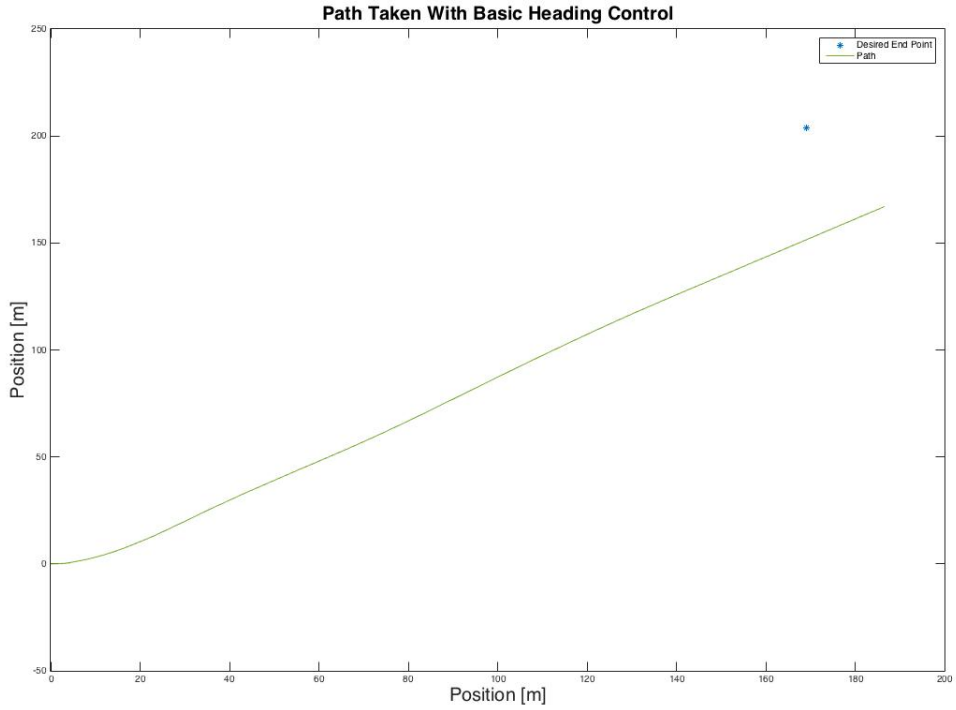
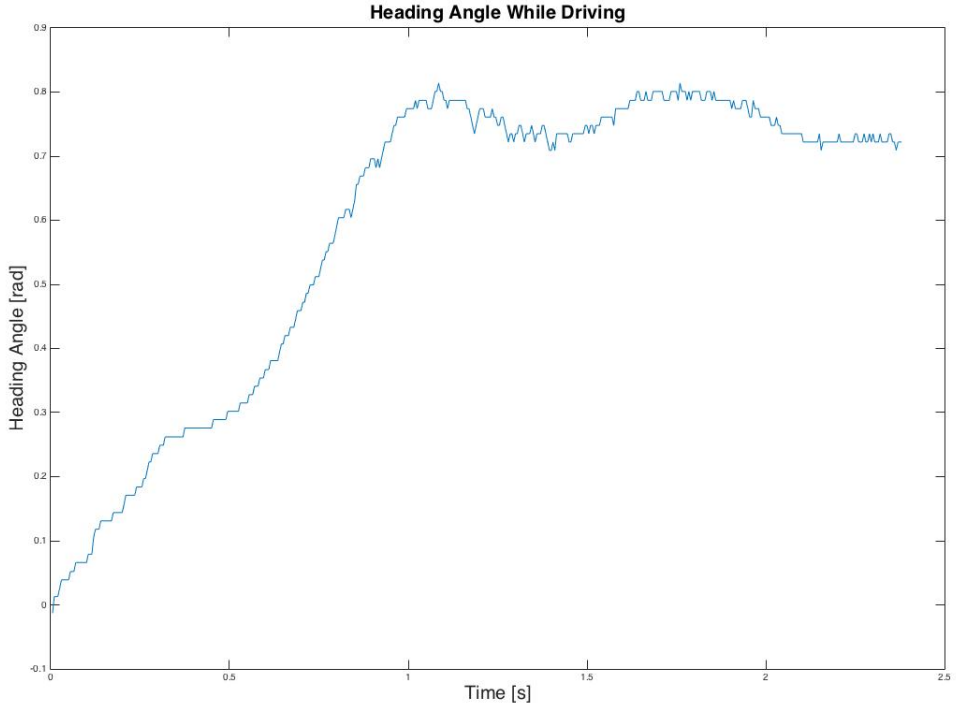


Figure 4.2: The approximate area in which the robot can move within one second

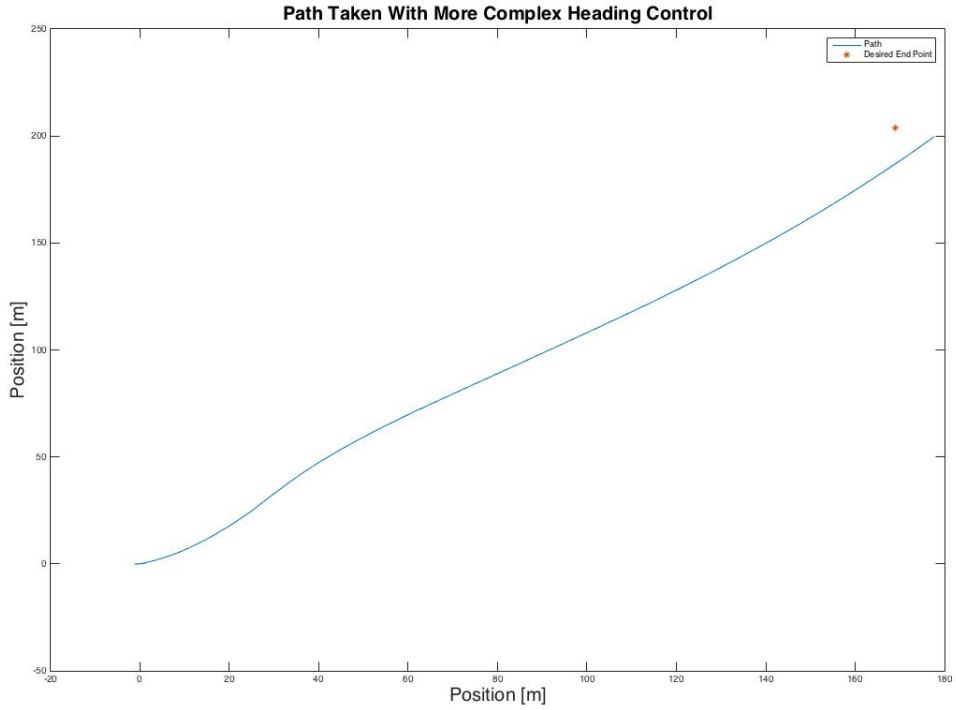


(a) The path driven by the robot when testing the simple heading driving strategy

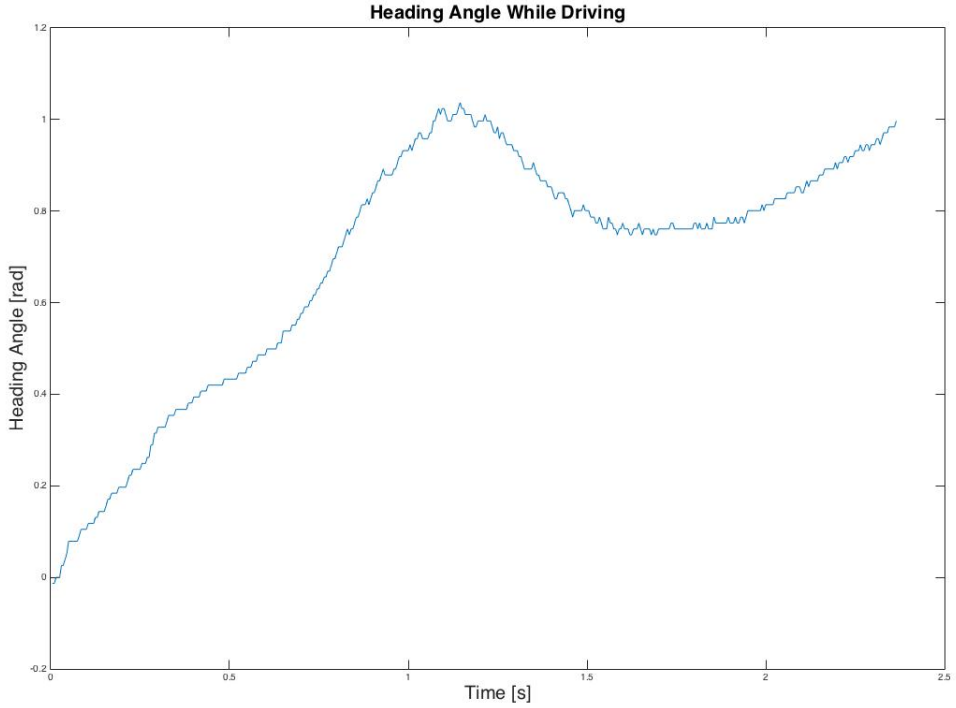


(b) The heading angle of the robot during the basic heading

Figure 4.3: The path taken and heading angle using the simple heading driving strategy

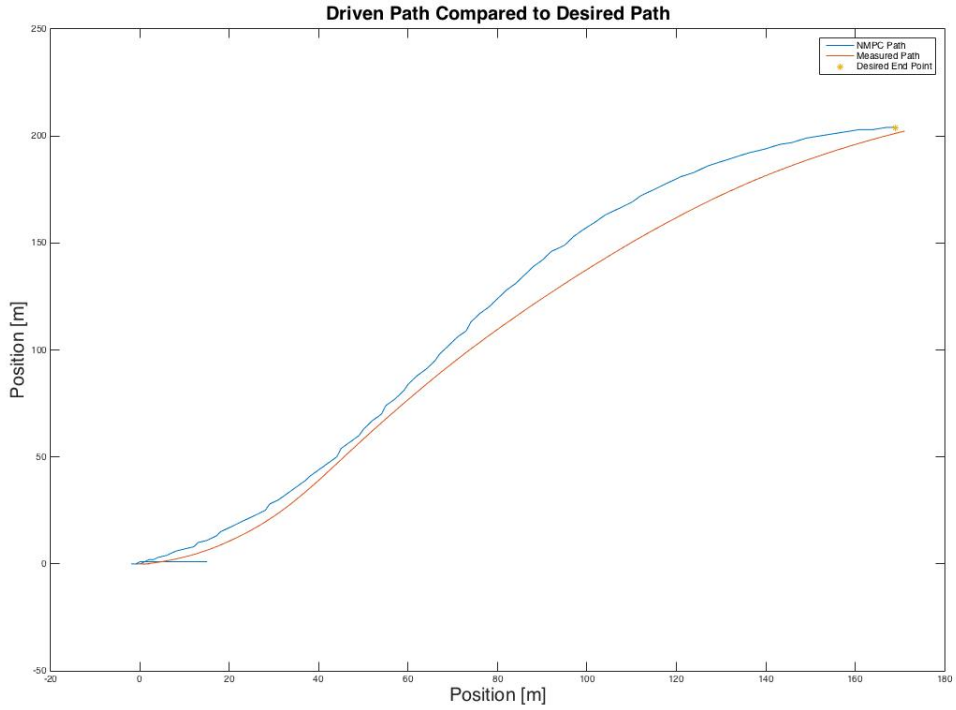


(a) The path driven by the robot when testing the updated heading driving strategy

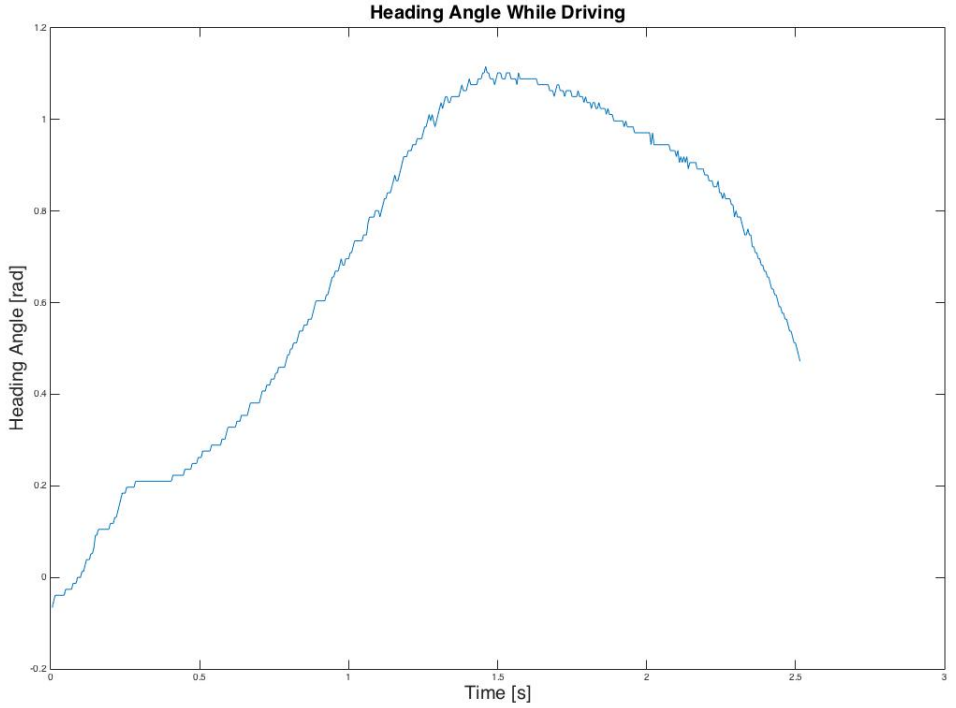


(b) The heading angle of the robot during the updated heading test

Figure 4.4: The path taken and heading angle using the updated heading driving strategy



(a) The path driven by the robot when testing the NMPC driving strategy



(b) The heading angle of the robot during the NMPC test

Figure 4.5: The path taken and heading angle using the NMPC driving strategy

Chapter 5

Testing the Prototype

5.1 Introduction

With the ball trajectory calculations written and the driving paths calculated, the last thing to be done before the prototype can be tested is to develop software to integrate all of the necessary functions. As mentioned in 2.2, the robot is controlled by a BeagleBone Black (BBB) with a robotics cape. Because there are libraries written for the BBB, which handle all of the low-level functions such as commanding motors and reading sensors, only the high-level functions to perform the desired task must be written. Once this task is completed, the catching performance of the robot can be tested.

5.2 Outline of the Code

The code was written to interface with James Strawson's balancing code for the Beagle Bone Black [13]. This existing code is a multi-threaded program written in C that handles all of the sensor fusion, state estimation, and balancing of the robot using the

controller from section 4.3. Interfacing with this code is as simple as adding more threads to the program. All of the camera calculations and measurements can be performed in one thread, while the path driving can be performed in another.

To make sure that the camera data is recorded as soon as it is taken, both cameras are checked every millisecond. Once a reading is received from one camera, the code checks for a reading from the other camera. If no reading is received within twenty milliseconds, then the second camera has not seen the object and the code goes back to checking both cameras. If a reading from the other camera is received before twenty milliseconds have elapsed, then each measurement is calibrated as outlined in 3.2.2. Next the code checks for new measurements from the cameras and then calibrates each as it is read. This process lasts until four measurements from one camera have been taken or no measurements have happened in twenty milliseconds, whichever happens first. If the ball moves out of the frame before each camera has taken at least two measurements, and one has taken at least three, then the code restarts because this is not enough data to make a prediction. If enough measurements have been gathered, then the two linear regressions are calculated using the calibrated measurements as outlined in 3.3.1. From this, the equations of 3.3.2 can be used to predict the ball's future location. If the ball will not travel high enough to reach the catching arm, then the code restarts. If the ball will reach the necessary height, then the location of the ball is predicted until it passes through the apex of its trajectory and then until it reaches the same height as the catching arm.

Once the impact location of the ball is calculated, the information is passed to the thread that handles path driving. First, the code checks whether the predicted location of the ball is within the catching area of the robot. If not, the camera thread restarts and the robot does not move. If the ball is catchable, the code checks which pre-calculated path

ends closest to the ball location using a lookup table. Then, the speed of the robot is set to the maximum speed, and the heading is calculated using the first path point. While driving, the location of the robot is calculated using the Euler method. When the robot's location is within twenty millimeters of the first path point, the next point is used to determine the heading. When the second to last path point is reached, the location of the ball is used instead of the last path point. Once the robot reaches the last point, it reverses its driving direction. It does not wait at the last point, because it cannot catch a ball while the robot is upright. It then heads back towards the starting location to make setting up for multiple catches easier. Finally, it restarts the camera reading code and waits for another ball to be thrown.

5.3 Results

This code was combined with the Strawson's balancing code and compiled on the BBB. To test the prototype, the robot was placed on a clean table, in a well lit room with no red objects in the background. The ball throws used to test the robot were started below the line on the table, and between 0.2 and 1 meter in front of the robot. Only throws that land behind the robot were tested, as the robot naturally tilts to the correct catching angle when moving backward. Multiple trials were conducted to test the catching ability of the robot. Typical results can be seen in Figures 5.1 and 5.2. The robot was unable to catch balls that had an impact location greater than $0.15m$ away from the robot, or with angles greater than about $0.17rad$. While the ball did land in the catching arm occasionally, it would often bounce off of the top or sides of the arm. To reduce this issue, foam tape was added inside of the arm.

The catching area is much smaller than was predicted by the NMPC algorithm. This

discrepancy is most likely caused by inaccuracies in modeling the physical parameters of the robot, most likely the motor parameters. As can be seen in Figure 5.2 the robot was able to correctly measure the angle of the ball in relation to itself, showing that the camera calculations outlined in Chapter 3 were accurate enough for the task. Te poor performance must be caused by the robotic platform, which was slow to accelerate and unable to achieve its maximum speed while the ball was still in the air.

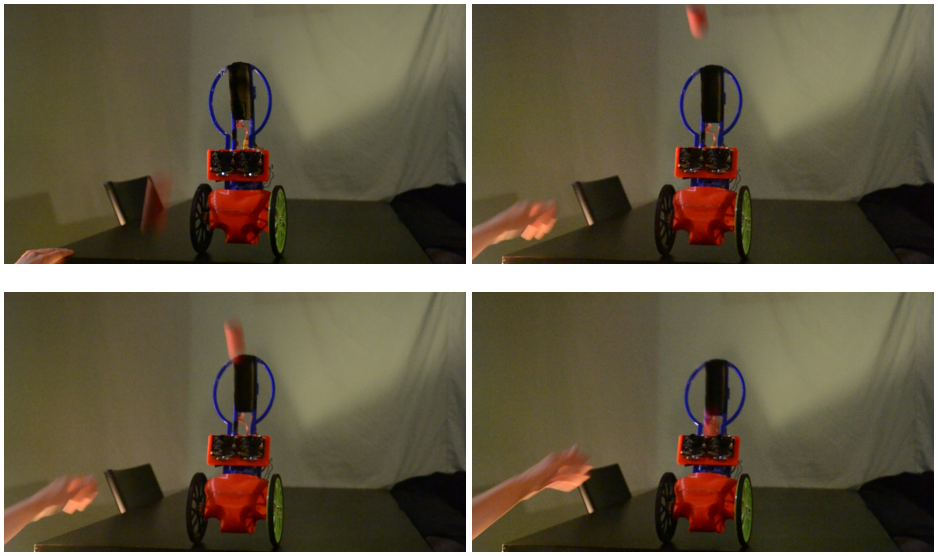


Figure 5.1: The robot successfully catching a thrown ball

To measure the maximum speed and the acceleration of the robot, a step input was used in the position controller The results of this test can be seen in Figure 5.3. The maximum speed of the robot was about 3m/s, which it achieved after about 1.7s. The angle of the body during this test can be seen in figure 5.4. The initial negative angle is necessary for the robot to move backwards because of the MIP design, but causes a 0.2s delay in achieving the maximum speed.

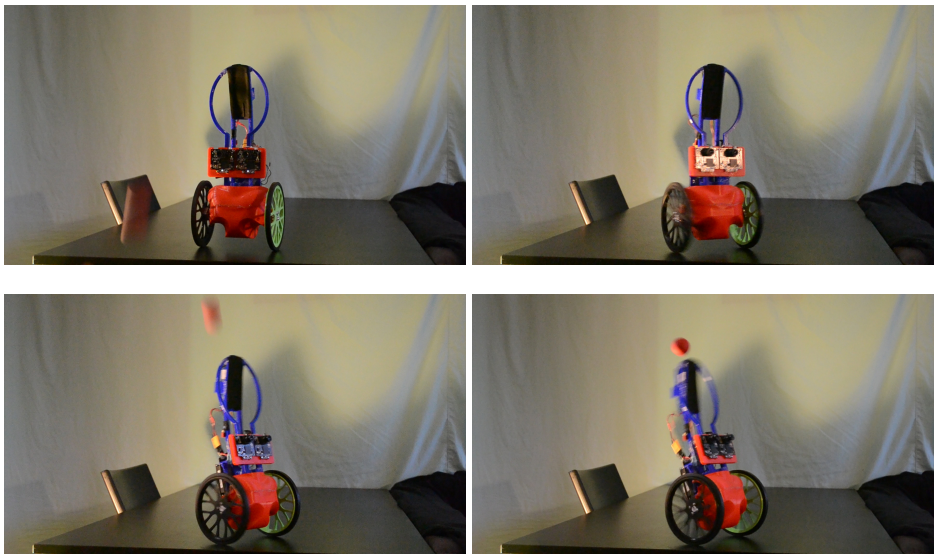


Figure 5.2: An unsuccessful attempt at catching a ball thrown to the right of the robot

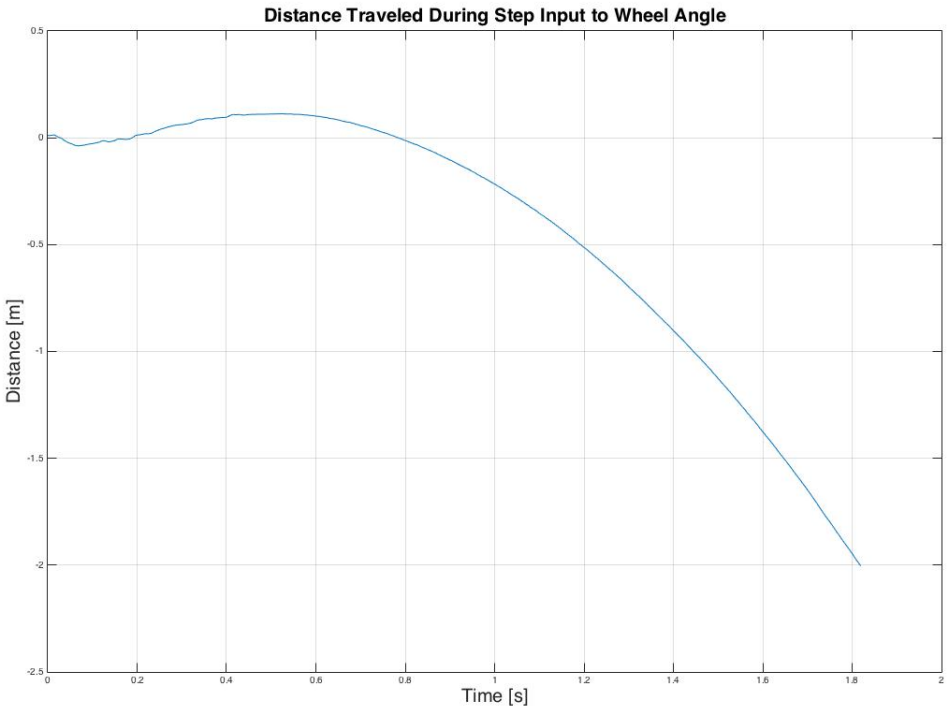


Figure 5.3: Distance traveled by robot after a step input to wheel angle

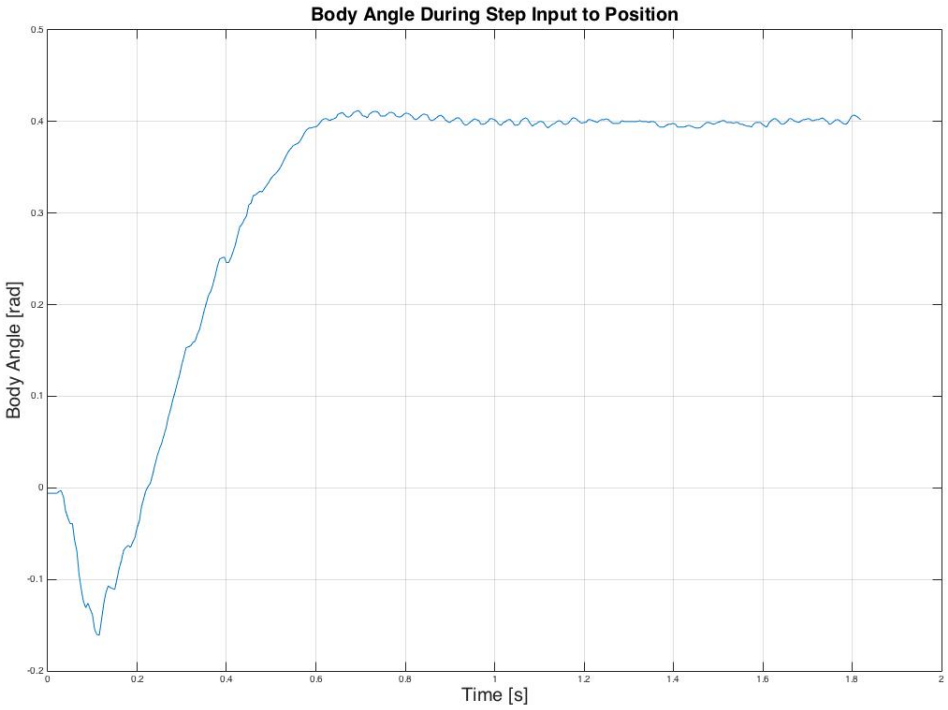


Figure 5.4: Body angle of robot after a step input to wheel angle

Chapter 6

Conclusions and Further Work

The goal of catching a ball with a low-cost robot prototype was accomplished. The robot was able to catch balls thrown from a distance of up to one meter that would have landed 0.15 meters or less directly behind the robot. The robot was also able to catch balls that did not land directly behind it, at a distance of 0.08 meters and less than ten degrees, as measured from the x-axis of the robot. While the primary goal was accomplished, the results were not as compelling as desired or predicted. To make the catching function interesting to a consumer, the robot would need to catch the ball in a larger area than it currently does. The small catching area is mostly caused by the speed at which the robot drives to a given location. The current robot design does have a fast maximum speed of $3\frac{m}{s}$, but to accelerate to this speed takes almost two seconds, which severely reduces the catching area of the robot. The discrepancy between the predicted catching area and the achieved catching area is most likely caused by inaccuracies in modeling the motors of the platform.

The catching area should be increased in future iterations of the robot. This objective could be accomplished by changing the parameters of the robot such that it has a faster acceleration rate. This change would require examining the motor choice and body size, and

finding the best placement of the center of mass of the robot.

Different driving strategies should also be explored in future iterations. Proportional navigation as shown in [14] is one such strategy. Other strategies that allow new measurements taken with the cameras to be incorporated into the path driving should also be investigated.

Because the ball often bounced out of the catching arm, the arm could be modified in future iterations to decrease the amount the ball bounces. One interesting strategy would be to experiment with different materials, such thermoplastic polyurethane which is more elastic than PLA and might absorb the energy of the ball better, reducing the amount of bounce. Another strategy that was not examined was a more sophisticated catching maneuver, such as swinging the arm forward after ball impact to better capture the ball inside the arm. This technique was not explored for this prototype because of the inability to move to the desired position in time, leaving no time for extra maneuvers.

The catching area could also be increased if the ball could be thrown from farther away, giving the robot more time to react. The low resolution of the Pixy camera combined with a small ball only allowed detection up to about $0.8m$ away from the cameras. To increase the range the smallest detected color area could be lowered, which would provide a larger range, but this could lead to false identifications if there are nearby items that have a similar color to the ball. The size of the ball could also be increased, but this would necessitate building a larger robot which would most likely decrease the maneuverability of the robot. The most likely solution to this problem is to change the type of camera used. At time of designing this prototype, there were no better cameras in the desired price range. One solution would be to design a camera similar to the Pixy that would be optimized for this project, including the capability to trigger both camera shutters at the same time. It is

also possible that better cameras in the same price range will come out in the future, making this project more feasible.

Appendix A

Equations and Values

$$
\begin{aligned}
L = -gL_b m_b \cos(\theta(t)) + \frac{1}{2L_w^2}((L_w^2(I_{w_2} + m_w R^2) + I_{w_1} R^2 \cos(\phi_1(t))^2 + \\
I_{w_1} R^2 \sin(\phi_1(t))^2)\phi_1'(t)^2 - R^2(2I_{w_1} + R^2(I_{w_1}(\cos(\phi_1(t))^2 + \sin(\phi_1(t))^2))\phi_2'(t)^2) + \\
\frac{1}{2L_w^2}(R^2(I_{w_1}\cos(\phi_2(t))^2 + I_{w_1}\sin(\phi_2(t))^2)\phi_1'(t)^2 - \\
R^2(2I_{w_1} + (L_w^2(I_{w_2} + m_w R^2) + I_{w_1} R^2 \cos(\phi_2(t))^2 + I_{w_1} R^2 \sin(\phi_2(t))^2)\phi_2'(t)^2) + \\
\frac{1}{2}(I_2\theta'(t)^2 + \frac{I_3 R^2 \cos(\theta(t))^2(\phi_1'(t) - \phi_2'(t))^2}{L_w^2} + \frac{I_1 R^2 \sin(\theta(t))^2(\phi_1'(t) - \phi_2'(t))^2}{L_w^2} + \\
m_b(L_b^2 \sin(\theta(t))^2\theta'(t)^2 + (L_b \cos(\theta(t))\cos(\frac{R(\phi_1(t) - \phi_2(t))}{L_w}\theta'(t) - \\
(L_b R \sin(\theta(t))\sin((R(\phi_1(t) - \phi_2(t)))/L_w)(\phi_1'(t) - \phi_2'(t)))/L_w + \\
\frac{R}{2}\cos((R(\phi_1(t) - \phi_2(t)))/L_w)(\phi_1'(t) + \phi_2'(t)))^2 + \\
(L_b \cos(\theta(t))\sin((R(\phi_1(t) - \phi_2(t)))/L_w)\theta'(t) + \\
(L_b R \cos((R(\phi_1(t) - \phi_2(t)))/L_w)\sin(\theta(t))(\phi_1'(t) - \phi_2'(t)))/L_w + \\
\frac{R}{2}\sin((R(\phi_1(t) - \phi_2(t)))/L_w)(\phi_1'(t) + \phi_2'(t)))^2)) \quad \text{(A.1)}
\end{aligned}
$$

$$C_1 = \begin{bmatrix} 2.48146208408e+02 & 0 & 1.42787261028e+02 \\ 0 & 2.48183441649e+02 & 1.01498200795e+02 \\ 0 & 0 & 1 \end{bmatrix} \tag{A.2}$$

$$d_1 = \begin{bmatrix} -4.4928089109319530e-01 \\ 2.5788853463591183e-01 \\ 1.2640764884591432e-03 \\ -2.1476066530935907e-04 \\ -8.5764088284348422e-02 \end{bmatrix} \tag{A.3}$$

$$C_2 = \begin{bmatrix} 2.46231309794e+02 & 0 & 1.47631903408e+02 \\ 0 & 2.46335102625e+02 & 1.03762459983e+02 \\ 0 & 0 & 1 \end{bmatrix} \tag{A.4}$$

$$d_2 = \begin{bmatrix} -4.4942525129197014e-01 \\ 2.5203419101027347e-01 \\ 1.7529221950900113e-03 \\ -3.4494025486728609e-04 \\ -8.0849238090097086e-02 \end{bmatrix} \tag{A.5}$$

$$E = \left[\begin{array}{c|c} E_{1,1} & 0 \\ \hline 0 & I \end{array}\right] \tag{A.6}$$

$$E_{1,1} = \left[\begin{array}{c|c} I & 0 \\ \hline 0 & M \end{array}\right] \tag{A.7}$$

$$M_{1,1} = I_2 + L_b^2 m_b \tag{A.8}$$

$$M_{1,2} = \frac{1}{2} L_b m_b R cos(\theta(t)) \tag{A.9}$$

$$M_{1,3} = \frac{1}{2} L_b m_b R cos(\theta(t)) \tag{A.10}$$

$$M_{2,1} = \frac{1}{2} L_b m_b R cos(\theta(t)) \tag{A.11}$$

$$M_{2,2} = \frac{4I_{w_2}L_w^2 + (4I_1 + 8I_{w_1} + L_w^2 m_b + 2L_b^2 m_b + 4L_w^2 m_w)R^2 - 2L_b^2 m_b R^2 cos(2\theta(t))}{4L_w^2} \tag{A.12}$$

$$M_{2,3} = \frac{R^2(-4I_1 - 8I_{w_1} + L_w^2 m_b - 2L_b^2 m_b + 2L_b^2 m_b cos(2\theta(t)))}{4L_w^2} \tag{A.13}$$

$$M_{3,1} = \frac{1}{2} L_b m_b R cos(\theta(t)) \tag{A.14}$$

$$M_{3,2} = \frac{R^2(-4I_1 - 8I_{w_1} + L_w^2 m_b - 2L_b^2 m_b + 2L_b^2 m_b cos(2\theta(t)))}{4L_w^2} \tag{A.15}$$

$$M_{3,3} = \frac{4I_{w_2}L_w^2 + (4I_1 + 8I_{w_1} + L_w^2 m_b + 2L_b^2 m_b + 4L_w^2 m_w)R^2 - 2L_b^2 m_b R^2 cos(2\theta(t))}{4L_w^2} \tag{A.16}$$

$$N = \begin{bmatrix} \theta'(t) \\ \phi_t'(t) \\ \phi_2'(t) \\ N_4 \\ N_5 \\ N_6 \\ \frac{R}{2}\cos(\gamma(t))(\phi_1'(t)+\phi_2'(t)) \\ \frac{R}{2}\sin(\gamma(t))(\phi_1'(t)+\phi_2'(t)) \\ \frac{R}{L_w}(\phi_1'(t)-\phi_2'(t)) \end{bmatrix} \tag{A.17}$$

$$N_4 = -gL_b m_b \sin(\theta(t)) - \frac{(I_1 - I_3 + L_b^2 m_b)R^2 \sin(2\theta(t))(\phi_1'(t)-\phi_2'(2))^2}{2L_w^2} - \frac{V}{\Omega k}(u_1(t)+u_2(t)) - \frac{1}{\Omega k^2}(2\theta'(t)-\phi_1'(t)-\phi_2'(t)) \tag{A.18}$$

$$N_5 = -\frac{1}{2}L_b m_b R \sin(\theta(t))\theta'(t)^2 + \frac{(I_1 - I_3 + L_b^2 m_b)R^2 \sin(2\theta(t))\theta'(t)(\phi_1'(t)-\phi_2'(t))}{L_w^2} + \frac{V}{\Omega k}u_1 + \frac{1}{\Omega k^2}(\theta'(t)-\phi_1'(t)) \tag{A.19}$$

$$N_6 = -\frac{1}{2}L_b m_b R \sin(\theta(t))\theta'(t)^2 - \frac{(I_1 - I_3 + L_b^2 m_b)R^2 \sin(2\theta(t))\theta'(t)(\phi_1'(t)-\phi_2'(t))}{L_w^2} + \frac{V}{\Omega k}u_2 + \frac{1}{\Omega k^2}(\theta'(t)-\phi_2'(t)) \tag{A.20}$$

$$A = \left[\begin{array}{c|c} A_{1,1} & 0 \\ \hline 0 & A_{2,2} \end{array}\right] \tag{A.21}$$

$$A_{1,1} = \begin{bmatrix} A_a^T \\ A_b^T \\ A_c^T \end{bmatrix} \tag{A.22}$$

$$A_a = \begin{bmatrix} A_{a_1} \\ 0 \\ 0 \\ \frac{-2}{\Omega k^2} \\ -\frac{((I_1-I_3+L_b^2 m_b)R^2 \sin[2\theta_b(t)](\phi_1'(t)-\phi_2'(t)))}{L_w^2} + \frac{1}{\Omega k^2} \\ \frac{((I_1-I_3+L_b^2 m_b)R^2 \sin[2\theta_b(t)](\phi_1'(t)-\phi_2'(t)))}{L_w^2} + \frac{1}{\Omega k^2} \end{bmatrix} \tag{A.23}$$

$$A_{a_1} = \frac{1}{2L_w^2}(2(I_1 - I_3 + L_b^2 m_b)R^2 \cos(2\theta_b(t))(\phi_{1b}'(t) - \phi_{2b}'(t))^2 + L_w^2 L_b m_b(2g\cos(\theta_b(t)) + R\sin()\theta_b(t))\phi_{1b}''(t) + R\sin(\theta_b(t))\phi_{2b}''(t))) \tag{A.24}$$

$$A_b = \begin{bmatrix} A_{b_1} \\ 0 \\ 0 \\ \frac{((I_1-I_3+L_b^2 m_b)R^2 \sin(2\theta_b(t))(\phi_1'(t)-\phi_2'(t)))}{L_w^2} - L_b m_b R \sin[\theta_b(t)]\theta_b'(t) + \frac{1}{\Omega k^2} \\ \frac{2(I_1-I_3+L_b^2 m_b)R^2 \cos(\theta_b(t))\sin(\theta_b(t))\theta_b'(t)}{L_w^2} - \frac{1}{\Omega k^2} \\ -\left(\frac{2(I_1-I_3+L_b^2 m_b)R^2 \cos(\theta_b(t))\sin()\theta_b(t))\theta_b'(t)}{L_w^2}\right) \end{bmatrix} \tag{A.25}$$

$$A_{b_1} = \frac{1}{2L_w^2}(R(-L_w^2 L_b m_b \cos(\theta_b(t))\theta_b'(t)^2 + 4(I_1 - I_3 + L_b^2 m_b)R\cos(2\theta_b(t))\theta_b'(t)(\phi_{1b}'(t) - \phi_{2b}'(t)) + \sin[\theta_b(t)](-L_w^2 L_b m_b \theta_b''(t) + 4(I_1 - I_3 + L_b^2 m_b)R\cos(\theta_b(t))(\phi_{1b}''(t) - \phi_{2b}''(t)))) \tag{A.26}$$

$$A_c = \begin{bmatrix} A_{c_1} \\ 0 \\ 0 \\ -\frac{((I_1-I_3+L_b^2 m_b)R^2 \sin(2\theta_b(t))(\phi_1'(t)-\phi_2'(t)))}{L_w^2} - L_b m_b R \sin[\theta_b(t)]\theta_b'(t) + \frac{1}{\Omega k^2} \\ -\frac{2(I_1-I_3+L_b^2 m_b)R^2 \cos(\theta_b(t))\sin(\theta_b(t))\theta_b'(t)}{L_w^2} \\ \left(\frac{2(I_1-I_3+L_b^2 m_b)R^2 \cos(\theta_b(t))\sin(\theta_b(t))\theta_b'(t)}{L_w^2}\right) - \frac{1}{\Omega k^2} \end{bmatrix} \tag{A.27}$$

$$A_{c_1} = -\frac{1}{2L_w^2}(R(-L_w^2 L_b m_b \cos(\theta_b(t))\theta_b'(t)^2 + 4(I_1 - I_3 + L_b^2 m_b)R\cos(2\theta_b(t))\theta_b'(t)(\phi_{1b}'(t) - \phi_{2b}'(t)) + \sin(\theta_b(t))(L_w^2 L_b m_b \theta_b''(t) + 4(I_1 - I_3 + L_b^2 m_b)R\cos(\theta_b(t))(\phi_{1b}''(t) - \phi_{2b}''(t)))) \quad \text{(A.28)}$$

$$A_{2,2} = \begin{bmatrix} 0 & \frac{R}{2}\cos(\gamma_b(t)) & \frac{R}{2}\cos(\gamma_b(t)) & 0 & 0 & -\frac{R}{2}\sin(\gamma_b(t))(\phi_{1b}'(t) + \phi_{2b}'(t)) \\ 0 & \frac{R}{2}\sin(\gamma_b(t)) & \frac{R}{2}\sin(\gamma_b(t)) & 0 & 0 & \frac{R}{2}\cos(\gamma_b(t))(\phi_{1b}'(t) + \phi_{2b}'(t)) \\ 0 & \frac{R}{L_w} & \frac{-R}{L_w} & 0 & 0 & 0 \end{bmatrix} \quad \text{(A.29)}$$

$$B = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ -\frac{V}{\Omega k} & -\frac{V}{\Omega k} \\ \frac{V}{\Omega k} & 0 \\ 0 & \frac{V}{\Omega k} \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \quad \text{(A.30)}$$

$$G_{inner} = 7.5\frac{-6.289 + 11.91z^{-1} - 5.634z^{-2}}{1.0 - 1.4z^{-1} + 0.4z^{-2}} \quad \text{(A.31)}$$

$$G_{outer} = -0.02\frac{0.887 - 0.8807z^{-1}}{1.0 - 0.9649z^{-1}} \quad \text{(A.32)}$$

$$K_p = 0.135 \quad \text{(A.33)}$$

$$K_d = 0.01 \tag{A.34}$$

$$Q = 0 \tag{A.35}$$

$$R = 0 \tag{A.36}$$

$$Q_T = \begin{bmatrix} 0.5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.04 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.04 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1.5 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.001 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.001 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 30 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 30 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{A.37}$$

Bibliography

[1] H. H. Rapp, "A ping-pong ball catching and juggling robot: a real-time framework for vision guided acting of an industrial robot arm," in *Automation, Robotics and Applications (ICARA), 2011 5th International Conference on*. IEEE, 2011, pp. 430–435.

[2] U. Frese, B. Bauml, S. Haidacher, G. Schreiber, I. Schäfer, M. Hahnle, and G. Hirzinger, "Off-the-shelf vision for a robotic ball catcher," in *Intelligent Robots and Systems, 2001. Proceedings. 2001 IEEE/RSJ International Conference on*, vol. 3. IEEE, 2001, pp. 1623–1629.

[3] G. Bätz, A. Yaqub, H. Wu, K. Kühnlenz, D. Wollherr, and M. Buss, "Dynamic manipulation: Nonprehensile ball catching," in *Control & Automation (MED), 2010 18th Mediterranean Conference on*. IEEE, 2010, pp. 365–370.

[4] V. Lippiello and F. Ruggiero, "3d monocular robotic ball catching with an iterative trajectory estimation refinement," in *Robotics and Automation (ICRA), 2012 IEEE International Conference on*. IEEE, 2012, pp. 3950–3955.

[5] R. Mori, K. Hashimoto, and F. Miyazaki, "Tracking and catching of 3d flying target based on gag strategy," in *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*, vol. 5. IEEE, 2004, pp. 5189–5194.

[6] L. Freda and G. Oriolo, "Vision-based interception of a moving target with a nonholonomic mobile robot," *Robotics and Autonomous Systems*, vol. 55, no. 6, pp. 419–432, 2007.

[7] F. Miyazaki and R. Mori, "Realization of ball catching task using a mobile robot," in *Networking, Sensing and Control, 2004 IEEE International Conference on*, vol. 1. IEEE, 2004, pp. 58–63.

[8] P.-T. Chen, "Simulation and optimization of a two-wheeled, ball-flinging robot," 2010.

[9] "Overview - CMUcam5 Pixy," http://cmucam.org/projects/cmucam5, accessed: 2017-05-23.

[10] W. Faig, "Calibration of close-range photogrammetric systems: Mathematical formulation," *Photogrammetric engineering and remote sensing*, vol. 41, no. 12, 1975.

[11] G. Bradski, *Dr. Dobb's Journal of Software Tools*, 2000.

[12] T. Bewley, *Numerical Renaissance: Simulation, Optimization, and Control*. San Diego, CA: Renaissance Press, 2012.

[13] J. Strawson, "Robotics cape installer," https://github.com/StrawsonDesign/Robotics_Cape_Installer, 2017.

[14] N. F. Palumbo, R. A. Blauwkamp, and J. M. Lloyd, "Basic principles of homing guidance," *Johns Hopkins APL Technical Digest*, vol. 29, no. 1, pp. 25–41, 2010.