Robotic Zippin' Airhockey
ENPH 479 - Team 2152

Zack Watkins, Evan Pham, Gregrory Reid, Gabriel Robinson-Leith

April 22, 2022

[3]

Project Sponsors:
Bernhard Zender
Miti Isbasescu
Dylan Gunn

Engineering Physics Project Lab
The University of British Columbia

Executive Summary

We were tasked by the UBC Engineering Physics Project Lab to develop a platform for a next generation AI and Automation course. Over the last two years we have designed and built an automated air hockey table for which students are able to implement their own agents to compete against human opponents.

Robotic zippin' air hockey - RZA - is an H-Bot gantry system capable of playing air hockey with human-like performance. We have designed a plug-and-play interface which separates the control into two distinct sections: high level control, and low level control. The high level control is responsible for choosing paths for the mallet to follow based on the current state of the game. The low level control is responsible for executing the chosen paths to high accuracy. This interface allows AI development students to not worry about the details of moving the gantry while developing their high level decision making systems.

The RZA allows students test and compare their respective agents on a real physical system. This will lead to an exciting final evaluation for students in this AI course as the performance of their decision making system is judged in a 1v1 competition format.

With a rudimentary high level decision making system, the RZA can play a human in air hockey. It can comfortably defend its net, and can score the occasional goal.

We believe a highly motivated 6 person team can further improve the H-Bot mechatronics and develop a powerful low level controller. We recommend an additional 4 person team undertake developing a simulation trained agent and test it on the physical table. This would demonstrate a proof of concept for the eventual ENPH 353 style course.

Contents

List of Figures

1 Introduction

In our experience, UBC Engineering Physics AI education occurs largely in two domains: simulated environments and basic hardware systems. Simulated environments allow for interesting tasks to be performed, but provide students with no experience dealing with the variability of the physical world. Basic hardware systems give students a more realistic experience, but typically are not capable of performing any particularly interesting tasks or operating with human-like competency.

This project was sponsored by the UBC Engineering Physics Project Lab with the goal of developing an autonomous air hockey robot capable of superhuman performance as a platform for AI education. The final deliverable will be a mechanical gantry system that can be controlled to play air hockey by an AI agent designed by an Engineering Physics student. Hereinafter we will refer to the entire system composed of the mechanical gantry system, and the computing architecture that reacts to the puck and plays air hockey as Robotic Zippin' Airhockey (RZA).

This report is the conclusion of a two-year project to develop an autonomous air hockey robot. We will introduce the reader to high level requirements for any robotic air hockey system that will lead into how we decided on a system breakdown for this project. We will then derive low level requirements for each subsystem that are informed by our overarching high level needs. We will then discuss our implementation of each subsystem and how we satisfied the necessary low level requirements. For a more in-depth discussion of the research and design phase of this project, please see our mid project review.[6]

2 Derivation of Requirements

To achieve superhuman performance the system needs to at a minimum move as fast as a human player could, with similar precision, and over the same spatial range.

From analysis of professional air hockey footage as well as accelerometer data we collected ourselves, it was found that a mallet acceleration of $44\frac{m}{s^2}$ and a top speed of 6 $\frac{m}{s}$ are required to match the performance of a human. The system must interact with the puck at any point in its zone, a 1x1 m surface.

To defeat a human opponent the system must make strategic decisions which are heavily dependent on the current gamestate. This gamestate includes but is not limited to the puck position and velocity, as well as both players' mallet positions. At a minimum, our system needs to track both its own mallet position and the state of the puck.

In addition, we are designing the RZA such that an ENPH 353 student can develop their own software decision making system that will interface with the mechanical gantry to play air hockey. Students should be able to easily design and test their decision making agents, leading us to the requirement that our system be plug-and-play in nature.

3 System Breakdown

The RZA needs a system that tracks the puck so that it can make high-level game play decisions. It also must have a mechatronic system to actuate the mallet, and a controller to realize the desired motions. The following is a broad outline of the low level requirements of each of these subsystems

derived from the above high level requirements.

3.1 Mechatronics Overview

The mechatronics must be able to move the mallet with high acceleration and velocity. This means that we need to minimize the inertia of whatever mechanical system we design to move the mallet while maximizing the power of the motors. To produce high torque, we will require a power supply that can source high currents.

High motor speeds will lead to large back emf. Motor inductance also creates voltages that oppose changes in current. When decelerating, this back emf and induced voltage combine to produce voltage spikes at the power supply. We will thus need a power system capable of safely handling these spikes.

In order to intercept and hit a puck, the mechanical system must be easy to control. We described this requirement in a previous report: "[T]he mallet must respond to motor movements consistently, independent of its position on the ice; a full rotation of the motor should correspond to the same movement whether the mallet is in the neutral zone or close to the net. We will refer to this criterion as positionally independent movement."[6]

3.2 High Level Control - Developing a Plug and Play Interface

Before getting into too much detail about how exactly we can create the rapid yet controlled motions we seek, we need to think about where the students' agents fit into the system. At the most basic, we could imagine students sending raw voltages to motors. But this would require each student seeking to implement an agent to spend time working out exactly what voltages are needed to attain their desired motion. If we want students to be able to focus on creating comprehensive agents, and learn about AI rather than dealing with control theory, a better approach is to abstract the details of the control from the students. If we are seeking to have students easily plug their agents into the system, then we should expose them to only the information that they need to make strategic decisions. This must at least include the position of their own mallet and the position of the puck.

Wanting to create a plug-and-play interface necessitates creating a low level controller (LLC) that executes the students' desired motion. The students would then develop a high-level controller (HLC, used interchangeably with agent) that decides on the motion. But having the control of the mallet split in this way now requires us to define an interface between these controllers.

To make an HLC - LLC interface we want to both minimize the amount of non-strategy work on the students' behalf while maintaining the ability to create highly flexible HLCs. At the very least the HLC will have to specify where and when to attempt to intercept the puck. But if the HLC only specifies a position, then it has limited influence over the direction of the shot. To be able to actually aim shots, the HLC needs to be able to set the velocity with which it impacts the puck. Finally, to allow for even more flexibility in the HLC, we can allow it to specify the acceleration with which it hits the puck. One can imagine having the HLC finish with positive acceleration to "follow through" shots, or to begin decelerating to get back into a defensive position. With that we can specify an interface between the HLC and LLC. The HLC must specify the desired mallet state, which is the time at which to collide with the puck, and the position, velocity and acceleration with which to do so.

3.3 Low Level Control

The task of the low level controller is two-fold. The LLC must first turn the desired final state of the mallet into a path for the mallet to follow. The LLC must then send commands to the motor controller to move the mallet along this desired path.

To achieve this first task, we need to generate a path that takes the mallet from its current state, to the final state set out by the HLC. The desired position, velocity, and acceleration combined with the respective initial conditions yields six degrees of freedom that uniquely define a fifth-order polynomial path between initial and final state. This is the desired path.

To move the mallet along this path, the LLC will need to send motor commands and account for deviations from the desired path. To accurately follow high velocity paths, we must retrieve motor positions and update motors commands at a rapid rate. This suggests the need for high speed encoders and a hardware device capable of performing this control loop at a sufficiently high frequency. In addition to providing the LLC with necessary feedback, the encoders tracking the mallet position also provide a portion of the gamestate to the HLC.

3.4 Puck Tracking

In addition to the mallet position, the high level controller also needs to know where the puck is so that it can intercept it. Tracking objects in real time is a common technical problem, and one typical, easy-to-implement solution is computer vision. The NHL, for example, is exploring using IR cameras to track their pucks.[5]

In order for a camera to track the fast moving puck in real time, we will need a high frame rate. The exact frame rate requirements depend on the expected puck speeds. To ensure the puck tracker is reliable, and to provide the HLC with complete information, the camera's field of view and placement should allow the entire table to be captured. Lastly, the accuracy of the tracking will depend on the resolution and the placement of the camera. Distant camera placement will result in lower positional resolution, but may be necessary to ensure the full play space is in frame. The balance between frame rate, field of view, and resolution will be important in selecting camera specifications.

3.5 Putting It All Together

To integrate the established systems, we need to consider their roles and the information they will need to exchange. The LLC is responsible for moving the mallet accurately along a high speed path. As such, it must read encoder feedback and send motor commands rapidly. This suggests using dedicated, efficient hardware for the LLC. These requirements are well suited to a microcontroller.

The HLC takes gamestate information and makes strategic decisions. It must be flexible with respect to its implementation to provide students freedom to design their HLC as they see fit. It can operate at a slower loop speed than the low level controller, but requires more flexibility, and should therefore be run on a separate computer.

These controllers must interface with each other and with peripheral hardware. A camera feed can be processed to extract the puck's position, which can then be fed into the HLC. The LLC will connect to motor controllers and encoders to monitor and control the position of the mallet. To bridge between these two systems, we will need a channel along which the HLC and LLC can communicate. This

will enable the HLC to send mallet paths to the LLC and retrieve the mallet's status. Our system architecture would then take the following form.

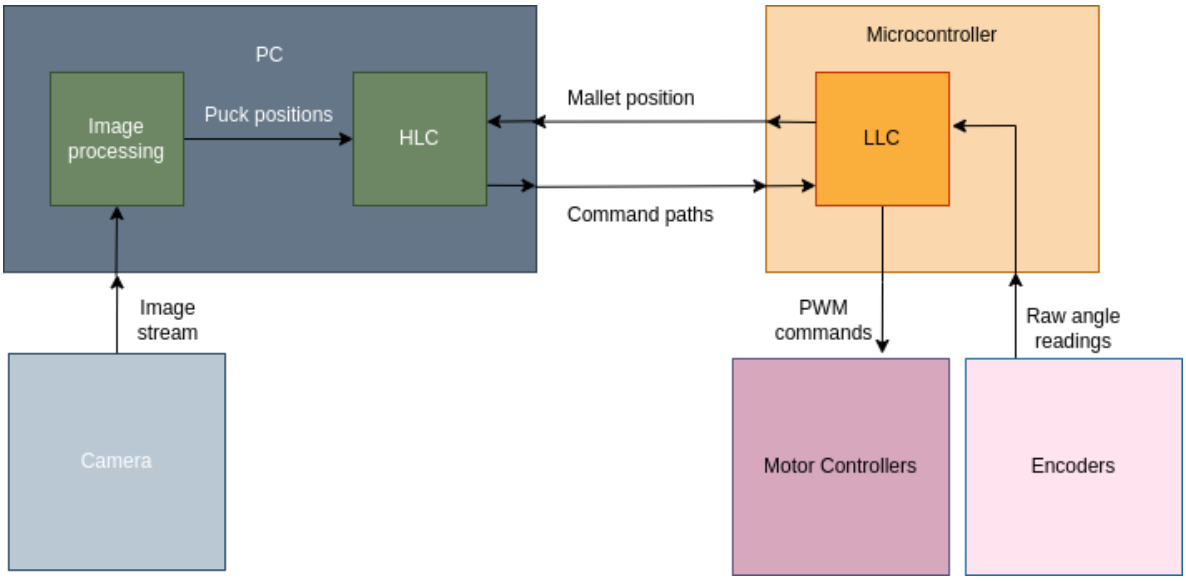

Figure 1: System architecture

Next we can go into more detail on each subsystem, referring back to these fundamentally motivated requirements within the architecture we have justified.

4 Design and Analysis of Mechatronics

Fundamentally, we want our system to allow student designed HLCs to be able to beat humans. This has two important consequences for the mechanical design:

- The system needs to be capable of high speeds, and high accelerations
- The system needs to accurately move along paths the students command

To these ends, our system needs to be lightweight, powerful, positionally independent and precise.

4.1 The H-Bot gantry

We selected an H-Bot gantry for actuating the mallet. This configuration allows us to move in both x and y directions without translating our motors - meaning our system can be at once lightweight and make use of powerful motors. Interested readers can refer to our ENPH 459 report[6] for a more in depth discussion of alternative designs and how we came to select an H-Bot gantry.

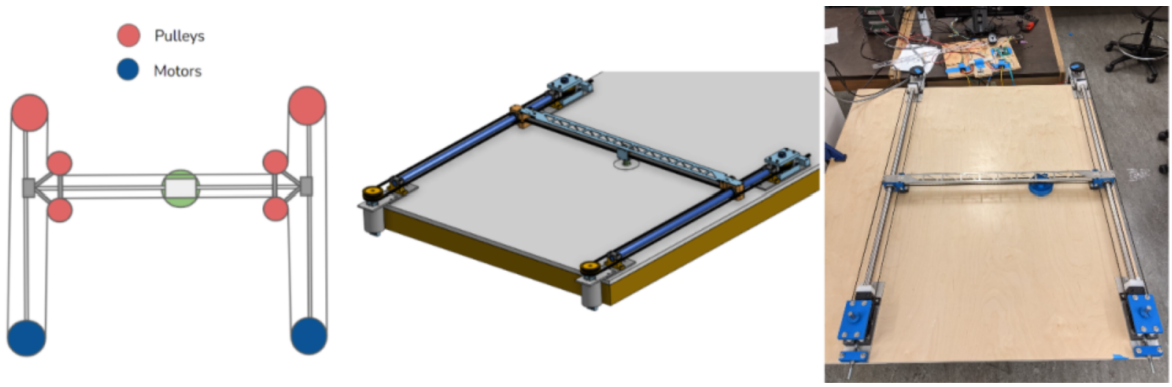

Figure 2: H-bot sketch (left), CAD model (center), and physical prototype (right)

Choosing an H-Bot configuration does not guarantee a lightweight and powerful system. We are left with the task of actually making it so. Furthermore, we need to make the system precise in its motion, and do this without greatly increasing the weight. Marrying the requirements and successfully evaluating trade-offs between them is the challenge of designing the mechatronics.

In addition to the gantry, we need to consider how we power the motors. While at a glance this seems like a separate problem, the physics of the gantry in a large way dictates our requirements for the power electronics. This happens in two ways:

- A heavier mechanical system necessitates higher power electronics to move sufficiently fast
- The rapid decelerations that a light mechanical system makes possible induce currents that our power system needs to manage appropriately

We will begin by describing the physics of the mechatronic system, as this elucidates the trade-offs and optimizations to be made. Once we establish a thorough understanding of the mechanisms at work in Section 4.2, we will discuss our design and where opportunities for ameliorations exist in Section 4.5.

4.2 H-Bot Physics

It proved useful throughout the design process to keep in mind the Lagrangian of the system. Doing so forces one to think about the exact ways in which parts are moving and how their individual inertias contribute to the overall inertia of the system.

First, let's sort out how the motor angles and x,y position of the mallet are related. As shown in the below graphic, rotating the motors in the same direction translates the mallet in x. Conversely, spinning the motors in opposite directions translates the mallet in y.

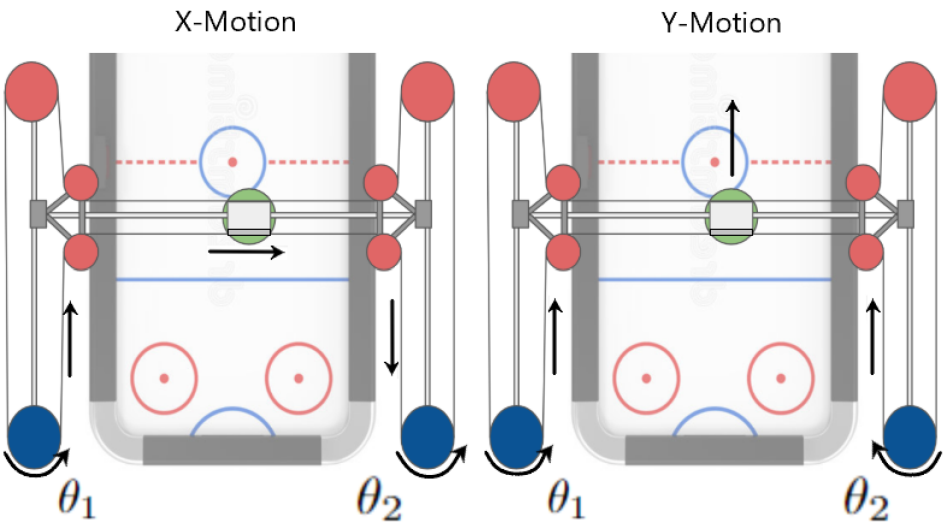

Figure 3: Demonstration of how motor movements translate into x and y mallet motions

Thus we have a simple linear map from θ_1, θ_2 to x, y (with our pulley radius being R):

$$x = R\frac{\theta_1 + \theta_2}{2}$$
$$y = R\frac{\theta_1 - \theta_2}{2}$$

$$\begin{pmatrix} x \\ y \end{pmatrix} = \frac{R}{2}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}\begin{pmatrix} \theta_1 \\ \theta_2 \end{pmatrix} \tag{1}$$

Since there is no change in potential energy involved in our system, we only need to consider the kinetic energy terms. There are four different terms to consider. These are the kinetic energy resulting from motion in x, y, θ_1 and θ_2

$$L = T = \frac{J_x}{2}\dot{x}^2 + \frac{J_y}{2}\dot{y}^2 + \frac{J_1}{2}\dot{\theta_1}^2 + \frac{J_2}{2}\dot{\theta_2}^2$$

To compute J_x, J_y, J_1 and J_2, one simply sums the inertia of objects that move along the corresponding axis. For example, the cross member translates in y, but never moves in x. Thus, it will contribute to J_y, but have no impact on J_x. We can further recognize the symmetry in our system; while it is not true that $J_x = J_y$, we certainly have $J_1 = J_2 = J_\theta$. We can then use (1) to write the Lagrangian as a function of either $\dot{\theta_1}, \dot{\theta_2}$ or $\dot{x}, \dot{y}$.

$$\begin{aligned} L &= \frac{J_x}{2}\dot{x}^2 + \frac{J_y}{2}\dot{y}^2 + \frac{R^2 J_1}{8}(\dot{x} + \dot{y})^2 + \frac{R^2 J_2}{8}(\dot{x} - \dot{y})^2 \\ &= \frac{J_x}{R^2}(\dot{\theta_1} + \dot{\theta_2})^2 + \frac{J_y}{R^2}(\dot{\theta_1} - \dot{\theta_2})^2 + \frac{J_1}{2}\dot{\theta_1}^2 + \frac{J_2}{2}\dot{\theta_2}^2 \end{aligned}$$

Next, we find the Euler-Lagrange equations, and after adding terms to account for viscous friction losses we find:

$$\tau = J\ddot{\mathbf{x}} + B\dot{\mathbf{x}} \tag{2}$$

Where τ is the vector containing the torque at each motor, B is a velocity-dependent-loss matrix, and J is the inertia matrix:

$$J_{x,y} = \begin{pmatrix} J_x & \frac{2J_\theta}{R^2} \\ \frac{2J_\theta}{R^2} & J_y \end{pmatrix} \tag{3}$$

Or, using θ_1, θ_2:

$$J_{\theta_1,\theta_2} = \begin{pmatrix} \frac{R^2}{4}(J_x + J_y) + J_\theta & \frac{R^2}{4}(J_x - J_y) \\ \frac{R^2}{4}(J_x - J_y) & \frac{R^2}{4}(J_x + J_y) + J_\theta \end{pmatrix} \tag{4}$$

The symmetry of J in the θ_1, θ_2 co-ordinate system makes these co-ordinates particularly useful. To build some intuition for how J relates back to our physical H-Bot gantry, we will consider motion in two basic cases: pure x motion and pure y motion.

When moving in x only, we have that $\dot{\theta}_1 = \dot{\theta}_2$, so the effective moment of inertia per motor reduces to $J = \frac{R^2}{2}J_x + J_\theta$. Moving only in y gives an effective moment of inertia per motor of $J = \frac{R^2}{2}J_y + J_\theta$. You can check these values by multiplying (4) by $\boldsymbol{\theta} = (\theta_1, \theta_1)^T$ (for x motion) or $\boldsymbol{\theta} = (\theta_1, -\theta_1)^T$ (for y motion). Figure 4 shows the contribution of each physical parts to these effective inertias.

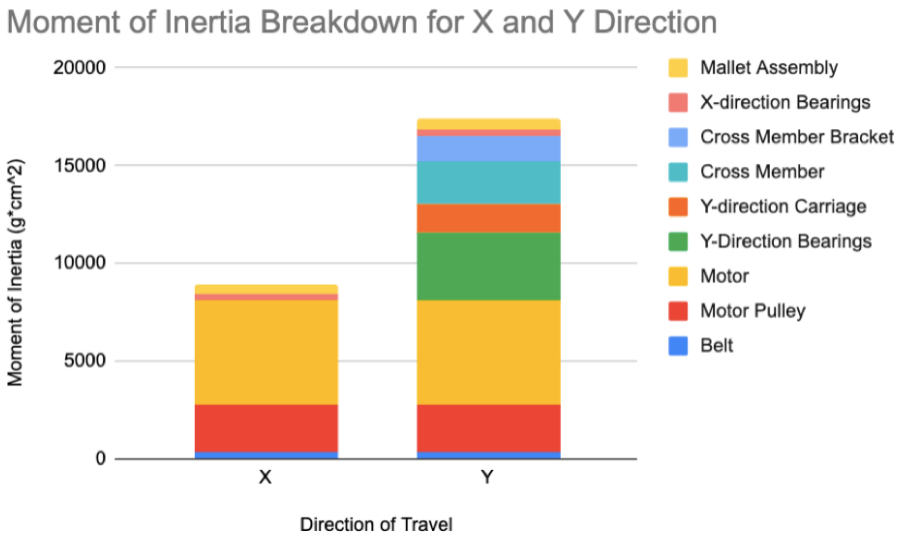

Figure 4: Inertia breakdown for both x and y motion

The inertia forms one of the two terms in our modified Euler-Lagrange equation (2). We also now have to consider the viscous friction $B\dot{\mathbf{x}}$. We are able to exploit the symmetry of our system to know that B must be a symmetric matrix:

$$B_{\theta_1,\theta_2} = \begin{pmatrix} b_l & b_s \\ b_s & b_l \end{pmatrix}$$

b_l and b_s are labeled the large and small damping terms respectively, as we expect $b_l > b_s$. More reasoning as to why this matrix is of this form can be found in Appendix B.6.

We now have some understanding of how motor torques create motion. But, as with any model, it is important to understand its limitations, and the regimes in which the model does not apply. For the mechanical system, the largest effect that our model does not capture is static-friction. Attempting to model static-friction would make the ODEs relating τ to θ non-linear. This effect is important at slow speeds, where the viscous friction term is small. When moving at high speeds, the viscous friction term begins to dominate and static friction and non-viscous friction can be neglected.

Limitations of this model aside, this is still an incomplete picture of the system. Our next challenge is to relate voltages applied to the motors to their torque output.

4.3 Modeling Motor Torque Output

We already have a way to relate motor torque to mallet motion. However, we can't directly control the torque the motors will produce - we can only control the voltages applied to the motors. Practically, this is done by means of pulse width modulation (PWM), but the inertia of the system acts as a low-pass filter. Thus, we can simply treat a time varying PWM signal as though it were the equivalent fractional voltage. We need not concern ourselves with any transient behaviour caused by the nature of the PWM signal.

To model the motors we will treat them as individual armature circuit.

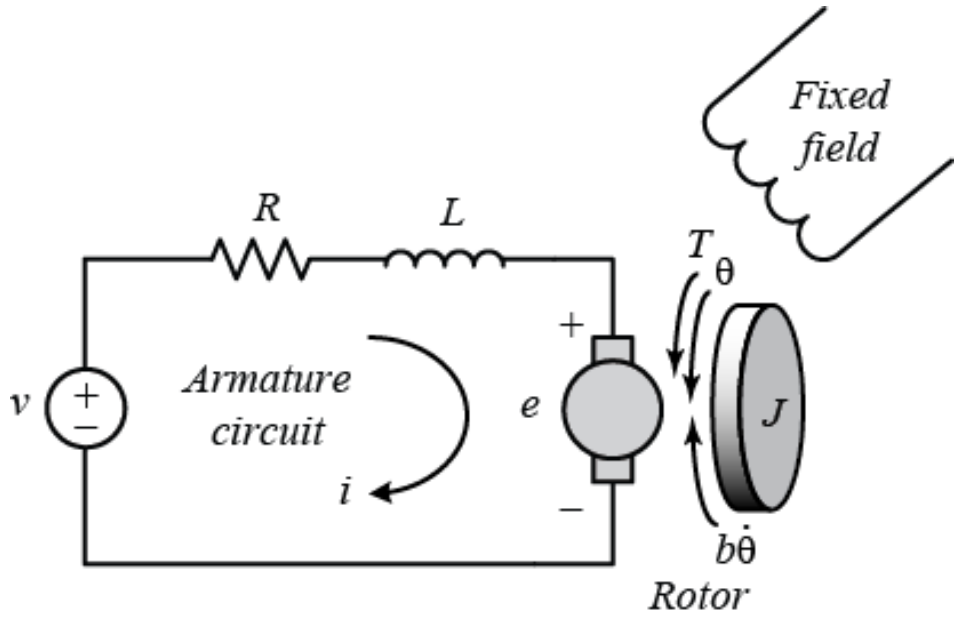

Figure 5: This is an armature motor model.[4] It represents a motor as a series circuit composed of a resistor, an inductor, and an angular velocity dependent voltage source (the back EMF).

It is generally taken that motor torque is proportional to the current i, by constant of proportionality

K_t

$$\tau = K_t i$$

Likewise, the electromotive force produced by spinning the armature in a constant magnetic field is proportional to the angular velocity of the rotor $\dot{\theta}$, by constant of proportionality K_e. In SI units K_e and K_t have the same value.[4] We will work in SI so take $K = K_t = K_e$.

Using Kirchhoff's voltage law, we have that:

$$v = Ri + L\dot{i} + K\dot{\theta} \tag{5}$$

This same equation holds for both of our identical motors. Written out as a vector $\mathbf{V} = (v_1, v_2)^T$, and $\mathbf{i} = (i_1, i_2)^T$ we have:

$$\mathbf{V} = R\mathbf{i} + L\dot{\mathbf{i}} + K\dot{\boldsymbol{\theta}} \tag{6}$$

$$K\mathbf{i} = J\ddot{\boldsymbol{\theta}} + B\dot{\boldsymbol{\theta}} \tag{7}$$

Rearranging so that we remove i gives:

$$\mathbf{V} = \frac{LJ}{K}\dddot{\boldsymbol{\theta}} + \frac{RJ + LB}{K}\ddot{\boldsymbol{\theta}} + \frac{RB + K^2}{K}\dot{\boldsymbol{\theta}} \tag{8}$$

Since L, K and R are the same between motors, they can just be written as scalar. B and J are both matrices.

Let us now consider what we have achieved, in the context of our project. Between (6), (7), (8) we can now relate the three most important physical quantities involved in our project: the current through the motors ($\mathbf{i}$), the applied voltage ($\mathbf{V}$), and the motion of the mallet ($\boldsymbol{\theta}$, or x, y).

Again, it is important to consider the limitations of our model. Aside from the effects of friction we discussed in Section 4.2, the largest assumption in our model concerns the ideal power supply. On large current draws, we saw the power supply voltage drop. Likewise, when current enters the positive terminal of the power supply, we saw an increase in power supply voltage. Neither of these effects are captured by our ideal model.

4.4 Power System

4.4.1 Back EMF and changing directions

Let us take a moment to consider the implications of (6) and (7). When actually playing air-hockey we are likely to be rapidly changing directions. A change in direction obviously requires a change in voltage. Let's consider the most extreme case: moving at top speed in one direction, and then suddenly switching the polarity of the applied voltage. Firstly, let's consider the state of the system just before we switch polarity. For simplicity, let's consider motion only in 1-D. At top speed, we are no longer

accelerating, so $\ddot{\theta} = 0$. Likewise, the current will have reached steady state so $\dot{i} = 0$. For an applied voltage v, we then have steady state current i_{ss} and angular velocity $\dot{\theta}_{ss}$ given by:

$$\begin{aligned} v &= Ri_{ss} + K\dot{\theta}_{ss} \\ Ki_{ss} &= B\dot{\theta}_{ss} \\ i_{ss} &= \frac{vB}{RB + K^2} \\ \dot{\theta}_{ss} &= \frac{vK}{RB + K^2} \end{aligned} \tag{9}$$

We can see from (9) that the system's top speed is linearly related to the applied voltage. Because we have an inductive circuit, the current isn't able to change instantaneously. Likewise, the motors must move with physically possible accelerations - meaning their angular velocity also can't change instantly. So at the *instant* we change the polarity of applied voltage, we will still be spinning at the same steady-state velocity, and the current will be the same steady-state current. When we take $V \to -V$, we see from (6) that:

$$\begin{aligned} -V &= Ri_{ss} + K\dot{\theta}_{ss} + L\dot{i} \\ -V &= V + L\dot{i} \\ \dot{i} &= \frac{-2v}{L} \end{aligned}$$

The inductance of the motor will begin to decrease the currents, and as the motor slows down the back emf will fall off. Nonetheless, for a significant period of time, we will have current entering the positive terminal of the power supply. For a real, physical power supply, such currents will cause behaviour our current model is unable to capture, and may even damage the power supply. Thus, when designing the power supply circuitry, we need to be able to account for these currents.

4.4.2 Power Supply and Dump Circuit

We will now contextualize these theoretical concerns by reviewing the behaviour of our physical, non-ideal power supply. The motor manufacturer (Ampflow) had a suite of recommended power supplies with a range of output power. Given our high current draw requirements, we opted for their most powerful supply (1000 W) with parallelizability in case we needed additional power. These power supplies were rated to be stable within $\pm 0.5\%$ of their output voltage under load, but subject to a large current draw this was not the case. When we attempted rapid acceleration, the voltage would drop below the 24 V set point. As a result, the relationship between the motor command sent to the controllers and the resulting acceleration was non-linear.

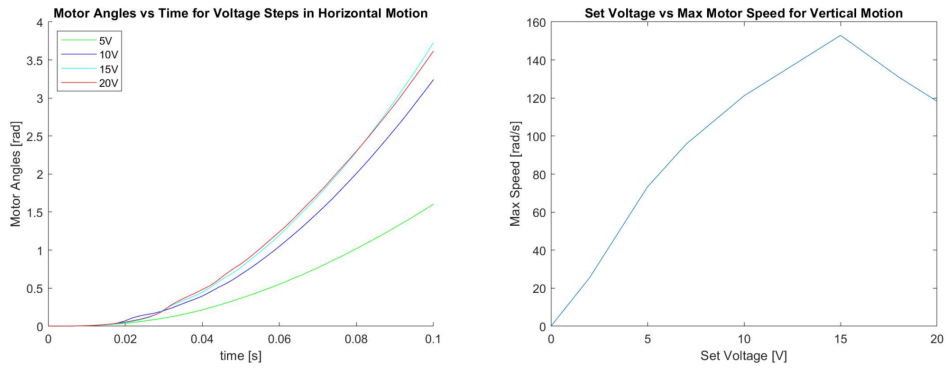

Figure 6: Demonstration of nonlinear voltage response

In Figure 6, the left plot shows motor angle vs time data resulting from step voltages applied to the motors. Generally, increasing the step voltage increases the acceleration and resulting steady state speed. This holds true up until around 15 V, at which point we see significant nonlinearity. When we request motor voltages of 20 V, we see the acceleration decrease. This is because the voltages are sent as PWM duty cycles, and with high current draw, the power supply voltage drops, resulting in a lower voltage seen at the motor. This nonlinearity reduces the effectiveness of our controller, as increasing the motor effort may actually reduce the resulting torque. The right plot more explicitly demonstrates this nonlinearity. The set voltage and the steady state angular velocity are expected to be linearly related, as discussed in the modelling section. Consistent with observations of the left graph, we see the steady state speed drop off after 15 V. Our peak speed at 15 V is around 155 rad/s, which translates to a mallet speed of 5.5 $\frac{m}{s}$, 90% of the human-level target of 6 $\frac{m}{s}$. Despite the voltage limitations, the H-Bot is nearly meeting specifications.

In addition to the troubles with rapid acceleration, deceleration has to combat back emf and motor induction, as discussed in Section 4.4.1. When applying a braking voltage to a fast moving motor, the power supply can see voltage spikes, raising the positive voltage rail above the power supply's output 24 V. Because this can damage the power supply, it has built-in safeties which turns itself off if it sees a sustained voltage spike or dip.

To prevent power cutouts due to voltage spikes, we have a dump circuit which passes back-flowing current through a power resistor, protecting the power supply. The motor controllers have a signalling pin that goes high when excessive backdriving occurs. This pin can be used to trigger a MOSFET, which activates the dump circuit.

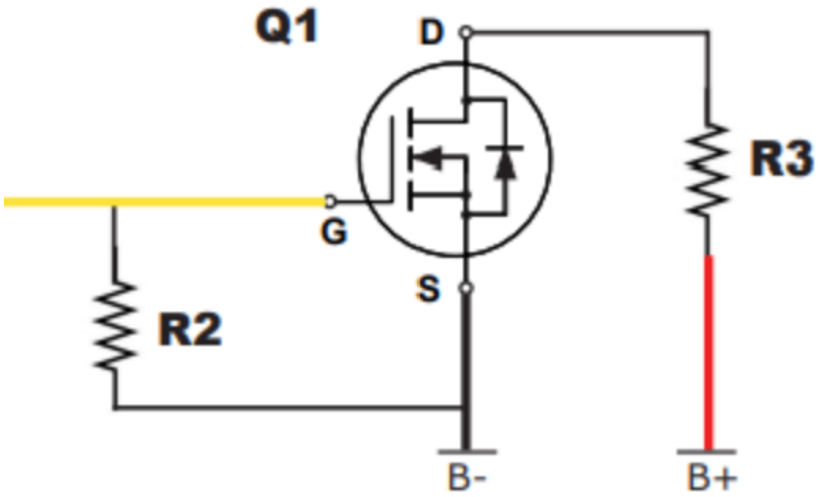

Figure 7: Schematic of basic current dump circuit[2]

This unfortunately introduces further nonlinearity. When the MOSFET is turned on and the power resistor is in parallel with the motor, the circuit physically changes. Therefore our system model is no longer accurate. The assumptions made about the circuitry in modelling will not hold, and as a result, system output behaviour will deviate from the predicted path.

4.4.3 Motor Controller

Due to the high power nature of our system, the motor controllers need to be able to facilitate large currents and rapid switching. From modelling our system we found rough estimates for the maximum currents needed for aggressive play to be around 60 amps per motor at 24 volts.

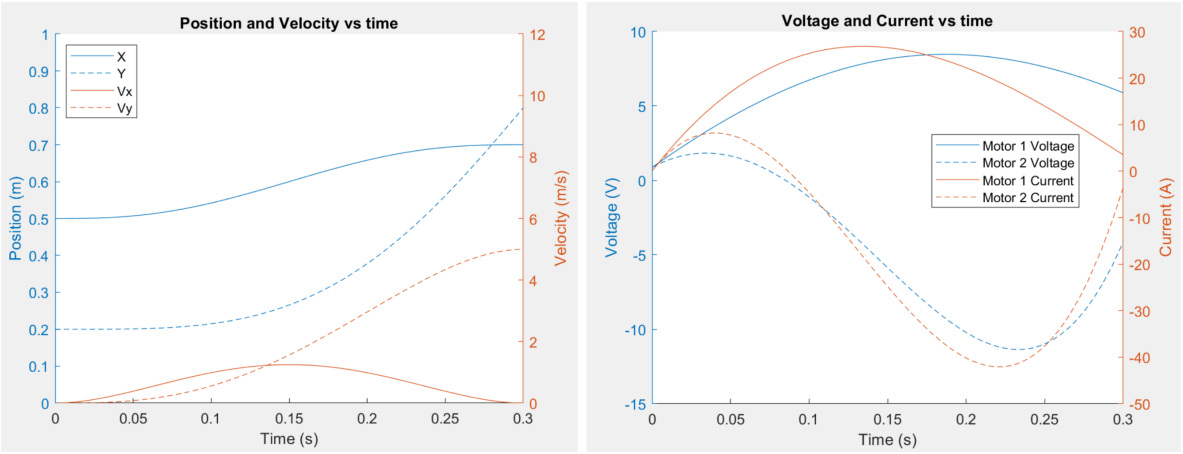

Figure 8: Plots from our 459 report[6] showing predicted motor voltages and currents for a shooting motion at human-level speed and acceleration. More aggressive paths can demand higher voltages and draw larger currents, but these provide a good example of the typical magnitudes.

Another requirement is that the motor controller we choose must be compatible with the microcontroller we use for the LLC.

Using these as requirements for possible motor controllers we arrived at the Roboclaw Solo 60 DC motor controllers. They can handle 60 amps continuous and up to 100 amps peak from 0-34 volts. They can go from full current forward to full current reverse without damaging their circuitry which is

important for fast paced air hockey where one moves erratically in various directions. Another great feature of these controllers is that they can communicate over a packet serial line, widely available to many microcontrollers. To see more about the specific settings and nuances of these controllers please refer to Appendix E.2.

4.5 Mechanical Design Discussion

Most of the practical mechanical design concerns were addressed in our previous report.[6] Our discussion of the physics (see. 4.2) of the H-Bot allowed us to correctly design the majority of the parts. However, there were some concerns that were not sufficiently addressed in our first iteration.

The largest such concern was the deflection of the x-motion shaft. When moving the mallet in x, there is a moment imparted on the cross-rod. Without adequate stiffness, or with excessive mechanical play, the y-motion carriages will move relative to one another. If the y-motion carriages move relative to one another, then the conversion between θ_1, θ_2 and x, y (1) is no longer accurate. For this reason, maintaining the stiffness of the gantry is crucial.

To achieve the required stiffness, we added a cross-member between the two y-motion carriages. This significantly increased the stiffness of the entire gantry assembly.

5 Microcontroller and Low Level Control

Now we will discuss how this mechanism will be controlled. Remember, the purpose of the low level controller is to operate the motors such that the mallet effectively follows polynomial paths. A successful low level controller handles the details of mallet movements so that students developing high level controllers can focus on the important strategic elements. As a result, in designing the LLC, we will need an understanding of the input-output behaviour of the mechatronic system. To ensure the desired motion is being followed closely, the low level controller will also need to monitor the state of the mallet, and will thus require feedback in some form.

5.1 The Hardware - BluePill and Encoders

We decided to implement the low level controller on a dedicated microcontroller in order to achieve the high loop speeds required to accurately track the position of the mallet and communicate with the motor controllers to correct any accumulated error. The following argument illustrates the need for a high speed microcontroller.

In order to track the position of the mallet, we placed 14-bit Hall Effect encoders on the motor shafts that can determine the angular position of the motors with a theoretical resolution of 0.02 degrees. These encoders only keep track of the current angle between 0-360 degrees, and it is up to the microcontroller to keep track of the accumulated angle. Our high-level requirements outlined above determine a maximum angular velocity 27 revolutions/sec based on our maximum mallet velocity and our motor pulley radius. For a discussion of optimal pulley radius see [6].

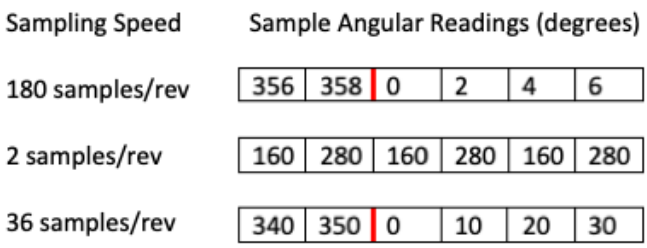

Figure 9: Different sampling speeds for a constant angular velocity reading, red lines indicate well defined zero crossings, it is difficult or impossible to determine at 2 samples/rev whether motor is oscillating or rotating at fixed angular velocity

If we wished to take advantage of the full encoder resolution, we would need a microcontroller operating at $(27\ \frac{\text{rev}}{\text{s}})(2^{14}\frac{\text{counts}}{\text{rev}}) = 442.4$ kHz. This is an upper bound on the useful microcontroller loop frequency. A lower bound is to sample the encoders once per revolution (27 Hz). However, if we receive two subsequent readings of, say, 30 degrees, there is no way to tell whether a full revolution has occurred in between readings, or the motors are sitting still. Figure 9 outlines the relationship between sampling speed and the challenge of determining the accumulated angle.

Through testing, we have found that a sampling rate of 36 samples/revolution and therefore about 1000 Hz results in robust angular reading measurements. Anything lower and our system is not resilient to missed readings and we begin to see accumulated error in our mallet position.

This loop speed requirement is what led us to use a microcontroller for the LLC. Separating the HLC and LLC in this manner necessitates communication between the microcontroller and the HLC running on an external computer.

In practice, we opted for a BluePill, as it was readily available in the Project Lab and it had the ability to communicate with our HLC and the motor controllers through easy to configure serial ports outlined in Appendix E.2. The I/O ports made the BluePill suitable to rapid prototyping, debugging, and additional sensor monitoring.

5.2 Feedback Control

The interface between the high and low level controllers has been defined as five degree polynomial paths that are decided on by the high level controller, and executed by the low level controller. To execute a path, the low level controller must send PWM values to the motor controllers that will move the mallet along the desired path.

The simplest and first method we used to follow paths was with PID control with closed loop feedback from the motor encoders. Each time angle readings from the encoders is received, the LLC computes the error between the current position and the current desired position. Since our BluePill is operating at approximately 1000Hz, the LLC will recompute and update motor effort values every 1 ms, stitching together the required five degree polynomial paths.

Proportional and derivative error terms are used to compute effort values to send to the motor controllers. The PD terms were computed using the no-overshoot Ziegler-Nichols method.[9] These terms were then manually adjusted during subsequent tests with the goal of reducing jerky motions.

We decided not to use an integral term as we are dealing with a constantly changing setpoint control problem, where the desired state of the mallet may change every 100ms. There is not enough time in

a fast-paced variable setpoint control problem for an integral term to be applicable.

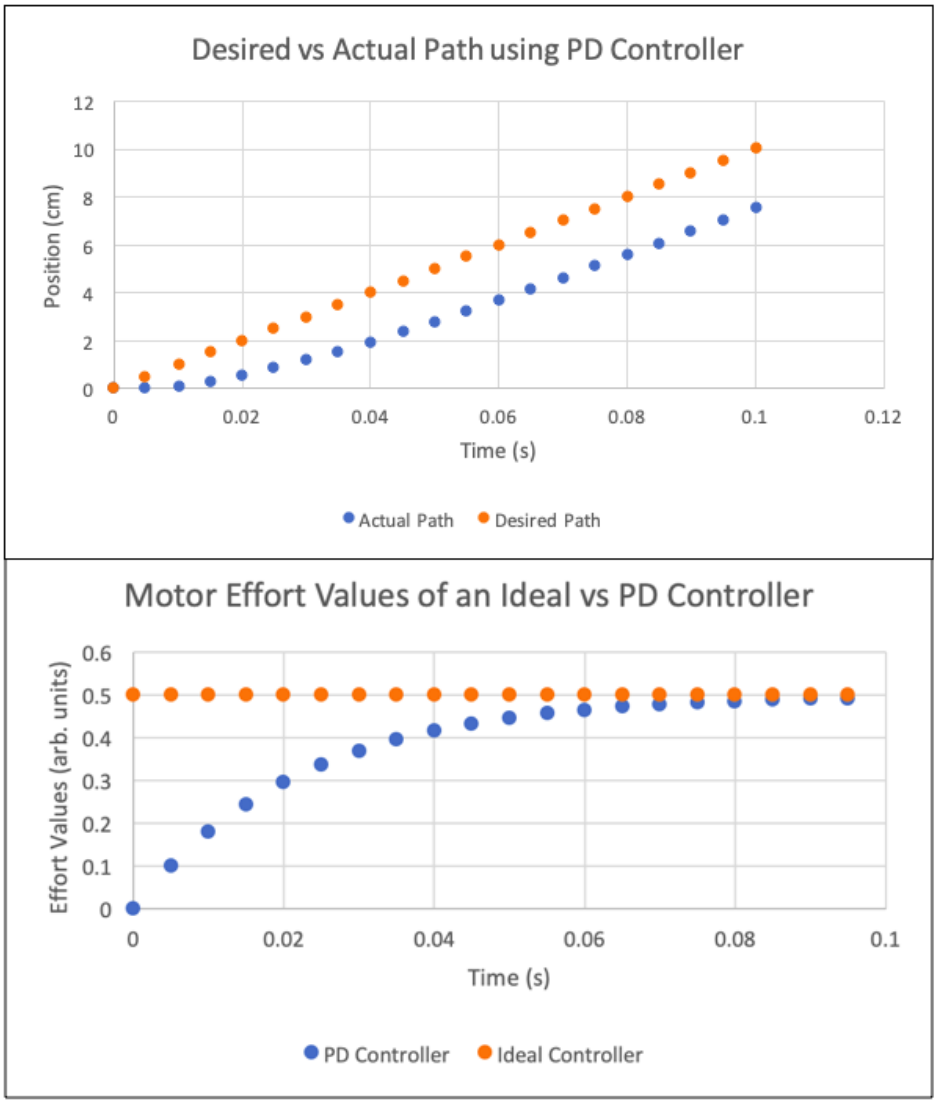

Figure 10: An example of a PD controller lagging a desired path. The effort is error driven, and will tend to lag behind at the path startup.

While PD has been sufficient for basic air hockey play, an error-based control system will always introduce a delay into gameplay as the LLC must wait for error to accumulate before actuating the motors in response. For this reason, we have begun developing a model based controller that can follow paths in real time.

5.3 Feed-forward Control

To improve on standard PD control, one common practice is to include a feed-forward term that predicts the required effort. This is typically done by developing a model of the system, and using that model to predict the necessary inputs. In Sections 4.2 and 4.3, we have already outlined a relationship between a path $\boldsymbol{\theta}(t)$, and the associated voltage signal $\mathbf{V}(t)$. Now, if the model perfectly described the system, and we had perfect knowledge of all the relevant underlying physical quantities, we could predict the exact voltages we would need to execute a given $\boldsymbol{\theta}(t)$ path. Unfortunately, our model is imperfect, and we don't have perfect knowledge of the physical quantities. To make a practical

feed-forward controller, we must include a standard PD controller to account for whatever errors we have in our modeling. We will call our model (see (8)) $\mathbf{H}(\dot{\boldsymbol{\theta}}, \ddot{\boldsymbol{\theta}}, \dddot{\boldsymbol{\theta}})$. Thus our form for the feed-forward controller is, for desired motion $\boldsymbol{\theta}$, and actual measured motion $\boldsymbol{\theta}_a$:

$$\mathbf{V} = \mathbf{H}(\dot{\boldsymbol{\theta}}, \ddot{\boldsymbol{\theta}}, \dddot{\boldsymbol{\theta}}) + K_p(\boldsymbol{\theta}_a - \boldsymbol{\theta}) + K_d(\dot{\boldsymbol{\theta}}_a - \dot{\boldsymbol{\theta}}) \tag{10}$$

A controller of this form doesn't have to wait for error to grow before imputing voltages, and thereby obviates the delay problems of a PD controller. To practically implement this feed-forward controller, we need to go from our our symbolic model, to real values for the relevant constants.

5.3.1 Determining Model Constants

Looking at (8), there are 3 terms, each with their own 2x2 matrix coefficient. Since both J and B are symmetric, there are a total of 6 independent coefficients that uniquely specify the model.

$$\mathbf{V} = \begin{pmatrix} a_1 & a_2 & a_3 & b_1 & b_2 & b_3 \\ b_1 & b_2 & b_3 & a_1 & a_2 & a_3 \end{pmatrix} \begin{pmatrix} \dddot{\theta}_1 \\ \ddot{\theta}_1 \\ \dot{\theta}_1 \\ \dddot{\theta}_2 \\ \ddot{\theta}_2 \\ \dot{\theta}_2 \end{pmatrix}$$

Each a and b can be written as a combination of some inertia (J), motor constant (K), damping constant (B), resistance (R) and inductance (L). See Appendix B.6 for a breakdown of these coefficients in terms of physical constants. If you were able to measure each of these, and then combine them as per the equations in Appendix B.6, you would have completely specified the model. However, many of these quantities are difficult to measure, thereby making this approach infeasible. Furthermore, the model only depends on specific combinations of L, J, R, B and K. We don't care what L is on its own, but we certainly care about the value of $\frac{LJ}{K}$ (see (8)). We are saved from having to measure all these quantities independently by the fact that our model is linear and time-invariant. This means that $\mathbf{H}$ is uniquely defined by its step response.

Thus, to extract all the as and bs we will give a step-input in voltage, and adjust as and bs to curve fit. To determine the form of the function we need to fit, we have to find an analytical solution to the *inverse* kinematics. That is, we need to solve the ODE from (8) for $\boldsymbol{\theta}$. Before doing so, let's simplify the algebra by only considering pure x motion, and pure y motion. Using symmetry, when moving in only x we have that $\dot{\theta}_1 = \dot{\theta}_2$ and $V_1 = V_2$. Likewise, when moving only in y we have $\dot{\theta}_1 = -\dot{\theta}_2$ and $V_1 = -V_2$. By letting $d_i = a_i + b_i$ for the first case, and $d_i = a_i - b_i$ for the second case, we reduce our system of ODEs to a single ODE. After we find the d_i values for each case, we can easily solve for every a_i and b_i.

The Green's function for the 1-d version of (8) is given by:

$$H^{-1}(t) = \frac{1}{d_3}\left[1 - \exp\left(-\frac{d_2}{2d_1}t\right)\left(\cosh\left(t\sqrt{(\frac{d_2}{2d_1})^2 - \frac{d_3}{d_1}}\right) + \frac{\sinh\left(t\sqrt{(\frac{d_2}{2d_1})^2 - \frac{d_3}{d_1}}\right)}{\sqrt{(\frac{d_2}{2d_1})^2 - \frac{d_3}{d_1}}}\right)\right] \tag{11}$$

The derivation of (11) can be found in Appendix B.6. From this Green's function, we can easily write an integral representation of θ. For any input V, and assuming we start at rest, we can write:

$$\theta(t) = \int_0^t V(t-\tau)H^{-1}(\tau)d\tau \tag{12}$$

This is the function we need to fit to. Again, doing this for both x and y motion lets us solve for all the coefficients in the models.

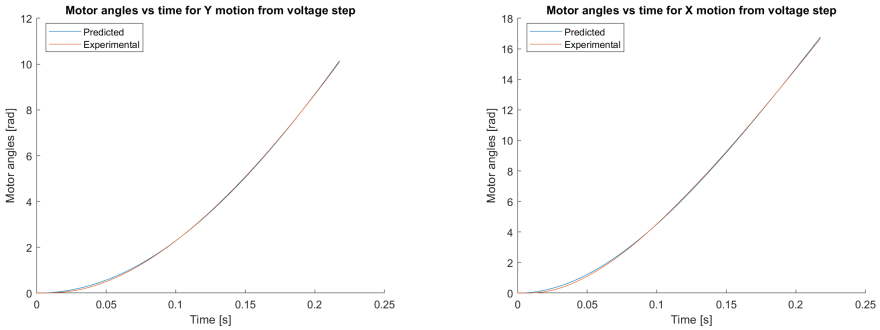

Figure 11: Example model fits to x and y motion

5.3.2 Feed-forward Control Issues

Issues with the feed-forward controller are tantamount to issues with the underlying model. We found that certain motions our model preformed well - meaning that the voltages it predicted actually moved the mallet more or less along the desired curve. However, we were unable to create a model that performed satisfactorily for generic polynomial paths.

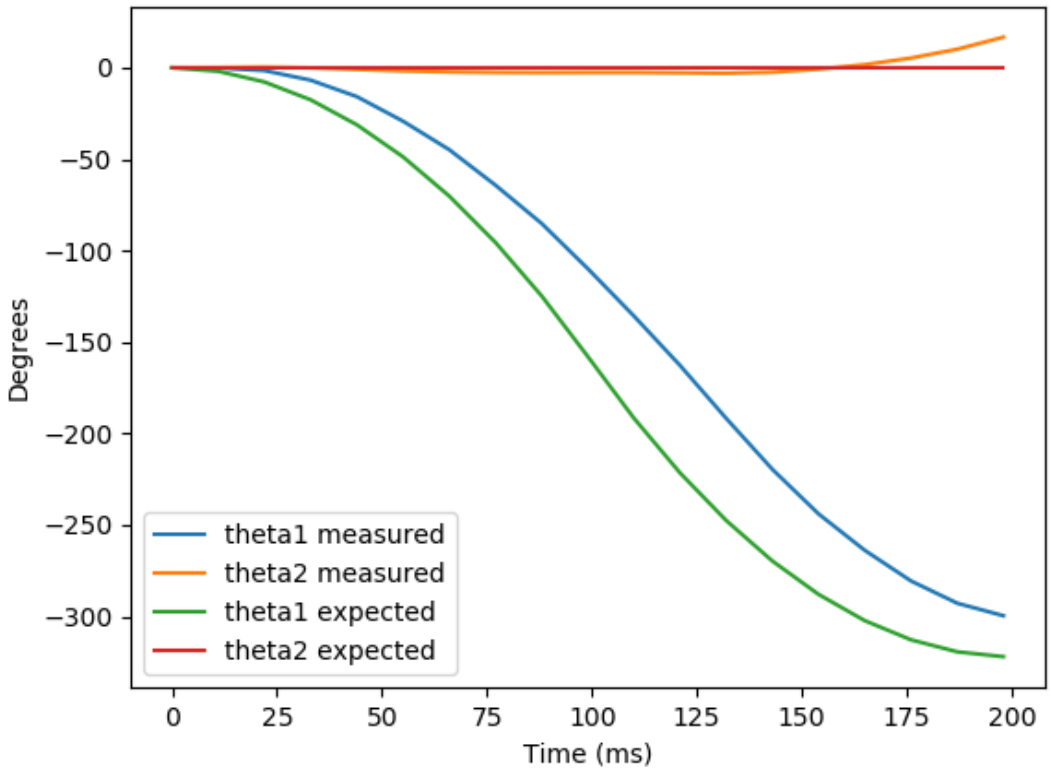

Figure 12: Desired v.s. actual θ_1 and θ_2, when voltage predicted by H are given to the motors. This is an example of a satisfactory behaviour, and not representative of the general case.

This was most pronounced on paths that saw the applied voltage leave the linear regime of our power supply (i.e. above 15 V, see Figure 6). We see two ways to remedy this. Firstly, we could update our model to reflect the non-linear behaviour. This approach would see (8) become an non-linear system of ODEs, and the math involved to determine coefficients becomes much more challenging. Secondly, one could imaging reworking the power-supply system to produce more linear behaviour. This approach would require more involved electrical design, but it is our opinion such work would prove worthwhile.

6 Camera, PC and High Level Control

6.1 Tracking the Puck

With a functioning low level controller, we must now establish the framework within which the high level controller operates. As previously stated, this control loop can occur at a lower rate than the low level control and should provide a flexible environment for students to develop their HLCs. Because of this, we opted to run the HLC on a PC rather than a microcontroller.

To make tactical decisions, it is crucial that the HLC be given accurate gamestate information. The motor encoder readings provide precise mallet positions, but we are still in need of a puck tracking system.

A camera mounted above the air hockey table can accurately track the position of the puck. In order

to accomplish this task, any camera that we source must have a sufficiently high frame rate (FPS) to track rapid puck movements, and a sufficient field of view (FOV)to see the whole table.

The FPS must be high enough to spot an incoming puck headed towards the RZA's goal line, and allow ample time for the RZA to respond. Through testing, we have found path times below 150 ms to be unstable and difficult to control. While further work on the LLC may increase the response time of the gantry, there will always be a fundamental lower limit on stable and achievable path times.

Using a 150 ms response time as our baseline, we may determine the required FPS needed to intercept an incoming shot. To accurately determine the velocity of the puck, we need at a minimum two separate frames of the puck moving, although with three frames we can determine the puck velocity with better certainty. The following is a broad derivation of camera FPS.

A puck moving at 10 m/s will cross the 2 m play surface in 200 ms. We need 150 ms to execute a path, leaving only 50 ms to detect the puck is coming at our goal and determine a defensive manoeuvre. 3 frames over 50 ms yields a minimum frame rate of 60 FPS assuming no latency in the communication between the camera, computer, and microcontroller.

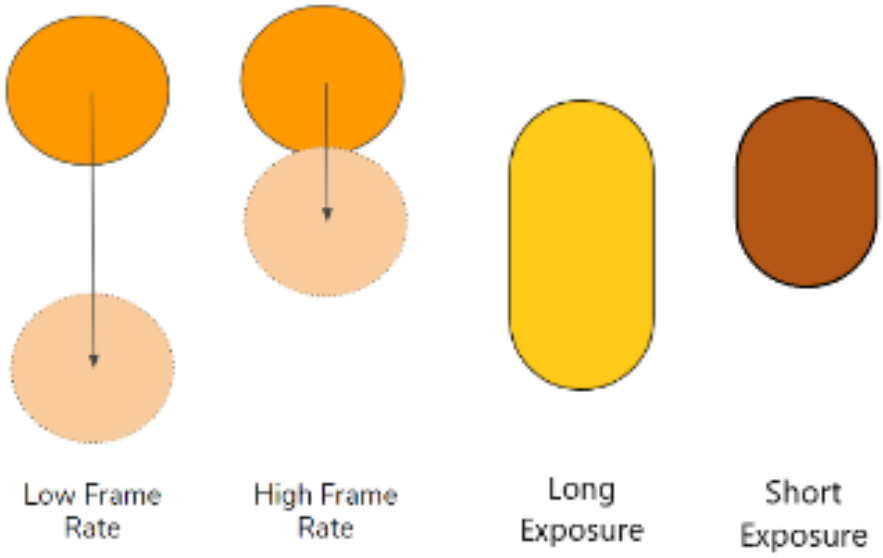

Figure 13: Consequences of changes in exposure and frame rate. Lower frame rates will result in more puck translation between frames, increasing uncertainty in puck position. Long exposures will have more motion blur, resulting in oblong puck images which blend in color with the background. Short exposures reduce the total light in the image, and thus reduce contrast, but provide less distorted puck streaks.

We also required a camera with a FOV covering the entire table while being mounted no higher than 2 m above the table due to height restrictions in our workspace. We therefore decided on a 90 FPS RasPi HQ camera module with an 80 degree horizontal FOV lens. The wide angle lens introduced distortions that will need to be dealt with in the image processing.

This RasPi camera only works through a RaspberryPi board. This board lacked the computing power to process the images at the necessary 90 Hz. We therefore used a UDP socket to send the raw frame data from the Raspberry Pi to the PC running the HLC.

The image processing originally used an HSV colour mask to search each frame for the red pixels of the puck. This method was unreliable under different lighting conditions and was susceptible to picking up skin tones in the HSV colour mask. We decided instead to mount an LED on the puck, and search for

points of high colour saturation in each frame to determine the position of the puck. In order to reduce motion blur, the exposure on the RasPi camera was lowered to 600 microseconds (see fig 13).

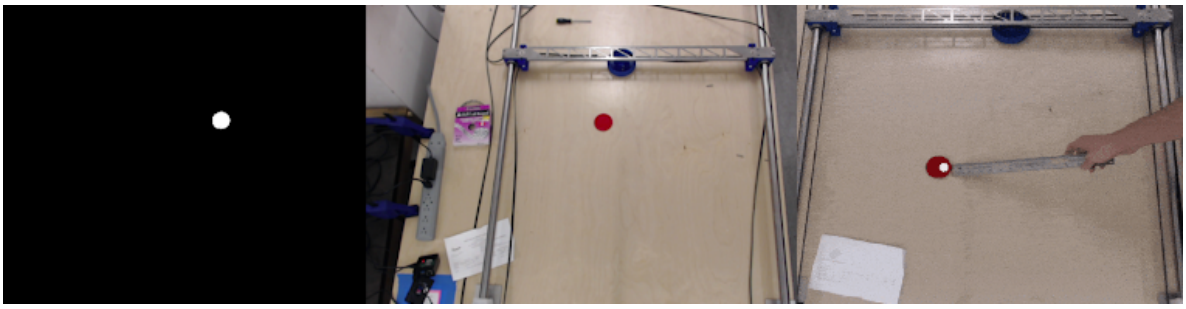

Figure 14: Left: RasPi camera image of red puck. Center: Binary image from HSV filter. Right: HSV tracking of standard red puck is thrown off by a hand in frame.

The task of the image processor was to pick out the LED, convert the pixel value into a position, and make the position available to the HLC all at a rate of 90 Hz. There are many options for improving the accuracy of the puck tracker (using multiple cameras, increasing camera resolution, applying various image transforms etc.), however each of these options will tend to increase the amount of time required to process the image stream and extract puck position. In order to ensure we are tracking in real time, the image processing loop must at least keep up with the camera frame rate. Increased processing time will also cause the HLC to receive puck positions with a larger delay, adding difficulties for students implementing HLCs. Here we see a trade-off between puck tracking accuracy and efficiency, with a minimum efficiency requirement set by the frame rate.

6.2 Sketch of the HLC

To solidify where the HLC fits into the system, it is helpful to go over a sketch of how the information flowing into the HLC might become a desired path sent to the LLC.

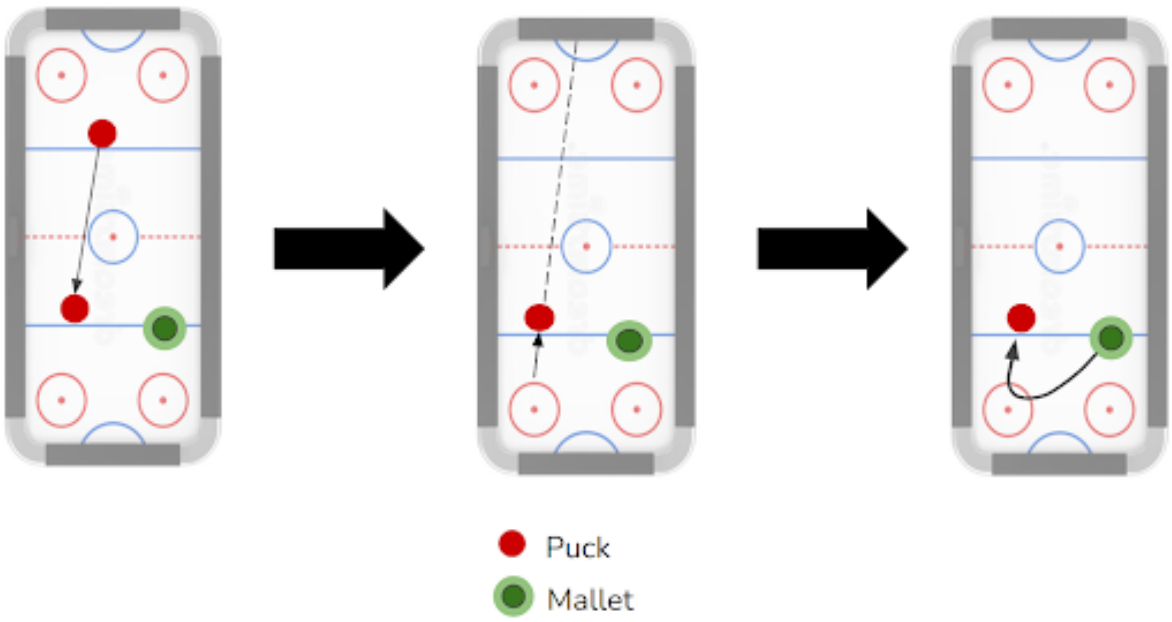

Figure 15: Visualization of an example HLC strategy

In Figure 15, the HLC gets puck tracking information, and notices that the puck is approaching its zone. In the left image the HLC decides that it wants to intercept the puck at a future position. In the middle image the HLC determines a desired shot trajectory, and finds the necessary mallet velocity vector with which to strike the puck. In the right image, a polynomial path is generated using the HLC defined final mallet state and the known current mallet state. It is now up to the LLC to actualize the HLC's desired mallet movement.

Keep in mind that this is meant to be a general example, and may not reflect the actual strategic decisions made by students' HLCs. For a concrete example, see our basic HLC implementation in Appendix J.

7 Conclusions

We set out to create an air hockey robot to provide students with an engaging AI development platform. Such a system would need to be capable of moving the mallet with speed and accuracy on par with a human player. Our H-Bot design avoids the need to move heavy components, resulting in relatively low inertia. Due to the low inertia and high power capacity of the system, the RZA can reach mallet velocities of around 90% of our human-level target of $6\frac{m}{s}$. Power supply voltage drops prohibit us from effectively applying motor voltages over 15 V, though the system mechanics and motor drivers have the capacity for up to 34 V, suggesting that the driving system is, in principle, up to spec.

Once a dependable power system is in place, the most significant limitation on the RZA is the low level controller. To provide a reliable, easy to use interface for students' agents, the low level controller ought to be able to closely follow prescribed paths. At the moment, a functioning PD controller is responsible for tracking the time-dependent mallet position set point. While sufficient for basic play, the PD controller does often miss incoming pucks, lagging the desired path and striking an area where the puck used to be. A well-fit feed-forward controller would likely yield significant improvement in path following by eliminating the lag inherent to PD control.

With respect to puck tracking, the method of using a camera to locate an LED has proven effective. There are still challenges in improving positional accuracy, with lens distortion being the largest contributor to error. Calibrating the placement of the camera is regularly required as the camera mount is not connected to the table, and the two may move relative to each other.

With a basic proof-of-concept high level controller, the RZA can play an interactive game of air hockey. The RZA fulfills our requirements in part - we have provided a robot capable of playing air hockey, but the RZA is not yet capable of beating humans.

8 Recommendations

The path of most efficient improvement for the RZA targets the most significant issue in three different subsystems: mechatronics, puck tracking, low level control. The power supply is holding back the quality of the mechatronics. As such, it should be the primary focus of the mechatronic redesign. Research into power electronics would be worthwhile. The goal is to create a constant voltage supply that does not drop on high current draw and is capable of safely handling back-driven current. Parallel capacitors or batteries may be helpful here.

Lens distortions reduce the puck tracker's accuracy. A distortion correction algorithm has been implemented (see Appendix H for details), but is not sufficiently accurate. Packages exist in OpenCV for lens distortion correction that may be beneficial.

The low level controller often lags behind desired paths. PD depends on error to drive motion and is not perfectly suited to the problem of path following. If the physics of our mechatronic system were actually linear - as we modeled them to be - then a properly IDed feed-forward controller would yield improvements in path following accuracy. Once the linearity of the power system has been improved, feed-forward control becomes feasible. Further model improvements could also be made via a learning controller (see [8]).

In terms of project structure, there is a strong case to be made for splitting the project into interdependent but separate tasks. Multiple capstone teams could take on these subtasks.

However, many of the remaining tasks are not involved enough to fill a complete capstone project. The major areas of improvement discussed above are each too small to be their own project, but together become daunting for a new team to tackle over a single capstone term. Consequently, we recommend that a single 6 person team take on the full project within the scope we have established. This team would be responsible for implementing the changes discussed (linearizing the power system, improving lens correction, and improving the path following capabilities of the LLC). They would also iterate on the mechanism, minimizing inertia while maintaining stiffness. Grouping the students into a single team avoids a situation in which students are stuck iterating and improving on a system beyond relevant levels of quality. If we were to assign a team solely to puck tracking, they are likely to create a puck tracking system that far exceeds the accuracy that is relevant to the task at hand. Their time is better spent if they have the option to pivot away from puck tracking towards low level control, for example.

It may also be worthwhile to have a team focused exclusively on developing a HLC agent for the robot. This could be an intensive project for a small team interested in ML and sim2real. This team would effectively be fore-running the proposed AI course, using the RZA to test out their HLC.

9 Deliverables

1. The RZA
   The table, sandwich and gantry, complete with all circuitry, camera equipment, and motors.
2. GitHub Repository
   Access to all the code written for this project: both code for running the table, and various other software tools.
3. Linux Image
   A hard drive containing an Ubunutu 20.04 image, with ROS and other tools installed.
4. Onshape Cad Model Access
5. Operating Instructions (see Appendix G)
6. Bill of Materials (submitted through the deliverables folder)

10 References

[1] URL: https://answers.opencv.org/question/196693/wrong-perspective-transform/.

[2] *Basicmicro Roboclaw Series Brush DC Motor Controllers*. Version 5.7. Ion Motion Control.

[3] Lou Dahl. *UBC ENGINEERING PHYSICS - FINAL PROJECTS SHOWCASE, 2022*. URL: https://www.loudahl.com/blog/2022/4/10/ubc-physics-engineering-2022.

[4] *DC Motor Speed: System Modeling*. URL: https://ctms.engin.umich.edu/CTMS/index.php?example=MotorSpeed&section=SystemModeling.

[5] D. Craig MacCormack. *NHL Puck Goes High-Tech for 2021 Season in the Name of Analytics*. URL: https://www.commercialintegrator.com/markets/sports/nhl-puck-high-tech-analytics/.

[6] Evan Pham et al. *Deftly Operated Unbeatable Gantry (DOUG) An Automated Air Hockey Robot*.

[7] Lance Roux. *HOW TO APPLY OSHA'S HIERARCHY OF CONTROLS TO MITIGATE SAFETY HAZARDS*. URL: https://www.safetyproresources.com/blog/how-to-apply-oshas-hierarchy-of-controls-to-mitigate-safety-hazards.

[8] Arimoto S, Miyazaki F, and Kawamura S. "Motion Control of Robotic Manipulator Based on Motor Program Learning". In: (1988). DOI: https://doi.org/10.1016/S1474-6670(17)54605-4.

[9] *Ziegler-Nichols Tuning Rules for PID*. URL: https://www.mstarlabs.com/control/znrule.html.

Appendices

Appendix A System Level Diagram

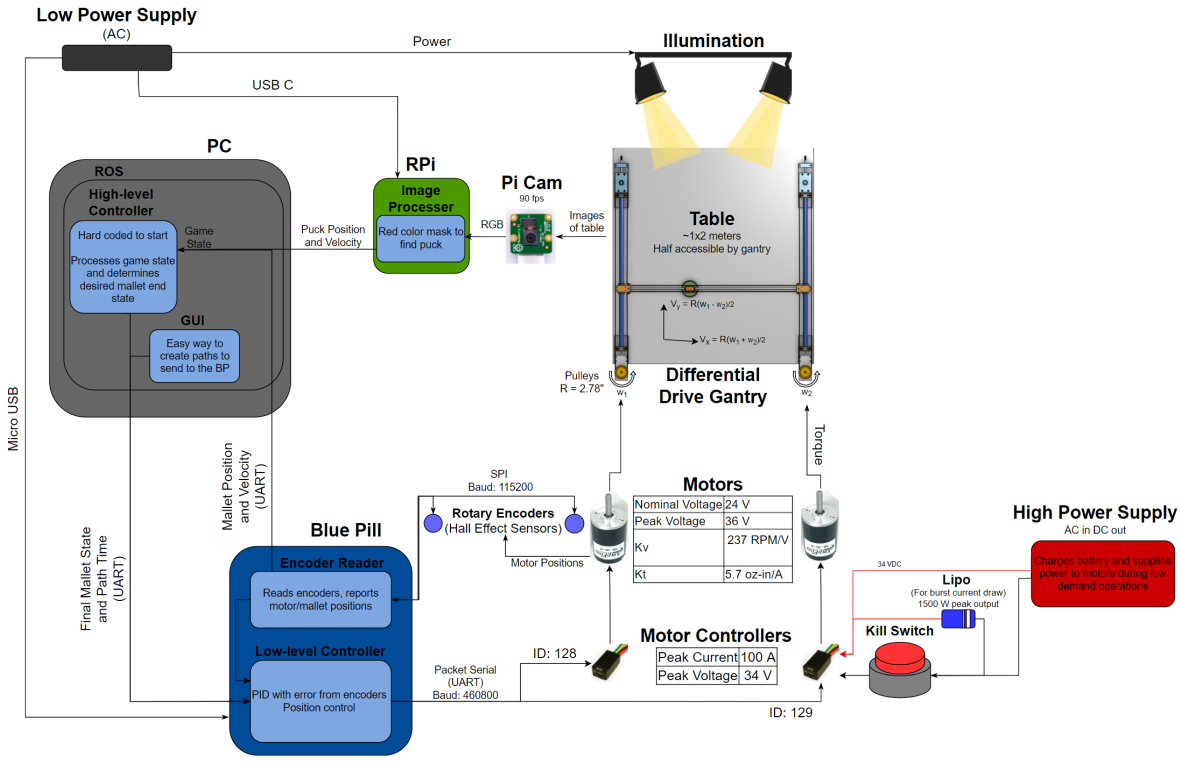

Figure 16: Our overall System Level Diagram that we developed and referenced throughout our 479 term. This diagram includes the high level computing architecture and how it interfaces with the hardware components of the RZA

Appendix B Modeling the Electrodynamics of the Gantry

This appendix was originally a stand-alone document to describe the modeling we undertook of the electrodynamics of our gantry. It was intended for use amongst our team and the terminology reflects that.

B.1 Modeling the Mechanics

We did this by writing down the Lagrangian of the system in terms of generalized coordinates $\theta = (\theta_1, \theta_2)^T$. One can easily transform from $X = (x, y)^T$ by using:

$$\begin{bmatrix} x \\ y \end{bmatrix} = \frac{R_p}{2} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} \theta_1 \\ \theta_2 \end{bmatrix} := E\theta \tag{13}$$

We then want to find the Lagrangian L. This is gross if you actually do it in terms of the masses and inertia of the components. We've done it, its somewhere, and if you really want it I'll dig it up. But I don't want to spend all of my twenties on latex, so let's just skip that. The important bit is that we are able to spot a symmetry in our system. Since nothing depends on position, and the system is symmetric w.r.t. $\theta 1$ and $\theta 2$, we can skip all that and just write

$$L = \frac{1}{2} \begin{bmatrix} J_l & J_s \\ J_s & J_l \end{bmatrix} \dot{\theta}^2$$

Which give that the torque τ we need is found by finding E.L. equations. This gives:

$$\tau = \begin{bmatrix} J_l & J_s \\ J_s & J_l \end{bmatrix} \ddot{\theta}$$

This equation neglects friction, because our Lagrangian derivation neglected it, because Lagrangian mechanics is stupid that way. But since we aim to be deriving all the inertia empirically anyways, this isn't a big deal. Well just tack on a friction term, and the argument from symmetry still applies, so the equations for dynamics should hold. We will have friction terms proportional to θ and X. We consider the frictions in our motors to be identical. This is still a pretty big simplification, since there will be a friction term proportional to the speed (i.e. $\sqrt{\dot{x}^2 + \dot{y}^2}$) resulting from the mallet on the play surface. Being air-hockey, this surface will be slippery, so hopefully this is negligible.

$$\tau_f = \begin{bmatrix} b & 0 \\ 0 & b \end{bmatrix} \dot{\theta} + \begin{bmatrix} f_x & 0 \\ 0 & f_y \end{bmatrix}_{x,y} \dot{X}$$

We need to change the basis of the second matrix, as well as convert X to θ. Using the change of basis matrix E (13) we have

$$\tau_f = \begin{bmatrix} b & 0 \\ 0 & b \end{bmatrix} \dot{\theta} + E^{-1} \begin{bmatrix} f_x & 0 \\ 0 & f_y \end{bmatrix}_{x,y} E\theta$$

This gives a symmetric matrix like the one for inertia. Renaming our friction coefficients to match l,s convention, and putting this together we find an O.D.E relating the driving torque to our angle:

$$\tau = \begin{bmatrix} J_l & J_s \\ J_s & J_l \end{bmatrix} \ddot{\theta} + \begin{bmatrix} b_l & b_s \\ b_s & b_l \end{bmatrix} \dot{\theta} \tag{14}$$

B.2 Modeling the Motors

To be able to do anything useful with this model we have to figure out how much torque we are able to produce. Typically, the torque a motor provide is proportional to the current flowing through it: $\tau = kI$. Now, if we were able to control the current directly that'd be problem solved. But we can't; we can only control the voltage, and let the current be what it may. Thus, we need to model the relationship between current and voltage. Modeling the motor as an LR circuit, and taking into account the back EMF, we have

$$L\dot{I} + RI = V - K\dot{\theta} \tag{15}$$

This constitutes our model for the model. Since we are treating the motors as being identical, L and R are both scalars, while V,I and θ are vectors. We are left with a system of two (matrix) O.D.Es to solve. This is done by moving to the Laplace domain.

B.3 Laplace domain

We wish to convert (14), (B.6) to the Laplace domain, so that we can deal with them algebraically. For now, we will assume 0 initial conditions - so long as this is reflected in the physical test we do, we will be able to extract the quantities we need. We can add the initial conditions terms in later for cases where they do not vanish.

In the Laplace domain, with vanishing I.C.s, (14) becomes:

$$\tau = (Js^2 + Bs)\theta$$

Here, J and b are the matrices from (14). Likewise (B.6) becomes:

$$(Ls + R)I = V - Ks\theta$$

Rearranging gives:

$$I = \frac{V - Ks\theta}{Ls + R}$$

Using $\tau = KI$ we can then write:

$$K\frac{V - Ks\theta}{Ls + R} = (Js^2 + bs)\theta$$

And so:

$$V = \left[\frac{Ls+R}{K}(Js^2+bs)+Ks\right]\theta$$

After expanding we see and grouping like terms we see:

$$V = \left[\frac{LJ}{K}s^3 + \frac{RJ+LB}{K}s^2 + \frac{RB+K^2}{K}s\right]\theta \tag{16}$$

Now, we can define our transfer function H to be the bit in the square brackets. Writing our H as a matrix gives:

$$HK = \begin{bmatrix} LJ_Ls^3 + [RJ_l + Lb_l]s^2 + [R(b_l) + K^2]s & LJ_ss^3 + [RJ_s + Lb_s]s^2 + Rb_ss \\ LJ_ss^3 + [RJ_s + Lb_s]s^2 + Rb_ss & LJ_Ls^3 + [RJ_l + Lb_l]s^2 + [Rb_l + K^2]s \end{bmatrix}$$

$$\begin{aligned} H_{11} &= \frac{LJ_L}{K}s^3 + \frac{RJ_l + Lb_l}{K}s^2 + \frac{Rb_l + K^2}{K}s \\ H_{12} &= \frac{LJ_s}{K}s^3 + \frac{RJ_s + Lb_s}{K}s^2 + \frac{Rb_s}{K}s \\ H_{21} &= \frac{LJ_s}{K}s^3 + \frac{RJ_s + Lb_s}{K}s^2 + \frac{Rb_s}{K}s \\ H_{22} &= \frac{LJ_L}{K}s^3 + \frac{RJ_l + Lb_l}{K}s^2 + \frac{Rb_l + K^2}{K}s \end{aligned}$$

In all, we have 8 unknown quantities. R is easily measurable, and K is given with the motor. In theory, we should be able to measure L with some degree of precision, but efforts to this end have thus far given questionable results. That aside, I'm going to introduce new coefficients so that when we later do curve fitting, we well be better able to ascertain what coefficients change what.

Let's define

$$a_1 = \frac{LJ_l}{K} \quad a_2 = \frac{RJ_L + Lb_l}{K} \quad a_3 = \frac{Rb_l + K^2}{K}$$
$$b_1 = \frac{LJ_s}{K} \quad b_2 = \frac{RJ_s + Lb_s}{K} \quad b_3 = \frac{Rb_s}{K}$$

This way:

$$H = \begin{bmatrix} a_1s^3 + a_2s^2 + a_3s & b_1s^3 + b_2s^2 + b_3s \\ b_1s^3 + b_2s^2 + b_3s & a_1s^3 + a_2s^2 + a_3s \end{bmatrix}$$

That's nicer to look at. There's an important point in doing this simplification, at first glance, we have 8 unknown quantities. However, looking at H, we see there are only 6 values that really make a difference. We're we to try to fit all 8, we'd run into degeneracy issues, and the fit may not terminate, Now, our next step is to somehow find these values empirically.

B.4 Experiments

To understand how our system is going to behave in all circumstances, we need to get all 9 of the above coefficients. We're going to do this by running by driving it with a known voltage, and measuring the resulting motion.

B.4.1 Moving forwards in y

We do this by driving m_1 with some positive voltage V (pwm), and m_2 with $-V$. We know that we will have $\theta_1 = -\theta_2$

$$V = H\theta \implies V_1(s) = ((a_l - b_1)s^3 + (a_2 - b_2)s^2 + (a_3 - b_3)s)\theta$$

There's an important point in here: if we just run forwards, there are only three values we can extract. Nonetheless, by preforming a curve fit of our angle data to the voltage we supplied, we will be able to determine these differences.

B.4.2 Moving forward in x

This is done by driving the motors with the same constant voltage (both magnitude and polarity). We know we will have $\theta_1 = \theta_2$, So

$$V = H\theta \implies V_1(s) = ((a_l + b_1)s^3 + (a_2 + b_2)s^2 + (a_3 + b_3)s)\theta$$

By fitting a curve to these sums, and using our fitted values for the sums of the coefficients we can solve for each individual coefficient.

B.5 Analytic Solution for F-B and S-S Motion

These are the cases described in the above experiment we preformed. Moving either side to side, or forward back, the equations relating V and θ are identical up to the values of the coefficients. Let's define:

$$V_1(s) = (d_1 s^3 + d_2 s^2 + d_3 s)\theta(s) \iff H^{-1} = \frac{1}{d_1 s^3 + d_2 s^2 + d3s}$$

Using partial fraction expansion gives:

$$H^{-1} = \left[\frac{1}{d_3 s} - \frac{\frac{d_2}{d_3} + \frac{d_1}{d_3}s}{d_1 s^2 + d_2 s + d_3}\right]$$

We can now, in a relatively straight forward manner, invert the Laplace transform.

$$H^{-1}(t) = \frac{1}{d_3}\left[1 - \exp\left(-\frac{d_2}{2d_1}t\right)\left(\cosh\left(t\sqrt{(\frac{d_2}{2d_1})^2 - \frac{d_3}{d_1}}\right) + \frac{\sinh\left(t\sqrt{(\frac{d_2}{2d_1})^2 - \frac{d_3}{d_1}}\right)}{\sqrt{(\frac{d_2}{2d_1})^2 - \frac{d_3}{d_1}}}\right)\right] \tag{17}$$

B.5.1 Modes of operation

There are three possible forms that (17) can take depending on the values of $\Delta = (\frac{d_2}{2d_1})^2 - \frac{d_3}{d_1}$

B.5.2 Critical Damping

In the unlikely (read: impossible) event that $\Delta = 0$, we would have neither hyperbolic sines or cosines, nor would we have regular sinusoids. (17) would reduce to:

$$H^{-1}(t) = \frac{1}{d_3}\left[1 - \exp\left(-\frac{d_2}{2d_1}t\right)\right]$$

B.5.3 Over Damping

Assuming $\Delta > 0$, the argument of all exponentials in (17) are real. And so the behaviour of the system is described by (17).

B.5.4 Under Damping

Should we have $\Delta < 0$, the argument of the hyperbolic sinusoids becomes imaginary. However, since we know $\cosh ix = \cos(x)$ and $\sinh ix = i\sin(x)$ (17) becomes:

$$H^{-1}(t) = \frac{1}{d_3}\left[1 - \exp\left(-\frac{d_2}{2d_1}t\right)\left(\cos\left(t\sqrt{-(\frac{d_2}{2d_1})^2 + \frac{d_3}{d_1}}\right) + \frac{\sin\left(t\sqrt{-(\frac{d_2}{2d_1})^2 + \frac{d_3}{d_1}}\right)}{\sqrt{-(\frac{d_2}{2d_1})^2 + \frac{d_3}{d_1}}}\right)\right]$$

B.6 Initial Conditions

Each time we execute a new path we begin with fresh initial conditions. We have to figure out how to accommodate for these. Let's start by repeating the Laplace transforms, but including IC's

$$L\dot{I} + RI = V - K\dot{\theta} \rightarrow LsI - LI_0 + RI = V - Ks\theta + K\theta_0$$
$$I = \frac{V - Ks\theta}{Ls + R} + \frac{LI_0 + K\theta_0}{Ls + R}$$

For the torque equation we have:

$$\tau = J\ddot{\theta} + B\dot{\theta} \rightarrow Js^2\theta - sJ\theta_0 - J\dot{\theta}_0 + Bs\theta - B\theta_0$$
$$\tau(s) = (Js^2 + Bs)\theta - [(Js + B)\theta_0 + J\dot{\theta}_0]$$

Putting these together:

$$\tau = KI = K[\frac{V - Ks\theta}{Ls + R} + \frac{LI_0 + K\theta_0}{Ls + R}] = (Js^2 + Bs)\theta - [(Js + B)\theta_0 + J\dot{\theta}_0]$$
$$V = \left[\frac{Ls + R}{K}(Js^2 + Bs) + Ks\right]\theta - \left[\frac{Ls + R}{K}(Js + B) + K\right]\theta_0 - \frac{Ls + R}{K}J\dot{\theta}_0 - LI_0$$

We have four terms here. The first is just our H matrix from before. The next term has the same coefficients as H, but is one order lower in s. Call this matrix $\tilde{H}$. So:

$$V = H\theta - \tilde{H}\theta_0 - \frac{Ls + R}{K} J\dot{\theta}_0 - LI_0 \tag{18}$$

However, in we can see in (18) that all the terms involving initial conditions are proportional to non-negative powers of s. This mean that the inverse Laplace transform of these terms will just result in various orders of derivatives of the Dirac Delta function. These Dirac's aren't every integrated over - only the multiplication of $H(s)\theta(s)$ leads to integration in the time domain.

$$V(t) = \int_0^t V(t-\tau)H(\tau)d\tau + \mathcal{L}^{-1}\{\text{Initial condition stuff}\}$$

The initial condition stuff is zero for all non-zero time so we simply have:

$$V(t) = \int_0^t \theta(t-\tau)H(\tau)d\tau \tag{19}$$

$$V = H(\dddot{\theta}_d, \ddot{\theta}_d, \dot{\theta}_d, t) + K_p(\theta - \theta_d) + K_d(\dot{\theta} - \dot{\theta}_d) \tag{20}$$

Appendix C Mechanical Assembly Instructions

C.1 Table Assembly

The table consists of two main parts: the top sandwich style portion with the air gap, and the table base which holds the top sandwich.

1. The aluminum mounting plates should be tightened to the table (air sandwich) before it is placed in the table base
2. It can be helpful to also have the forward back shafts firmly mounted in the pillow blocks as additional gripping points
3. The sandwich portion of the table should be lowered into the base, it may be useful to loosen 1 or 2 of the table legs to allow for one of the side panels (shorter side) to expand the length of the table base so the sandwich fits in easier
4. Tighten all table legs once sandwich is installed

C.2 Sub-assemblies

C.2.1 Tensioners

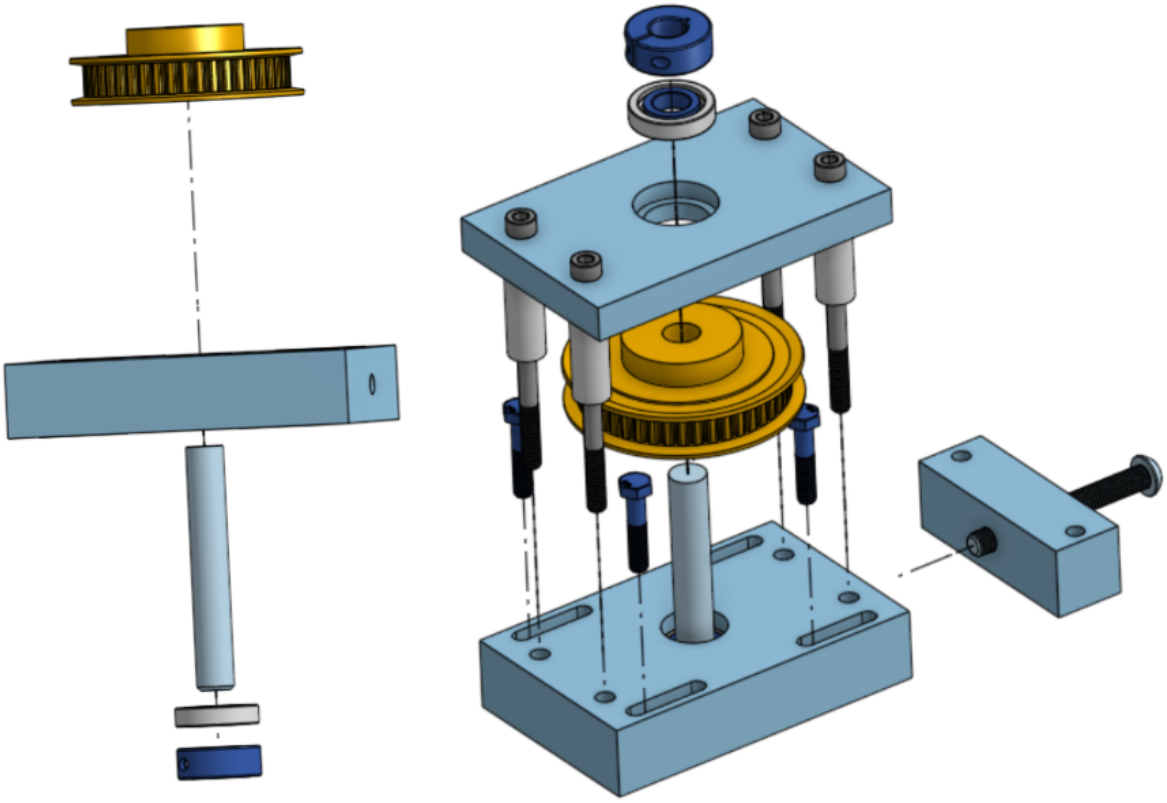

Figure 17: Left: lower plate of belt tensioner exploded view. Right: belt tensioner exploded view

1. Start by placing a bearing and shaft collar on the lower side of the steal shaft
2. Next place the shaft through the bottom plate. Place the pulley on the shaft (above the lower

plate)

3. Next do a similar step by step but with the top plate
4. Use the standoffs to separate the top and bottom plates

C.3 Forward-Back Carriages

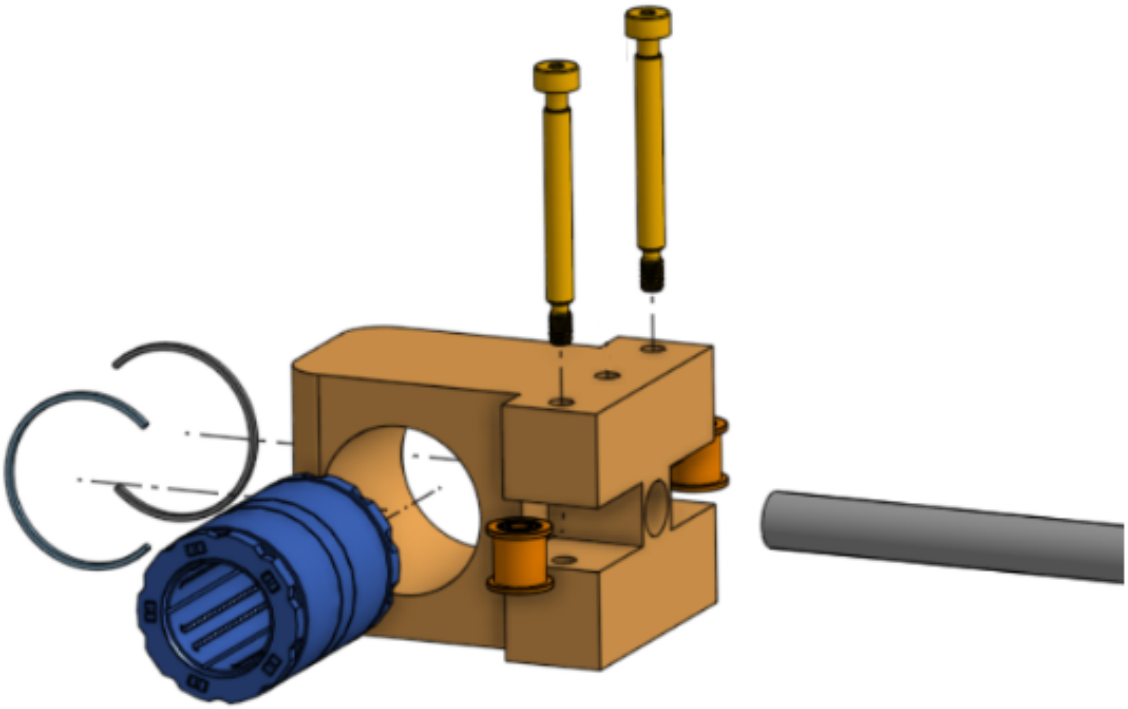

Figure 18: y-motion carriage exploded view

1. Insert the linear bearings into the carriages and secure them with the C-rings
2. Place the small pulleys in their allotted slots and secure them with the shoulder bolts with nuts inserted on bottom side of carriage
3. Insert the side to side shaft WITH THE MALLET HOLDER ON ALREADY

C.4 Gantry Assembly

1. Aluminium mounting plates should be mounted to the table
2. Mount motors (with pulleys on)
3. Mount tensioner assembly, already assembled: Mount pillow blocks
4. Put mallet holder on the side-to-side shaft Put forward-back carriages on the side-to-side shaft
5. Put the forward-back shafts through the forward-back carriages and lower into the pillow blocks
6. Before tightening the pillow blocks, put the mallet into the mallet holder

7. String the belt through the pulleys and tighten the loose ends together to the mallet holder
8. Attach the supporting brace across the forward-back carriages
9. Tension as needed

C.5 Camera Mounting

1. An 8020 L shaped addition is needed to get to the full height and proper FOV
2. Mount the camera to the 8020 brace
3. Attach the camera ribbon cable to the camera Mount the 8020 addition to the main brace
4. Connect the camera to the raspberry pi WHILE THE PI IS OFF
5. Turn on the pi and view the image feed, centering the camera above the table

Appendix D Circuits

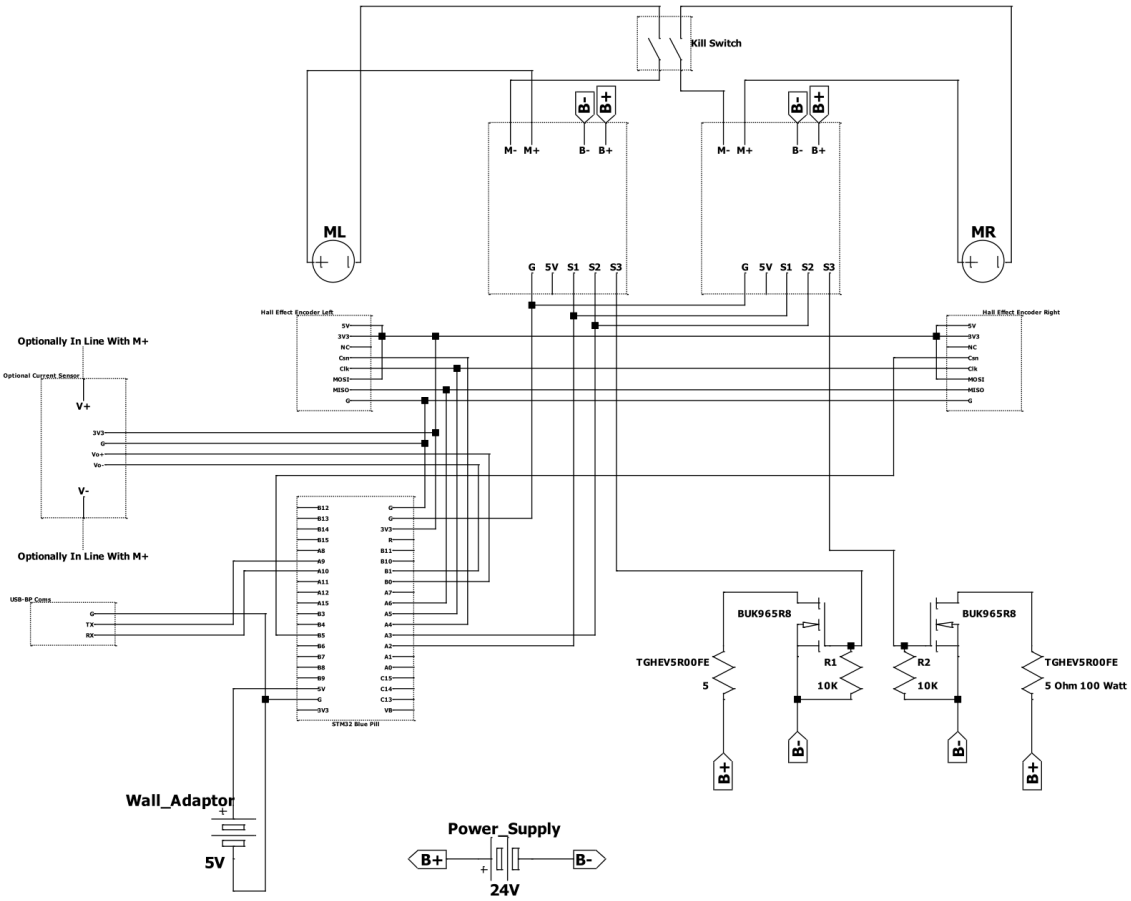

Figure 19: Circuit diagram

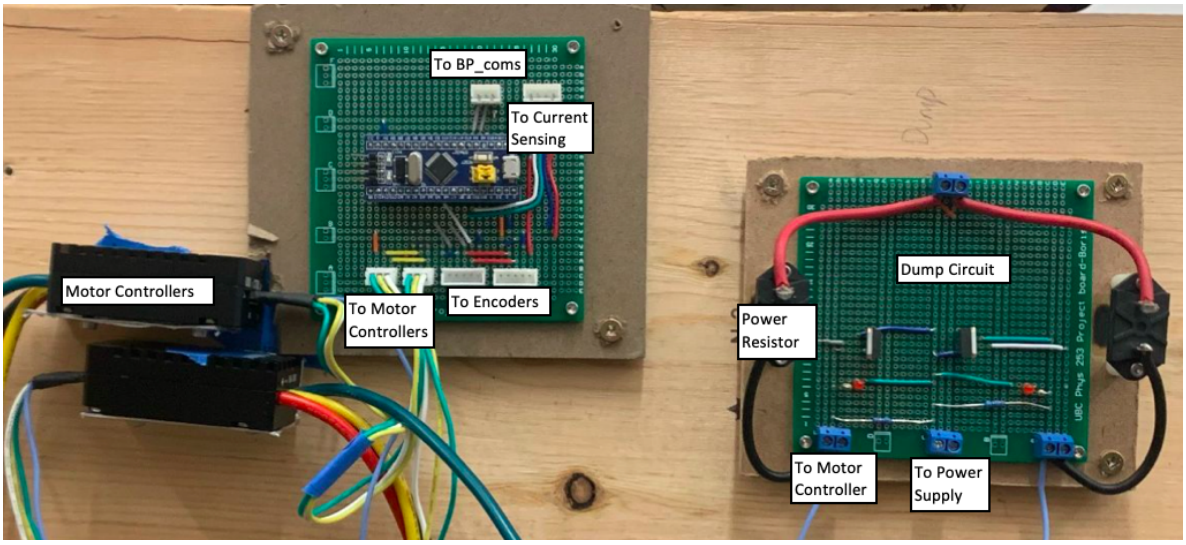

Figure 20: Physical circuitry on protoboard

The above circuit diagram shows the architecture of our mechatronic control circuitry. The blue pill is powered by a 5V USB to wall adaptor. It is connected via one of its SPI communication lines to 2 hall effect encoders. Both encoders share all the same connections except for the chip select

pins which allows the microcontroller to choose which it is reading from at any given time. The motor controllers are also connected to the blue pill over one of its serial communication lines. The controllers share the same connections and are selected by the blue pill via their serial address, set in BasicmicroMotionStudio (see appendix E.2). The motor controllers are also each connected to a voltage clamp circuit through their respective S3 pins. The blue pill also has lines going to an optional current sensor. When connected to the pill the current sensor should also be placed in line (in series) with the motor's high voltage line (ie: current sensor V- connected to motor controller M+ and current sensor V+ connected to motor +). Finally, the blue pill is communicated over BPComs via the tx,rx communication line through the USB connection.

Appendix E Motor Controller Information

BasicmicroMotionStudio is available for Windows machines and is useful for setting up the Roboclaw motor controllers. Its GUI allows you to change many key settings in the motor controller easily.

E.1 Firmware

Upon unboxing, the Solo-60 motor controllers need a firmware update which will be prompted when connected to a laptop via micro-usb cable and BasicmicroMotionStudio (BMS) is opened.

- At time of writing this (April 2022) the motor controllers are on firmware version 4.1.34

E.2 General Settings

- Control Mode: Set to packet serial, this allows for serial communication to a microcontroller. The main command of interest is writing the motors a speed value between -127 and 127 which determines PWM duty cycle (0 = not spinning, 127 = full speed, +/- for direction)
- PWM mode: We used sign amplitude
- Packet Serial address: this is the address assigned to each motor controller. It doesn't matter which value you assign it as long as each controller has a unique value. It is used by the microcontroller to distinguish which controller receives the sent command.
- Baud Rate: We used the maximum value available, 460800, as this is available to be matched by the microcontroller and the largest possible ensures the fastest possible communication.
- Battery Cutoff: if using a power supply the next fields are critical and this field should be set to "use user settings"
- Max Main Battery: this needs to be above the power supply voltage but should be close to it (no more than about 1 volt higher). If the controller sees the battery voltage go higher than this (from back emf and for example) then it will send a signal to pin S3 to turn on the voltage clamp. What the controller thinks the voltage of the battery is can be seen at the top of the window in the "main battery" field. If the power supply voltage is adjusted the max battery voltage field should be checked and made sure that it is above the new power supply voltage before a connection is made.
- Min main battery: If the voltage of the supply drops below this (from a voltage dip due to high current draw for example) the motor controller will tell the motors to "freewheel" meaning no voltage is applied.
- Logic battery was not used at the time of writing this report.
- S2 Mode: should be dissabled
- S3 Mode: should be set to voltage clamp if you are using a voltage clamp circuit (needed if using a power supply).

Appendix F Software Directory Structure

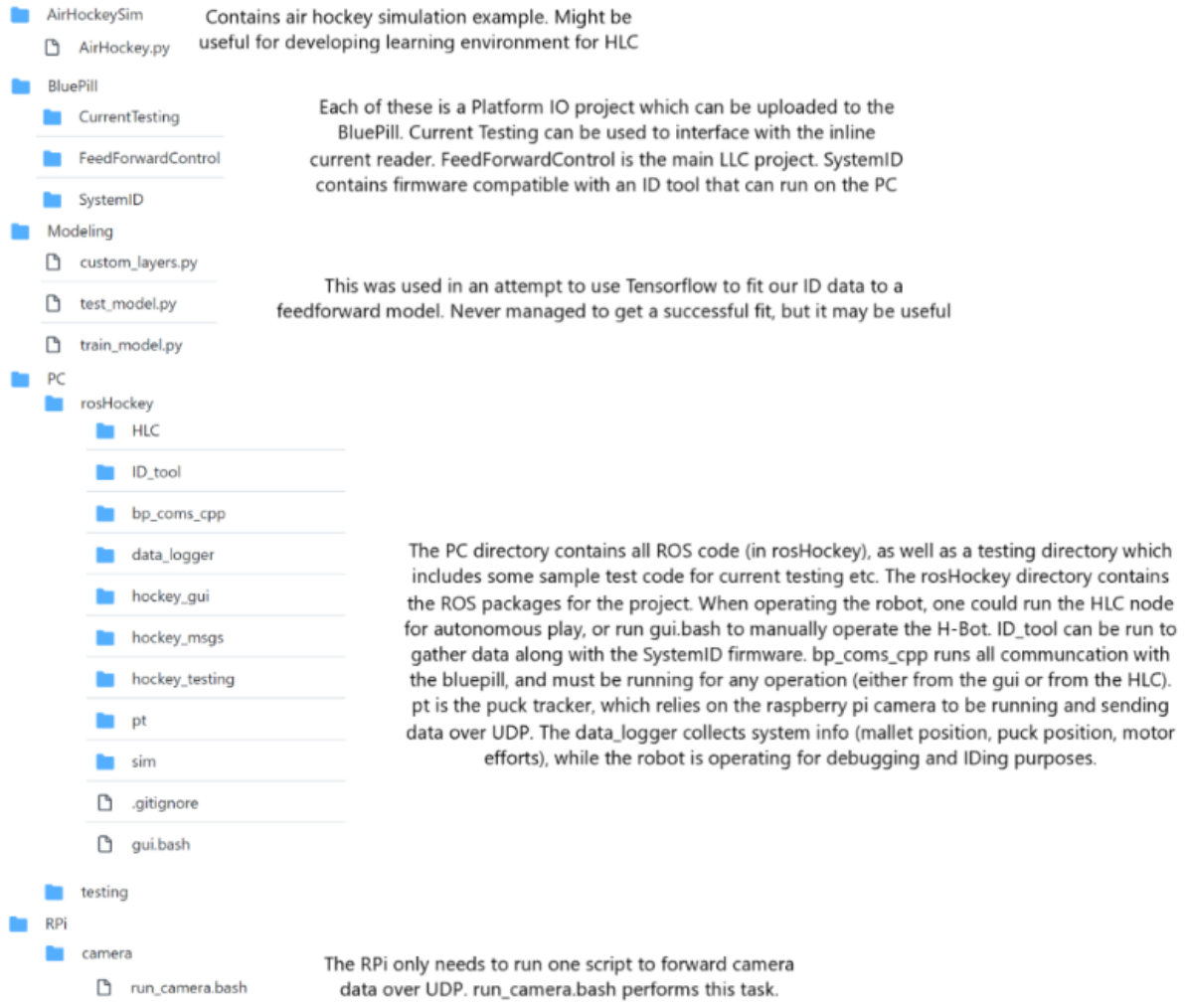

Figure 21: The general directory structure currently in the Github

Appendix G RZA Operating Instructions

The following is a rough guide to operating RZA. This system is dangerous, and you should always be aware of when the gantry is powered, and whether you are currently sending path commands to the motor controllers. If you are unsure, use the ROS capabilities to echo /PATH. If you make changes to the low level controller, small bugs may result in egregious motor effort commands from the BluePill to the motor controller. You can always echo /MOTOR while the motors are powered off to see what kind of effort values are being sent to the motor controllers. This system is dumb. It will ram itself into a wall at Mach 1.3 if you ask it to. Please be safe. Consult the additional safety Appendix I for more information.

We use ROS to operate RZA as it allows us to easily run several scripts in parallel, while also state variables are available to all ROS nodes through ROS topics. Most of our scripts were originally written in Python, but we transitioned to C++ as process loop speed requirements became more stringent.

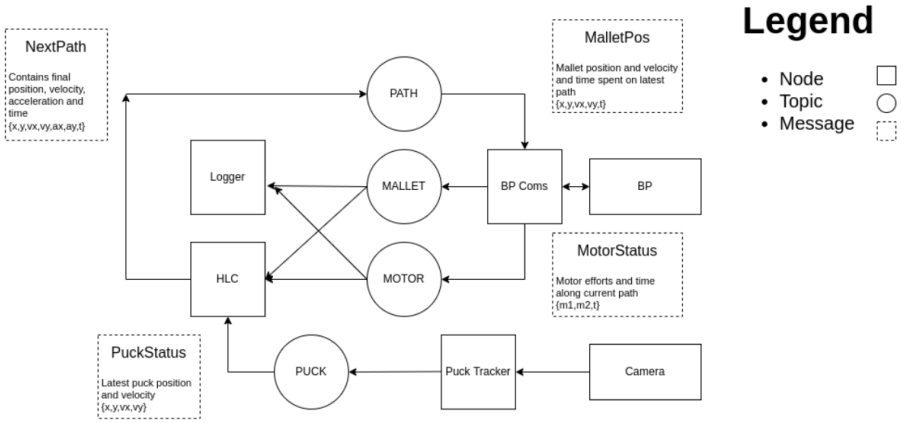

Figure 22: The ROS network structure, showing all nodes, topics, and message types

Figure 22 shows how the different ROS nodes interact with each other.

Here is a general workflow example for writing code and updating a script. I will use puck tracker for this example.

1. Make changes to /PucksInDeep/PC/rosHockey/pt/src/puck_tracker.cpp
2. Save changes (CTRL+S)
3. You must source ros and build the packages in /PucksInDeep/PC/rosHockey/
   (a) "cd /PucksInDeep/PC/rosHockey/"
   (b) "sourceros"
   (c) "Sourcelocal"
   (d) All run commands in the following section assume you are in the rosHockey directory
4. If the script is a ROS node, it must be built with colcon build. The general structure is:
   (a) "colcon build –packages-select PACKAGE_NAME"

i. If it's a python script (such as HLC) you can run this with "–symlink" such that you don't have to build before running. It will build it when you call "ros2 run"

ii. "colcon build –packages-select HLC –symlink"

(b) https://colcon.readthedocs.io/en/released/user/quick-start.html

(c) PACKAGE_NAME is set in CMakeLists.txt or setup.py and package.xml.

(d) "colcon build –packages-select pt"

5. Now you can run it. Ensure that it is not currently running in another terminal tab. You can use "htop" command or "ros2 node list"

   (a) "ros2 run pt puck_tracker"

   (b) The script you run must be made executable in CMakeLists.txt or setup.py

6. Woohoo!

Here are some good ros commands to know:

- "ros2 topic echo ⟨topic⟩"
  - Useful for monitoring, here ⟨topic⟩can be /PUCK or /MALLET or etc.
- "ros2 node list"
  - See what nodes are currency running
- "ros2 topic list"
  - See what topics are currently live
- "htop"
  - Not a ros command, but let's you know what program is frying your computer
- "ros2 topic pub ⟨topic_name⟩⟨msg_type⟩"
  - Publish to a topic manually, this is nice if you are just publishing to /PATH and want to manually move the mallet around. Use "–once" to only publish once. Alternatively, just use the GUI described later

Here are all the main scripts that are used to operate RZA. These are all functional at the time of handoff. There are a few other packages that we left in the repo such as systemID and hokey_testing that were still being developed and might be useful for students wishing to develop a new LLC.

- Pi Scripts
  - /PucksInDeep/RPi/camera/run_camera.bash
  - "./run_camera.bash"
  - To access the pi use ssh over ethernet
    - "ssh pi@10.42.0.124"
      - This was the ip address when we used the pi, but it's not static and is subject to change.

        - Pi password: enph353
    - This bash script lives on the Pi. It sets up the UDP port from the Pi and sets all camera settings
        - Use nano or vi or whatever to edit bash script
        - https://www.raspberrypi.com/documentation/accessories/camera.html
    - This script must be run after puck_tracker. It will close automatically if puck_tracker dies. It's best to ctrl-C on puck_tracker rather than on the Pi
- Puck tracker
    - /PucksInDeep/PC/rosHockey/pt/src/puck_tracker.cpp
    - "ros2 run pt puck_tracker"
    - Run before running the pi bash script
    - Most arguments are detailed in calibration appendix
    - tracker::show(void) in tracker.cpp can be edited to show binary mask image or enable video writing
    - the files in pt/calibration should be edited manually to adjust HSV mask and table camera calibration (see more in calibration appendix)
- BP coms
    - /PucksInDeep/PC/rosHockey/bp_coms_cpp/src/bp_coms.cpp
    - "ros2 run bp_coms_cpp bp_coms"
    - This sets up communication from the PC to the bluepill. Can sometimes fail and must be reset.
    - To see if it's running check /MOTOR or /MALLET
    - This must be running before HLC or GUI
- HLC
    - /PucksInDeep/PC/rosHockey/HLC/HLC/main.py
    - "ros2 run HLC HLC"
    - This is the basic high level controller we developed in the few days before the project fair
    - As soon as this script is running, paths will start to be sent to the motor controllers, make sure hands are out of the way (or the motors are unpowered)
    - Build it with symlink so that you don't have to colcon build each time
    - Sorry about the logic
- GUI
    - /PucksInDeep/PC/rosHockey/hockey_gui/hockey_gui/main.py
    - "./gui.bash"

  - This is an alternative to the HLC to send paths to the bluepill.
  - Although this script technically launches bp_coms if it isn't already running, I've found that it is more stable to first run bp_coms, then run the GUI.
  - Mode of operation: click, generate path, send path
- Logger Node
  - /PucksInDeep/PC/rosHockey/data_logger/data_logger/logger_node.py
  - "ros2 run data_logger logger"
  - After running it will write everything published to a topic into a text file, ctrl-c to end data collection
  - You can edit logger_node.py to write more data to text file if you add more topics
  - Use plotter.py (not a ros script) to plot data, file read arguments and plotting arguments must be changed manually
- Bluepill script
  - /PucksInDeep/BluePill/FeedForwardControl/src/main.cpp
  - main.cpp is the combined communications and low level controller script running on the bluepill
  - Upload it using platform.io and the JTAG
- Common Operating Procedures
  - Here is a example procedure to run RZA and start playing air hockey. You must be in /rosHockey to run all the scripts and have ROS sourced
    1. Open terminal
    2. "ros2 run pt puck_tracker"
    3. Open new terminal window
    4. "ssh pi@10.42.0.124"
    5. "enph353"
    6. "./run_camera.bash"
    7. Open new terminal window
    8. "ros2 run bp_coms_cpp bp_coms"
    9. Open new terminal window
    10. "ros2 run HLC HLC"

Platform I/O can upload firmware to the bluepill, and upload properties are configured in platformio.ini files in each project. Upload and build requirements differ for the different bluepill varieties. The left-most blue pill above, with the round reset button, requires the configuration as

shown in the code. The bluepill on the right with the square button requires the upload flag to be set, as shown in the commented line.

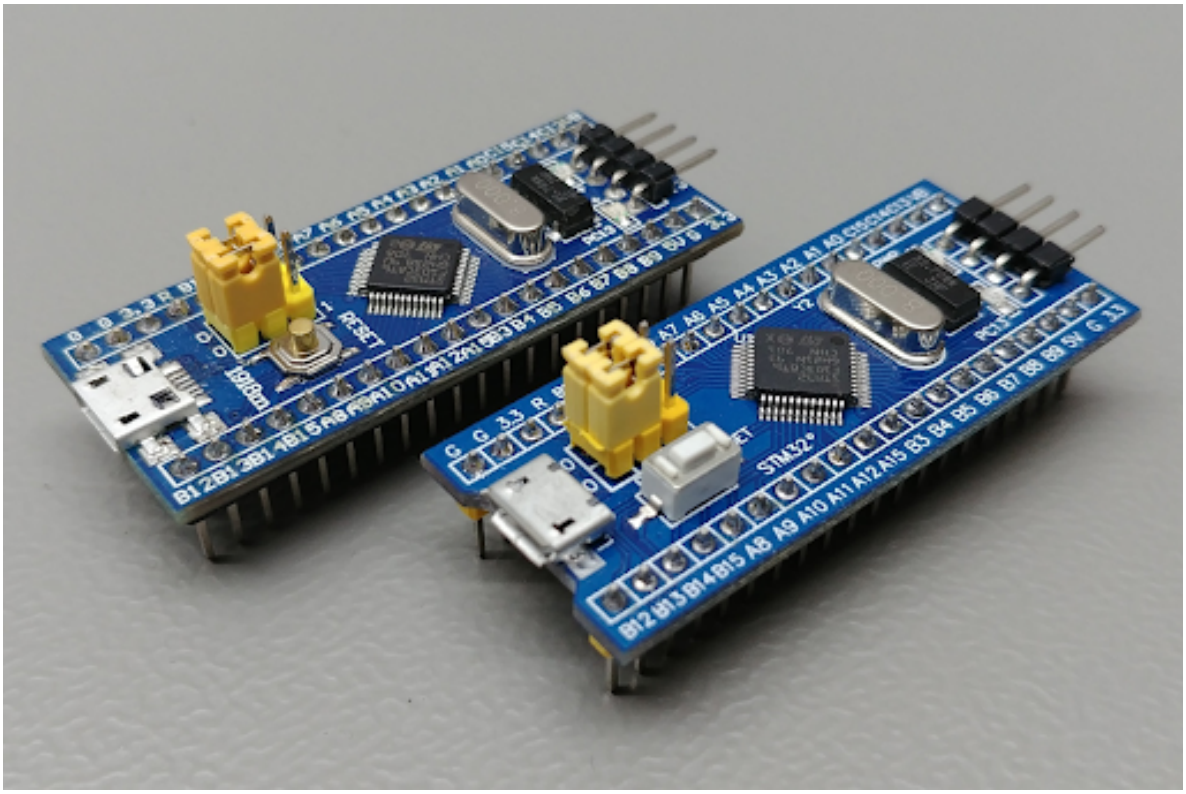

Figure 23: There are two common bluepill varieties, easily distinguished by the shape of their reset button

```
[env:bluepill_f103c8]
platform = ststm32
board = bluepill_f103c8
framework = arduino
debug_tool = stlink
upload_protocol = stlink
; upload_flags = -c set CPUTAPID 0x2ba01477
```

Figure 24: The upload configuration requirements differ between bluepills. The commented line may be needed if you are using a certain variety.

Appendix H Camera Calibration Instructions

This is a guide to calibrate the Raspi camera. To do this it is assumed that the pi is powered and connected to the camera, that you are able to ssh into the pi, and that you are able to run ROS on your machine. It is important that the camera is first centered as best as you can over the table as this will minimize the distortions that are needed to be corrected in software.

There are two calibration settings. The first is the perspective warp transform, which will map the four corners of the table to a rectangle. This will correct for not having the camera perfectly centered

as seen in Figure 25:

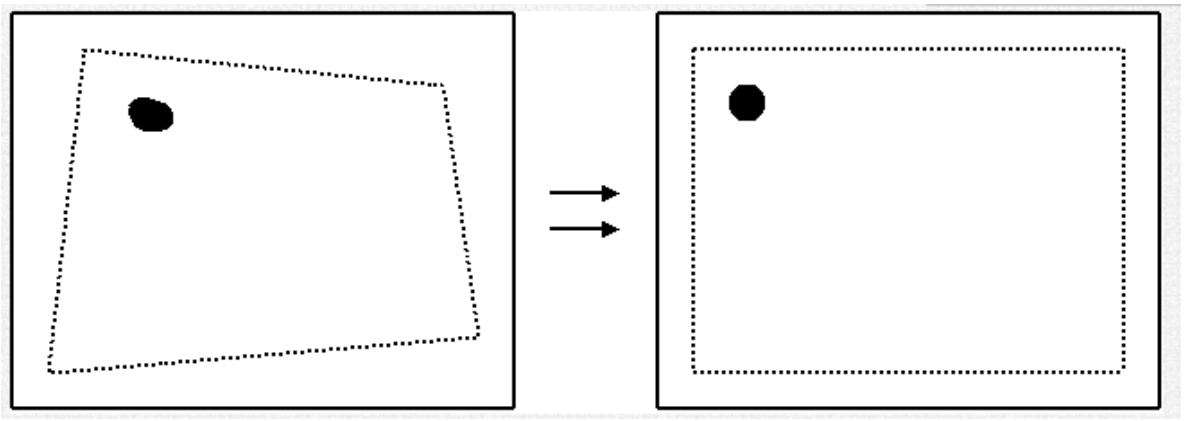

Figure 25: Basic example of how a skewed input image undergoes a perspective transform to become rectangular[1]

The idea is to use the LED puck to retrieve the value of the four corners. To output the current pixel value of the puck to terminal, uncomment "cout ⟨⟨x_img ⟨⟨" " ⟨⟨y_img ⟨⟨" " ⟨⟨"n";" in /PucksInDeep/PC/rosHockey/pt/src/lib/PuckTracker/tracker.cpp. Place the puck in the corner #1 as seen in Figure 26. The corner numbers are noted in Figure 27. From the terminal, read the x and y pixel values and enter these into line 1 of /PucksInDeep/PC/rosHockey/pt/calibration/calibrate.txt. Move the puck in order to each subsequent corner, and input the pixel values into the respective lines in calibrate.txt.

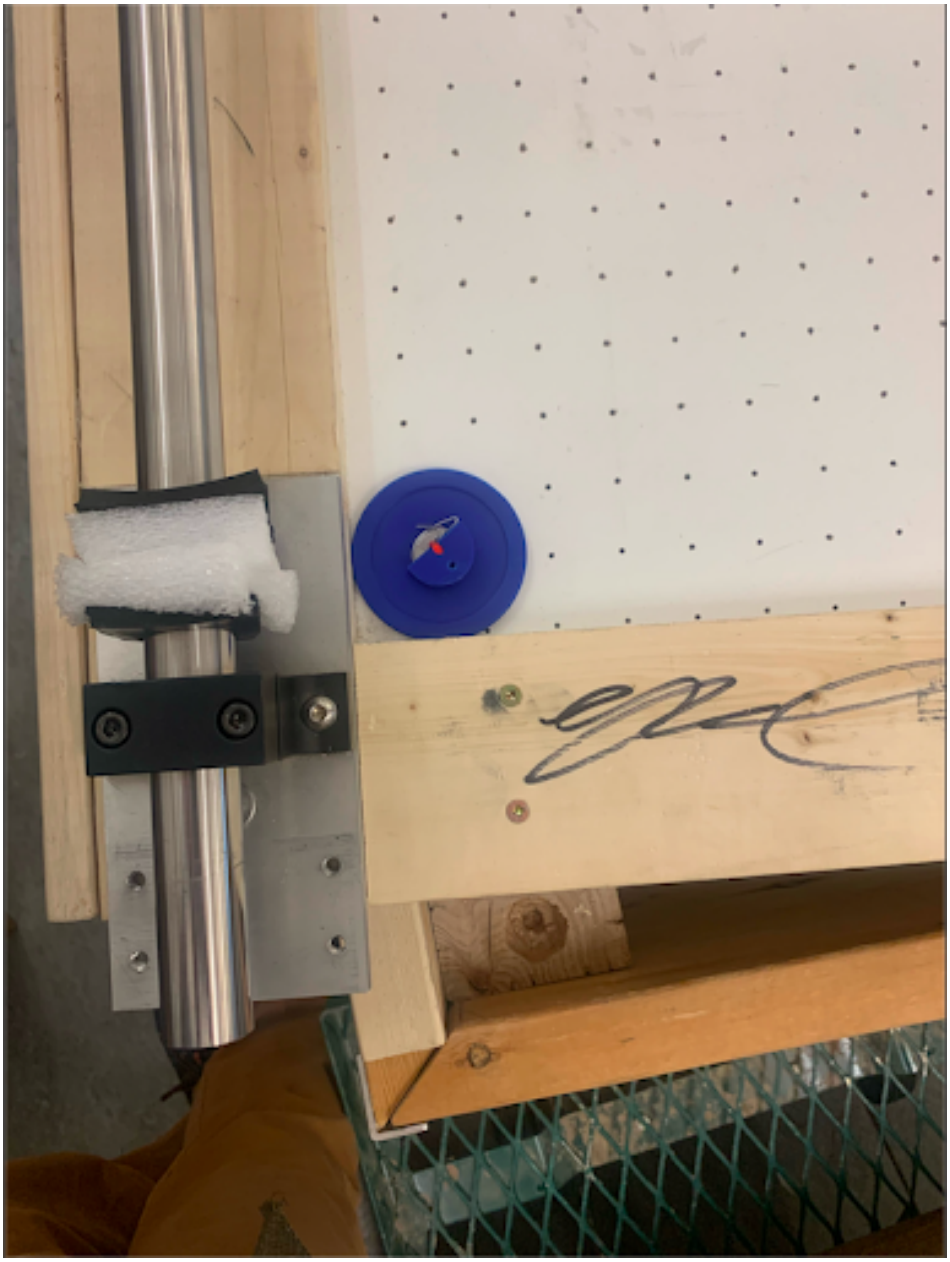

Figure 26: Puck placed in lower left corner of table to set this as the (0,0) position

The calibration procedure will set the center of the puck when placed in corner #1 as (0,0) and the center of the puck when placed in corner 3 as (TABLE_X_DIMS, TABLE_Y_DIMS) as defined in /PucksInDeep/PC/rosHockey/pt/src/lib/PuckTracker/tracker.hpp. This obviously means that the origin of the table is not the corner, but a puck radius in from each edge. The origin point is arbitrary, as long as you are consistent between the mallet origin and puck origin.

The mallet origin is determined by the constants "start_x_offset" and "start_y_offset" in "/PucksInDeep/BluePill/FeedForwardControl/src/main.cpp". The mallet cannot reach the inside corner as the travel of the mallet is limited by the safety padding and the placement of linear guide rails. When the BluePill is initiated or reset, the left motor will spin clockwise bringing the mallet into the bottom left corner until it is stopped by the safety padding. The BluePill treats this point as the mallet origin. To offset the BluePill mallet origin such that the mallet and the puck share the same origin (which is

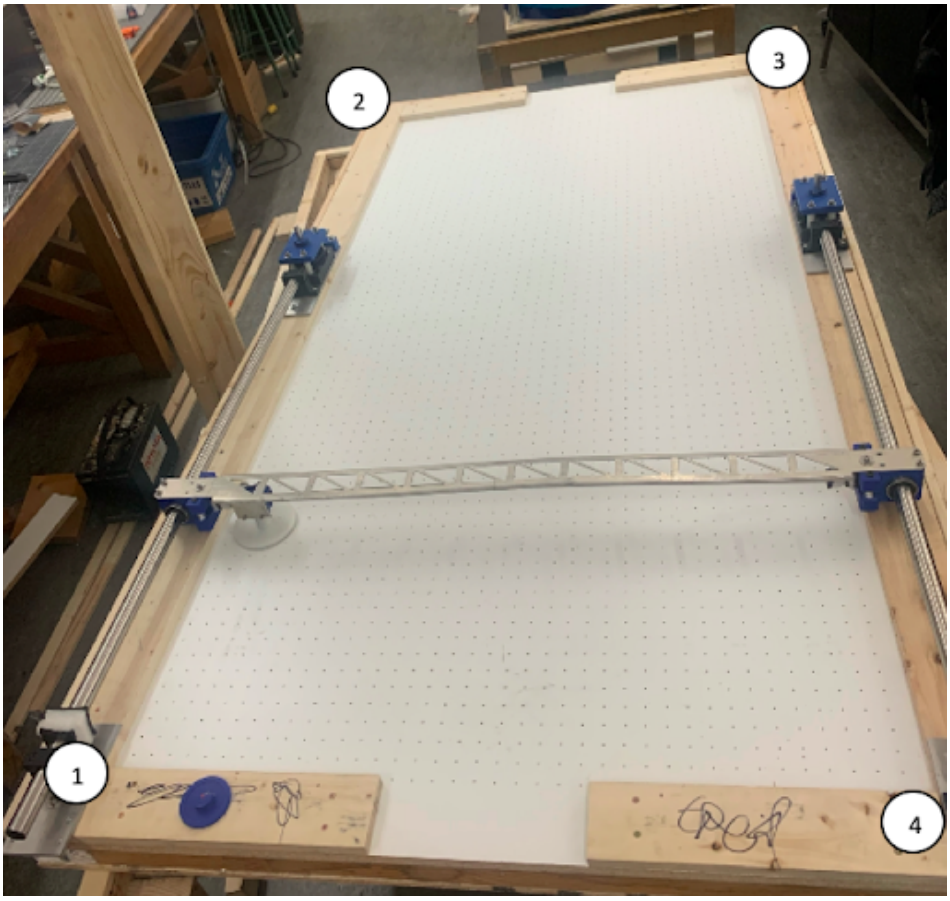

Figure 27: The tables corners are each assigned a number indicating the order in which their positions in the camera frame should be taken

critical if you'd like to hit the puck), adjust the "start_x_offset" and "start_y_offset" in main.cpp on the BluePill by measuring the offset pictured in Figure 28 with a ruler or some callipers.

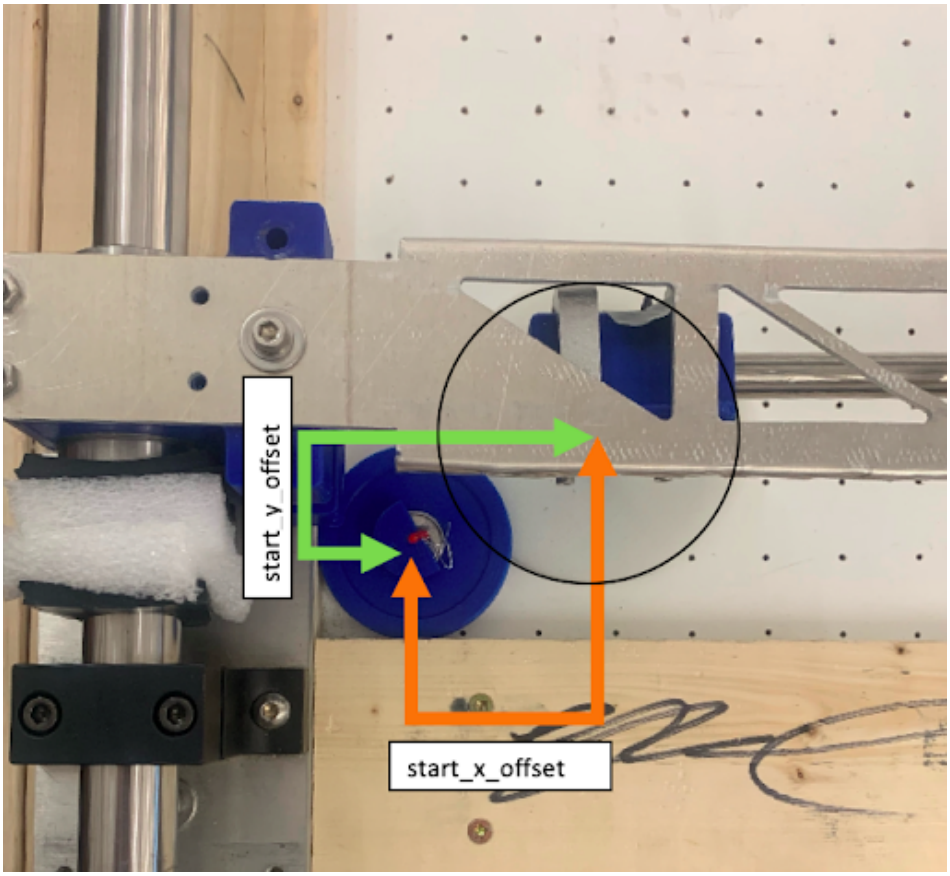

Figure 28: The puck can reach further into the corner than the mallet, so a positional offset is needed.

The wide angle lens introduces some major distortion into the image. The puck_tracker process assumes that the image after the perspective transform is a perfect rectangle, which is obviously not the case. We deal with this distortion by approximating the curvature of the camera as parabolic (as opposed to circular), and offset the coordinate value of each pixel based on how far it is from the center of the image.

To do this you must first set "const float A" and "const float B" in tracker.hpp to 0. These are the constants that we will be trying to compute to correct for lens distortions. With the puck placed in corner #1, you should note the position of the puck by calling "ros2 topic echo /PUCK" in a terminal window. The puck position should be (0,0) within a half cm. If it is not, then you either bumped the table/camera, or made a mistake during the initial calibration. Next move the puck along the sidewall towards corner #2 until it is halfway between corner #1 and corner #2. The x-coordinate should no longer be reading 0cm, this is due to lens distortion. Write down the x-coordinate from /PUCK. Move to the other side of the table and read off the x-coordinate when the puck is centered between corner #3 and corner #4. With no lens distortion, this should read TABLE_X_DIMS. But it will be offset due to lens distortion. Note the offset. Take the average of these two offsets (x offset between #1 and #2, and x offset between #3 and #4), and input it as the "const float x_offset" in tracker.hpp. Now we need to do the same for the y offset. Move the puck halfway between corners #1 and #4. Without lens distortion, this should read 0cm. Note the offset in the y position. Now move the puck between corners #2 and #3. Again, without lens distortion this should just be TABLE_Y_DIMS. Note the offset, and take the average. This is "const float y_offset".

tracker.hpp will automatically compute the A and B constant values that will apply the parabolic

corrections. This is not a great solution, but it's the solution that we came up with the day before the project fair. The puck position will sometimes be off by as much as 4-5cm, which is obviously not great. A good improvement to the image processing software would be to find some better way to correct for the spherical distortions. Possible solutions include using a better camera with less distortion, implementing a better distortion algorithm, using two cameras, or fixing the distortions with some sort of lens.

Appendix I Safety

This is a rough guide to safety and risk mitigation that should be read and reviewed by anyone working on this table. If you are a team that is further developing the air hockey project, you must continue to develop your own risk mitigation strategies.

This machine is dangerous. The RZA can source up to 100A and move at 25 km/h with no regard as to what or who is in the way. We cannot build a functioning air hockey table without these hazards, and so it is our job as engineers/designers to mitigate the risks posed by the hazards. Where possible, we eliminate risk completely by removing hazards. This is the most effective risk mitigation strategy.

Looking at OSHA's heirarchy of controls, we can see the decreasing effectiveness of other risk mitigation strategies.

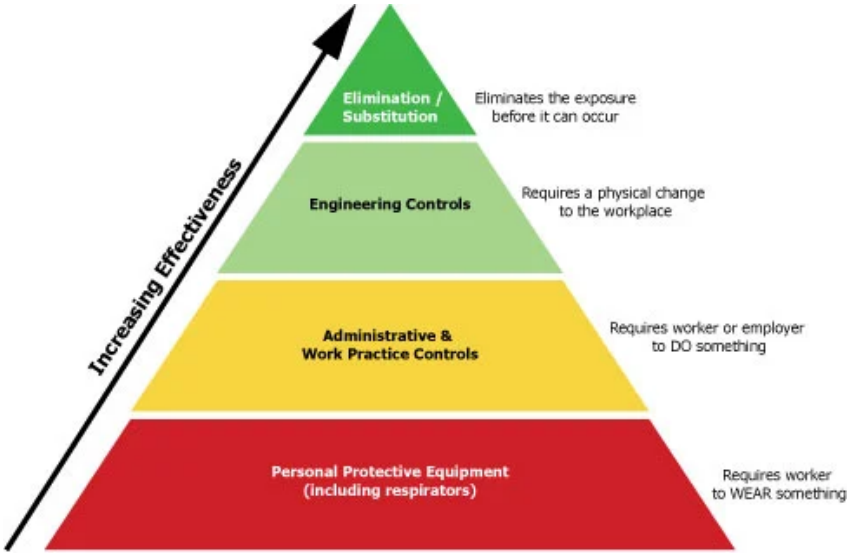

Figure 29: OSHA's heirarchy of controls to mitigate safety hazards[7]

The next most effective type of control is an engineering control. An example of this would be bounding the playable area for the HLC. The very rudimentary HLC that we developed only hit pucks that were on our half of the table, even though the gantry was long enough to reach into the opponent's zone. This created a larger buffer region between the opposing player and our gantry to minimize the risk of a collision. These types of controls are good to implement, but should either be very robust or not depended on. Especially when it comes to software, small bugs may nullify the engineering controls you implement. A better type of engineering control would be hard stops to prevent the gantry from moving into the opponent's zone.

When we began operating the table, we began assigning a single person to be "Safety Sheriff". This person was designated with a goofy cowboy hat, so that everyone working on the table clearly understood who was the current Sheriff. The Sheriff was responsible for powering the motors, running the HLC, instructing people when it was safe or not safe to reach over the table and adjust the gantry. Having a single person that is in charge of operating the table means that there is no chance for miscommunication: "I thought the table was off, so I started unscrewing the mallet holder and then it sprang to life and hit me". The Safety Sheriff position worked well for our team, and I would encourage anyone picking up this project to use a similar strategy. This is an example of a work practice control.

These only work if you are consistent in your work procedures. If you make a rule within your group, be consistent. As soon as one person starts bending the preset rules and inconsistency develops, you are at higher risk of a miscommunication.

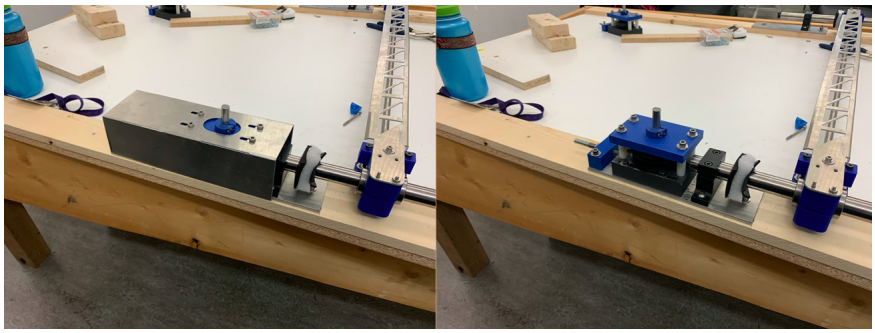

Figure 30: Protective covers are placed over the tensioners to cover the fast moving pulley-belt system

The last type of safety measure we can implement is PPE. This is anything from safety glasses to the protective covers we placed on the pulleys. We also placed foam protective padding along all linear rails to soften impacts. These are a last resort. If the table is constantly disintegrating and splintering, safety glasses only prevent you from going blind, not taking a puck to the teeth. If you find yourself increasing PPE and other protective measures on the table, you should ask yourself whether you have done all you can to eliminate risk instead of mitigating it. That being said, there are many hazards inherent to our project (large power, fast moving parts), and so we have worked to mitigate these risks as best we can.

Appendix J HLC

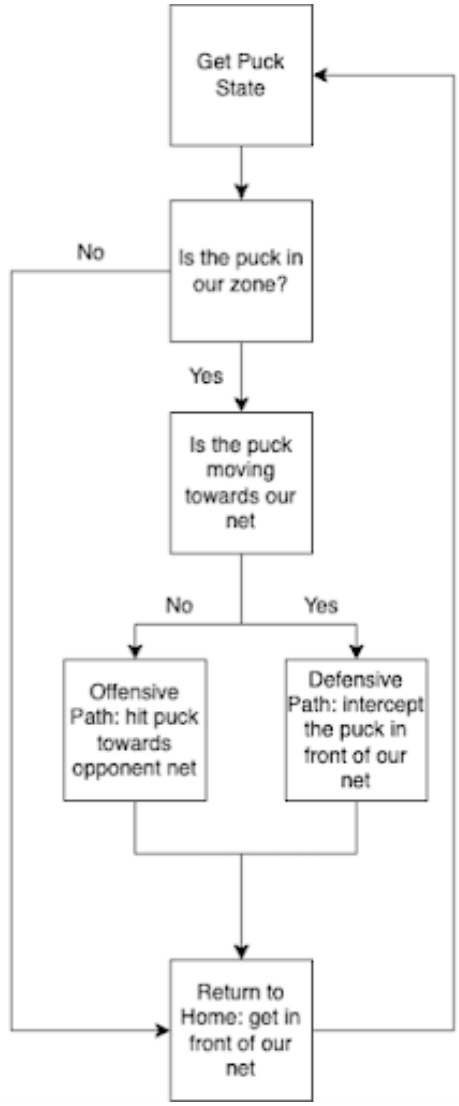

Figure 31: The flowchart for the current, basic high level controller