Autonomous Foosball Opponent

A Technical Report submitted to the Department of Electrical and Computer Engineering

Presented to the Faculty of the School of Engineering and Applied Science

University of Virginia • Charlottesville, Virginia

In Partial Fulfillment of the Requirements for the Degree

Bachelor of Science, School of Engineering

Coleman Jenkins

Spring, 2023

Technical Project Team Members

Hudson Burke

Aidan Himley

Jake Long

Zach Yahn

On my honor as a University Student, I have neither given nor received unauthorized aid on this assignment as defined by the Honor Guidelines for Thesis-Related Assignments

Harry Powell, Department of Electrical and Computer Engineering

Autonomous Foosball Opponent
Single Use Smoke Machine

Hudson Burke, Aidan Himley, Coleman Jenkins, Jake Long, Zach Yahn

December 13, 2022

Capstone Design ECE 4440 / ECE4991

Signatures

Statement of Work

Hudson Burke:

Hudson was primarily responsible for all of the mechanical design and integration. He researched actuation methods to determine the most practical and effective method to control the foosball player rods and put together a list of components to order to assemble the belt-driven actuators. To ensure that the motors could achieve adequate performance based on calculated metrics, he calculated the required speeds and torques to move the rods linearly and rotationally. Hudson designed all the 3D printed and metal parts and performed nearly all of the machining and other hand tool work needed. He designed the entire device in Fusion 360 to mock-up the layout for practical implementation.

Hudson also consulted with other teammates to coordinate the integration of the mechanical components with the electrical hardware and embedded firmware. He worked closely with Jake and Coleman when they were designing the motor drivers and limit switch and encoder input circuitry to give specifications for all the switches, optical interrupters, motors, and encoders. He also worked with Coleman to write the script and record shots for the final video. With the rest of the team, he spent the last few weeks of the semester fully assembling, and debugging, the final system.

Aidan Himley:

Aidan designed and programmed the entirety of the image processing pipeline based on Video for Linux 2 (v4l2). He researched low-level methods for interfacing with the camera, learned how to control it using v4l2, and implemented all of the logic of the loss function, corner detection, ball center calculation, and relative position calculation that appears in the final version of the code, as well as a finite impulse response filter-based method for ball center calculation that was unused in the final version due to worse reliability and speed performance than the threshold and centroid-based method. In addition, he designed and programmed the tools for viewing the output of the image processing at runtime and calibrating its parameters.

Among Aidan's other contributions, he researched methods of achieving deterministic real-time performance on the Raspberry Pi and applied the real-time patch its Linux kernel. He also overhauled the player planning algorithm to use the region-based approach with Block, Shoot, and Ready regions. Towards the end of the semester, he pair programmed with Jake to implement much of the embedded code on the MSP432, including the encoder calibration routine, communication to and from the Raspberry Pi, the stall watchdog timer and stall recovery, and the motor control logic to implement rotational motor action states and linear motor positions.

Coleman Jenkins:

Coleman was responsible for the driver board layout, design of the power supply transient protection, and selection of components. He designed the encoder input protection for both linear and rotational motors and the 5V and 3.3V isolated supply. Coleman worked closely with Jake to design the power distribution for high power and signal level devices connected to both boards.

He worked on getting the boards ready for assembly with Jake and took the lead on communicating and working with WWW. With Jake, he worked on debugging the hardware issues and selecting new components. He implemented the initial version of the game play and planning algorithm which was later modified and used by Zach, then by Aidan and Jake on the Raspberry Pi. Coleman also modified the original UART embedded implementation to make it work with the planning. He worked with Jake to write embedded code to generate PWM for the rotational and linear motors. He wrote the script, recorded, planned, and edited the final video. Along with the rest of the team, he spent much of the last two weeks of the semester in the NI lab working on immediate needs for mechanical assembly and logistics, among other things.

Jake Long:

Jake was responsible for designing motor driver circuitry, switch input protection, and photo interrupter biasing. Working closely with Coleman, he selected components, interconnects, and appropriate design strategies for the high-power driver PCB. After researching the capabilities of the MSP432 and creating an optimized header pinout, he produced the layout for the header board, again working with Coleman to complete the manufacturing and assembly process for both PCBs. He worked on hardware debugging and redesign for elements of the boards that did not work as intended. Additionally, Jake worked on the embedded code responsible for PWM generation, encoder signal handling, stall protection and recovery, and gameplay logic. Much of this was pair programmed with Aidan. He also configured the serial communication interface of the MSP432 to transmit and receive data using UART and produced a python simulator to test the MSP432 UART functionality. The final weeks before the capstone fair were spent in the NI lab working on assembly and system testing, along with the rest of the team.

Zach Yahn:

Zach was also responsible for ball detection, and he worked on a separate method at the onset of the project. This involved using OpenCV's C++ and Python libraries to apply out-of-the-box functions to ball detection, including contours, Hough Transforms, color segmentation, and more. Though Aidan's v4l2 approached proved to be much faster than OpenCV, it was still worthwhile to try both approaches in parallel, especially since OpenCV is the established standard for computer vision applications. Zach also transferred and adapted the original path planning code to the Raspberry Pi and integrated it with the vision code. Once part orders began to arrive, Zach 3D printed all of Hudson's models using the printers available in UVA's Robertson Media Center. These models required fine-tuned extrusion, infill, print speed, wall fill, raft, and heat settings to adapt to the various available 3D printer models and desired properties of the prints. Across all iterations this amounted to over 20 parts, in total requiring over 60 hours of print time. Like the rest of Single Use Smoke Machine, he spent the last two weeks of the semester assembling the project and performing various small tasks to

Table of Contents

Contents

Table of Figures

Abstract

The robotic foosball table is an autonomous system that operates one side of a foosball table to play against a human opponent. The system consists of a mechanical interface to control the foosball players, a camera and Raspberry Pi [1] to detect the ball and plan a response, and a microcontroller connected to custom-designed printed circuit boards (PCBs) to control the motors and collect sensor data. Each of these subsystems is physically connected to one another such that information regarding the state of the foosball game can be collected, processed, and converted into a desirable response like blocking an opponent shot or scoring a goal. This project is an example of robotic automation of a typically human task. While this application is for entertainment, demand for similar technologies is growing rapidly as costs fall [2].

Background

As relevant technologies continue to improve at a rapid pace, the demand for robotic systems capable of sensing their environment, drawing conclusions, and acting accordingly increases along with them. Automation capable of approximating human capabilities is especially relevant in the United States due to labor shortages in sectors like leisure and manufacturing [3]. While our project does not tackle this issue directly, it represented an enjoyable way to develop the skills and experience necessary to engineer potential solutions in the future. Additionally, robotic systems that are novel and entertaining add some subjective value as perceived by the user. There are many successful examples of interactive robotic systems that only provide value through entertainment, and the popularity of these robots has increased in recent years due to isolation induced by the Coronavirus Disease (COVID-19) pandemic [3]. Our team selected foosball specifically because competition with robots is exciting and the fast and dynamic nature of the game makes for a fascinating and challenging project.

Robotic foosball tables have been made before, often by students in formats like this capstone project. Representative examples include robots from École Polytechnique Fédérale de Lausanne (EPFL) in Switzerland [4], Brigham Young University (BYU) [5], Indiana University [6], and Western Sydney University [7]. The system created by the students at EPFL was the most powerful of these prior works; it tracked the ball using a camera with a frame rate of 300 fps, could hit the ball at 6 meters per second or more, and moved players with up to 9g of acceleration. The ball was tracked by placing the camera below the table and replacing the surface below the players with clear acrylic [5]. BYU has a similar system, however the surface below the players is unmodified and the ball position is tracked using computer vision searching for the ball's color [5].

The projects from Indiana University and Western Sydney university were less about physically actuating the foosball table, and instead focused on algorithms relevant to playing the game. The Indiana University project did not create a robotic foosball table but instead researched tracking algorithms. The input to the final algorithm was the visual feed from the camera above the table, and the goal of the algorithm was to completely describe the game play in real time – it tracked the ball position, player position, and player rotation [6] Western Sydney University also focused on algorithms, along with building a physical robotic table. Rather than using traditional neural

networks for machine learning, because of the need for quick response, they used neuromorphic vision sensors and “brain-inspired algorithms” [7].

Our project differentiates itself from these prior works due to stricter constraints on time and budget. By necessity, the various systems that comprise our robotic foosball table were built from scratch and optimized to provide an acceptable level of gameplay with cheaper parts. Our project also automated a smaller table than previous designs, in keeping with the stricter constraints.

Knowledge from throughout the Electrical and Computer Engineering (ECE) curriculum was essential to successfully design and produce a working autonomous foosball robot. In order to design the header board and driver boards, the project drew on circuit design and PCB layout skills learned in the Fundamentals of Electrical and Computer Engineering (FUN) series (ECE 2630, 2660, 3750). Skills from the Embedded Computing and Robotics (ECR) classes (ECE 3501 and 3502) were necessary for using the MSP432 microcontroller [9], writing embedded code, and handling Input/Output (I/O) for sensors and motors. Algorithm design utilized general concepts in Computer Science (CS) classes like algorithms and machine learning (CS 4102 and 4774). In order to make the Raspberry Pi [1] deterministic, the project applied learning from operating systems, computer architecture, and ECR in order to modify Linux [10] and use interrupts (CS 4414, ECE 4435, ECE 3501 and 3502). For the mechanical designs, Hudson applied skills learned in mechatronics and advanced mechatronics (MAE 4710 and 4720). In addition to information from courses listed above, members of Single Use Smoke Machine made use of experience from internships and extracurricular activities related to the task. This includes experience with various computer science disciplines, embedded programming, sensor applications, and robotics.

Physical Constraints

Design Constraints

The MSP432 [8] needs to control many hardware devices during their fast-paced movement. The clock speed is set to its maximum of 48 MHz. It needs to generate PWM signals and record the encoder counts for all four motors along with managing 4 limit switches and 2 optical interrupters. Optimization in the interrupt handling was required to achieve the desired performance with the speed limitations of the CPU. This represented a trade-off, where a faster processor would allow for more complexity in our gameplay algorithms.

All programming that was used in the system was done in C which is an open-source language that does not require licensing. Some licensed CAD software tools were used in the project, but licenses were available through the university, so availability was not limited. Python and Matlab were used for design and analysis; Python is open source and Matlab licenses are provided by the university.

The printed circuit board (PCB) manufacturing capabilities for this class did not include plated slots. Due to the relatively large amount of power needed for our application, this limited the selection of power interconnects and switches available rated for sufficient current and voltage.

Surface mount replacements were used where possible but resulted in durability concerns due to the frequency with which the interconnects were used.

As expected, part shortages played a significant role during the hardware design process. If our capstone adviser had not been able to provide the motor driver integrated circuits (ICs) used in our project, finding replacements with similar specifications would have been challenging. The ideal components for less pivotal roles were also frequently out of stock – shrouded interconnects, high-force tolerant limit switches, and isolator ICs all required a degree of compromise in our design.

Cost Constraints

The class budget was $500, and the initial expected cost distribution can be seen in Table 1. Many parts were already owned, like the Raspberry Pi [9], MSP432 [10] and camera, and other additional parts were purchased separately.

Table 1: Expected Costs

Motors & Encoders	$173.00
Mechanical Assembly	$131.00
PCBs and Electrical	$176.00
Emergency	$20.00
Total	$500.00

The actual budget use can be seen in Table 2. PCBs were more expensive than anticipated, mostly because of assembly costs. The mechanical assembly was also more expensive than anticipated given the frame below the table was not initially accounted for.

Table 2: Budget Use Overview

Mechanical	$74.40
Motors	$90.85
PCBs	$266.13
Total	$431.38

In addition to the class budget, additional materials (mostly mechanical) were purchased and can be seen in Table 3.

Table 3: Additional Costs

Mechanical	$291.35
Motors	$133.85
Foosball Table	$41.99

Other Electrical	$57.35
Total	$524.54

Tools Employed

We made use of several tools and technologies throughout the development of the robotic foosball table. In this section, we list those tools, explain the role they played in our work, and expand upon how we improved our skills with them.

Software

1. *C* [11]. This is the primary programming language for our project. Nearly all systems, including image processing, path planning, and embedded were written in C. All group members were comfortable with C from prior coursework and projects.
2. *Linux* [12]. Giving our image processing the highest priority on the Raspberry Pi's CPU required an in-depth exploration of the Linux Kernel. Though we were all familiar with Linux beforehand, solving this problem pushed our skills further.
3. *PREEMPT_RT Linux Patch* [13]. PREEMPT_RT is a modification to the Linux kernel source code that makes the entire kernel preemptible by user processes. We applied this patch to the Raspberry Pi's Linux kernel to ensure deterministic real-time performance of the image processing and planning code. No group members had patched a kernel before, so we needed to learn how to apply the patch to the source code and configure and compile the kernel.
4. *Video for Linux 2 (v4l2)* [14]. This is how images were retrieved from the camera file descriptor. No group members were familiar with it prior to this project, so implementing it required extensive research.
5. *Yet another YUV viewer (yay)* [15]. This is a simple open-source tool for viewing images in the format that our camera outputs them. We used and extensively modified it to view the output of the image processing for debugging and calibration. No team members had used it, so we needed to learn it from scratch.
6. *Simple DirectMedia Layer (SDL)* [16]. SDL is a software library for displaying images to a window and receiving user input from a keyboard and mouse. It is used in the implementation of yay, so using and modifying yay required use of SDL. No team members had used it before, so we needed to learn the basics of the interface from scratch.
7. *Python* [17]. We created a UART simulator for testing communication between the Raspberry Pi and MSP432. All group members were very familiar with Python prior to beginning the project.
8. *Matlab* [18]. Some image processing calculations were performed with Matlab scripts, including generating graphs from our early game trials. All group members were very familiar with Matlab prior to beginning the project.

Embedded

1. *Code Composer Studio* [19]. All embedded code was written, debugged, and flashed via Code Composer Studio. All group members were familiar with the software from previous classes, especially the ECR series.

Hardware

1. *KiCAD* [20]. All hardware design was completed using KiCAD. Most group members were not familiar with KiCAD prior to this project, so it presented a significant learning opportunity.
2. *FreeDFM* [21]. Hardware was prepared for circuit board manufacture by running it through the standard checks on FreeDFM.com. All group members used FreeDFM in multiple classes beforehand, notably the FUN series.

Mechanical

1. *Fusion360* [22]. All 3D printed parts were designed with Fusion 360. This includes the rotational and linear motor mounts, shaft couplings, limit switch mounts, and other pieces. Some group members had extensive experience with this beforehand.
2. *Cura* [23]. 3D parts were sliced and prepared for printing using Ultimaker Cura, which generates gcode that the printers can interpret. It also allows the user to select printer settings, which were essential for determining the quality and speed tradeoffs of the prints.
3. *Ultimaker Printers* [24]. The UVA Robertson Media Center offered free 3D printing with Ultimaker S3, Ultimaker 3, and Ultimaker S3 Extended printers. Some group members had extensive experience with these printers beforehand, including having the necessary training to use them.
4. *MakerBot Printers* [25]. The Robertson Media Center also offers 3D printing with MakerBot Replicator+ printers, which were used alongside the Ultimakers. Some group members had extensive experience with these printers beforehand.
5. *Waterjet* [26]. Lacy Hall provides a waterjet that students can be trained on. We used this to cut steel brackets for fastening the foosball table to the aluminum frame. None of our group members had experience using this machinery before, presenting another learning experience.

Societal Impact Constraints

Environmental Impact

Several of the components involved in the system would represent environmental concerns if the device was produced at scale. The microcontroller and Raspberry Pi [1] require various mined raw materials and substantial water consumption to produce [27]. The proposed motors and other components must also be produced and have a resulting environmental impact.

Now that the project has concluded, the parts will eventually need to be disposed of. For the electronic parts, they will create e-waste which is an international problem because of the contaminants they contain (like lead and mercury) and the health effects that can have to those exposed [28]. The non-electronic parts will also have to be disposed of but carry a much less remarkable environmental cost than the parts described above.

Since our project makes use of so much 3D printed material, it is important to also consider how these plastics affect the environment. One study showed that the plastics used in 3D printing can become microplastics that pollute natural habitats and endanger wildlife. The same study also showed how the energy usage of 3D printing technology contributes to climate change [29]. Another study recommends using recyclable or biodegradable plastics and to optimize the printer to reduce idle time, however given our limited control of the 3D printing technology available for this project these efforts were not possible [30].

Sustainability

Even though the project will be deconstructed at the end of the semester, it is important to consider how sustainable it might be if produced in another context. The robotic foosball table is constructed from a mixture of parts with varying longevities. The steel brackets and aluminum frame, for example, are highly durable and likely to last for decades [31]. 3D printed components, on the other hand, are liable to wear down much faster. The foosball table itself is also susceptible to wear and tear. Even in a single semester, parts of the wood chipped, and the metal rods bent. Users of this product might feel the need to purchase it every few years, driving up the environmental impact of the design.

Health and Safety

Foosball play typically involves close interaction with the table, meaning that this device must move at speeds that will not harm the users. The motors are controlled so that reaching into the table to pick up a dead ball within range of a motor-powered component does not pose a risk of injury. The lateral movement of the rods is also a concern, since it is possible for the device to jab a human player. To prevent this, limit switches were installed on the side of the table to prevent the rods from extending to a distance where they might injure a player or spectator. Furthermore, each limit switch mount includes a mechanical stop to prevent the motor from extending far enough to jab a player, keeping the motion safe even if the limit switches fail. See section below on safety standards for official standards.

Ethical, Social, and Economic Concerns

The major ethical issue with this device is the justification of the environmental cost and expended resources to produce a device used solely for entertainment. Various expensive components, both monetarily and environmentally, were consumed to accomplish the design. This concern was offset by the academic value of the project to the members of Single Use Smoke Machine, as well as the social value that entertaining devices provide. Were the device to be produced at a larger scale, the balance of this concern and justification would shift, and the ethics of the device would have to be revisited.

We also consider whether this project is accessible for all potential users. Admittedly, the foosball table is not playable if one is not tall enough to reach the handles. This means that young children may not be able to enjoy it, nor will people who are not able to stand next to the table for extended periods of time. It is possible that a shorter project table might alleviate this problem, especially if it was low enough that one could sit while playing.

External Considerations

External Standards

The table contains a potential electrical hazard in the PCB header board and microcontrollers, and a potential mechanical hazard in the motors and belt system. To protect the device and its users, these parts were contained in an enclosure satisfying National Electrical Manufacturers Association (NEMA) Type 1, which specifies protection “against access to hazardous parts” and “ingress of solid foreign objects” [32]. The table is not meant to be operated outdoors or near liquids or other hazardous materials, so no higher NEMA type is required.

During board design, standards from the Institute for Printed Circuits (IPC) were followed to ensure that the boards were safe, effective, and manufacturable. These standards are outlined in IPC-2221[33].

While not meant for industrial applications, this product makes use of several motors and moving parts that could cause harm to the user if not properly guarded. This guarding will be pursuant to OSHA in 1910.212 which requires machine guarding from hazards that may occur due to rotating parts, points of operation, etc. [34]. In this case, the user is guarded from the actuation of the motors moving the rods linearly and rotationally.

The table also complies with several sections of the National Fire Protection Association (NFPA) National Electrical Code (NEC) [35]. The wiring connecting the motors and sensors to the header board are protected by overcurrent as specified by Article 240.5 and Table 400.5(A)(1). Given our use of 14 American Wire Gauge (AWG) wire, the system can support a maximum of 18.75 amps of current [33]. These wires are mounted to the system such that no pull is transmitted to the wire joints as required by Article 400.14, and they are protected from damage by contact with the system as required by Article 400.17. The insulation of the wires connected to ground show a white or gray tracer or solid color as required by Article 400.22. The motors were also considered when designing the wiring and overcurrent protection. Table 430.247 provides full-load current ratings for DC motors, and Article 430.22 specifies requirements for the wiring size, depending on the motors’ current ratings.

Finally, the Barr Embedded C Coding Standard was followed when developing embedded software [36]. This standard ensures that code is readable, maintainable, and less error prone. It also outlines some best practice naming conventions and debugging techniques which were referenced throughout the software development process.

Intellectual Property Issues

Several existing patents cover parts of this project. For example, a patent for a Broadcast-ready Table Sports System has claims to “a plurality of cameras spaced apart from one another about a detection region” where the detection region is “a table having a tabletop defining a playing surface for supporting a movable object” [37]. These claims are independent, because they do not reference prior work that they further limit. These claims also encompass the broad scope of our project, where a movable object can be interpreted as a ball. Though our project only uses one camera, we expect that this is not novel enough to make a distinction.

Other patents also claim inventions that overlap with this project, notably Camera-based Tracking and Position Determination for Sporting Events using Event Information and Intelligence Data Extracted in Real-time from Position Information. This patent specifically mentions “image processing . . . as an object localization technique.” It is applicable to “gaming practices, filming movies, or other live events that wish to be captured with sophisticated comprehensive coverage” [38]. It is worth noting that these independent claims largely pertain to large sporting events, for example soccer or basketball, that require multiple camera angles. However, it is possible that foosball, an official sport, could count.

In terms of our object tracking software, a patent concerning a Probabilistic Object Tracking and Prediction Framework might be relevant. This patent applies to dynamic object tracking for autonomous vehicles, though the claims extend to probabilistic path planning for dynamic objects in general [39]. This patent illustrates how our project might differentiate itself from prior art. Our project does not use probabilistic planning, but rather computes a single expected location based on the current state. Given the emphasis on generating a distribution over possible future states in this patent, we believe that our prediction method is novel enough.
Considering these three relevant patents, we do not believe that our robotic foosball table presents enough novelty for its own patent. The systems presented in the first two patents have too much in common with our own, especially claiming cameras for tracking balls in a sports setting [37] [38]. Though it may have some distinct characteristics from these works, especially compared to the third patent, it is most likely insufficient for a claim to a new filing.

Detailed Technical Description of Project

We break the robotic foosball table project down into five subsystems: ball sensing, path planning, embedded code, hardware, and mechanical assembly. Each of these subsystems is further dichotomized into detailed processes and operations. An overarching diagram of how these systems connect is found in Figure 1.

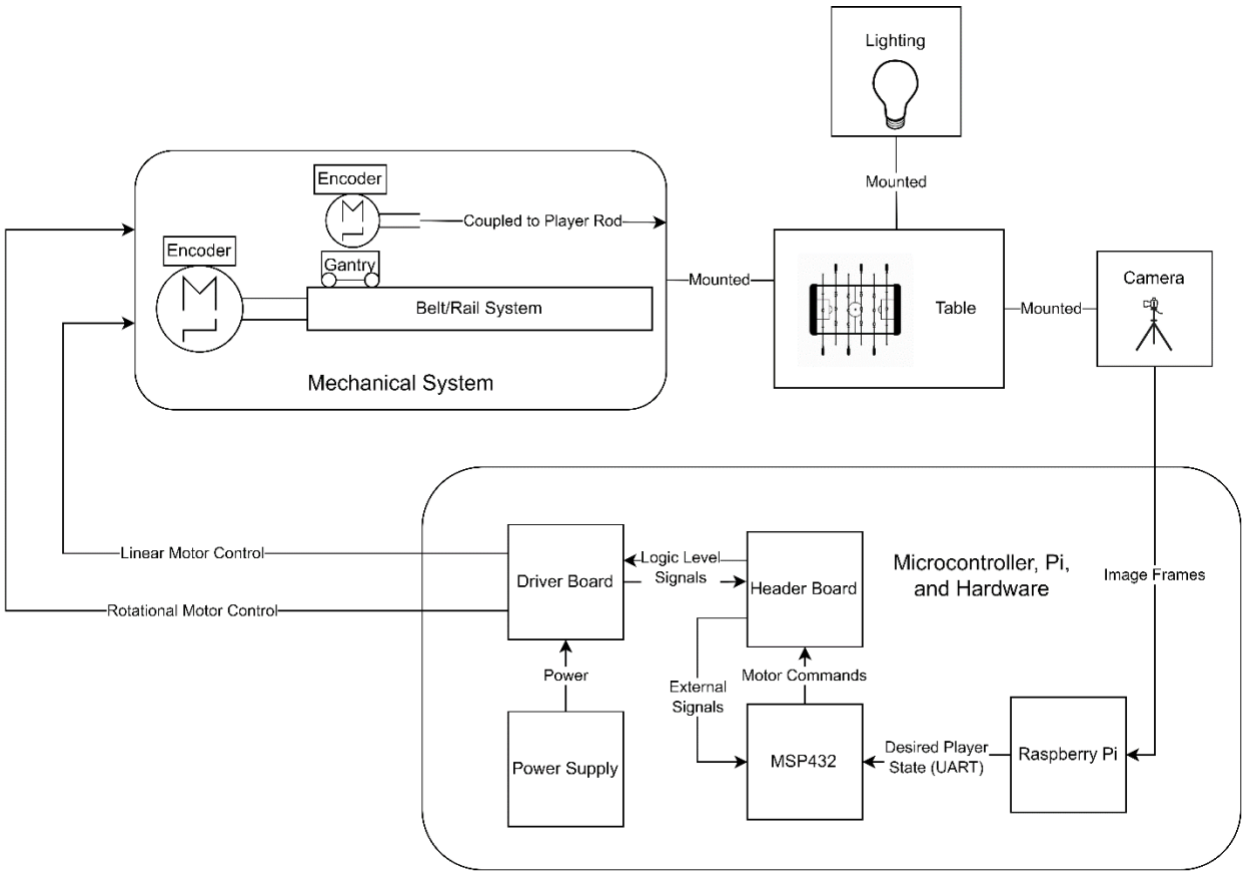


Figure 1: System Diagram

Ball Sensing

Dataflow through the system begins with ball detection on the Raspberry Pi [9]. The Pi receives image frames via USB (Universal Serial Bus) from the camera. The camera captures 640x480 pixel images at 30Hz with a YUV (Luminance, Chrominance blue, Chrominance red) color format [40]. These frames are read into memory with v4l2 (Video for Linux 2) [14]. Processing uses two memory buffers: while one reads in the next frame, the other is used for processing the current frame, with the two switching roles each time a new frame is read. Each frame passes through a pipeline of functions that extract useful data.

First, the image passes through a loss function that examines each pixel independently. For each pixel, the loss function computes the absolute value of the difference between the pixel's Y, U, and V values and corresponding target Y, U, and V values, set by the color of the ball. The differences are summed to produce a single loss value that represents how similar the pixel is to the ball color, where low loss values represent pixels similar to the ball and vice versa. These loss values can be interpreted as pixels in a greyscale image and viewed at runtime; Figure 2 shows a typical image of the table before any image processing and Figure 3 shows the same image after application of the loss function. To make the behavior of the loss function easier to see, a contrast booster can be applied to the loss function at runtime, which inverts the loss values such that similar pixels to the ball are bright instead of dark and increases the difference

in brightness between loss values. Figure 4 shows the same image of the table after application of the contrast booster.

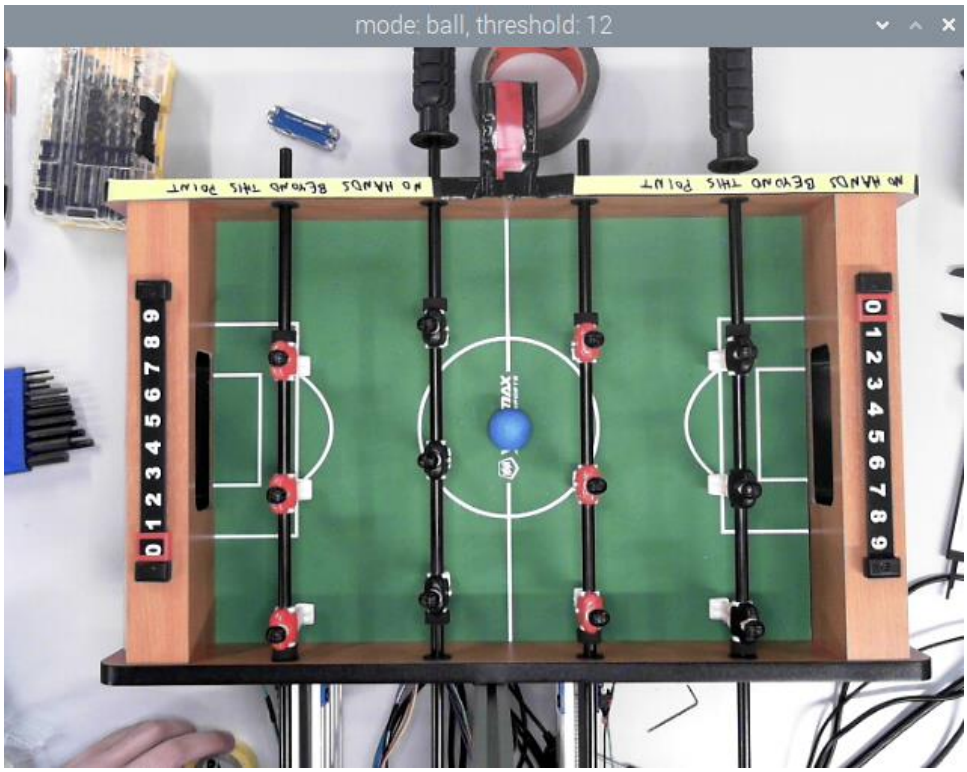


Figure 2: Image of the Table Before Processing

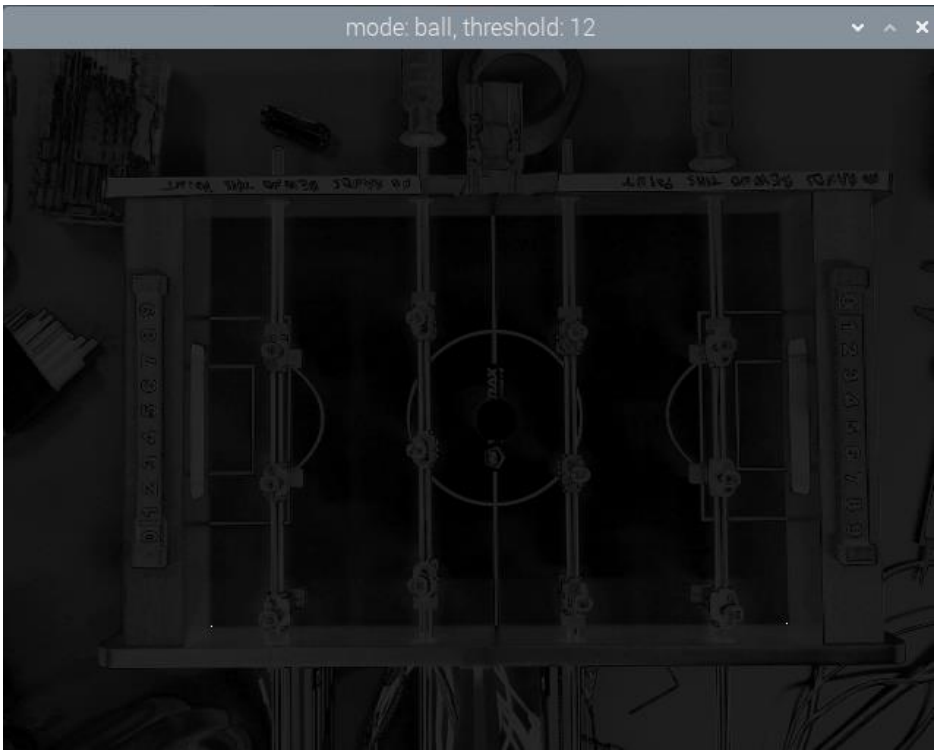


Figure 3: Image of the Table After Loss Function

Figure 4: Image of the Table After Loss Function and Contrast Booster

Second, the original image is also passed through a function that finds the bottom-left and bottom-right corners of the playing field, in order to find the position of the ball relative to the table in a later processing step. This function starts with a user-defined estimate of where in the image the corners of the field are, then searches a square of a set size (in number of pixels) around each estimate. For each pixel in the search region, the function performs a computation analogous to the loss function, comparing the pixel's YUV values to a set of target values tuned to the color of the playing field. The function compares the loss to a threshold that can be adjusted at runtime, and considers all pixels with loss under the threshold to match the color of the playing field. Finally, the function finds the bottom-leftmost matching pixel in the bottom-left search region and the bottom-rightmost matching pixel in the bottom-right search function and sets the position of the corners of the playing field to the coordinates of those pixels. For debugging and calibration, the function also sets the loss value at the corners to the maximum value, so that the corners appear as white dots in the loss image. These white dots can be seen at the bottom-left and bottom-right of the playing field in Figure 3.

Third, the loss image is passed through a function that simultaneously decides whether or not a ball exists in the image and finds the center of the ball if it exists. This function also takes in the results from the corner-finding function and uses them to first calculate the region of the image that corresponds to the playing field, so that objects outside the playing field that match the color of the ball do not interfere with the calculation. Then the function iterates through each pixel in the region, comparing the loss value of each to a threshold value adjustable at runtime, where pixels with losses under the threshold are considered part of the ball. The function keeps a running count of matching pixels, and a running sum of the x and y coordinates of matching pixels. After checking all pixels, the function compares the total count of matching pixels to a pre-set threshold. If there are less pixels than the threshold, it indicates there is no ball in the image. Otherwise, it indicates there is a ball and sets the ball coordinates to the mean x and y coordinates of all matching pixels. For debugging and calibration, the function also sets values of another greyscale image: black for pixels outside the playing field, dark grey for pixels inside the playing field, light grey for ball pixels, and white for the center of the ball. Figure 5 shows the resulting image, again corresponding to the original image in Figure 2.

Figure 5: Image of the Table After Ball-Finding Function

Fourth, the results of the ball-finding and corner-finding are used to calculate the position of the ball relative to the table. The program also records the position of the ball on the previous frame and uses it to calculate the ball's velocity. In the event that the ball is not found in the image, the ball's previous position and velocity are used to estimate its position on the current frame. This extrapolation is performed for a maximum of 3 consecutive frames, after which the ball-detection program decides that it does not know where the ball is. Finally, the position of the ball relative to the table and a flag indicating whether or not the ball is found are passed to the path planning subsystem.

Many of the parameters to the ball-detection algorithm are configurable at runtime by a mouse and keyboard interface and the screens shown in Figure 2 through Figure 5. The full controls to the interface are enumerated in Table 4. The program saves these settings to a file when exiting and reads them back in on startup.

Table 4: Controls to Image Processing Interface

Input	Function
Left Click	Set the target color of the ball, the bottom-left corner, or the bottom-right corner, depending on the mode set by the B, R, and L keys.
Right Click	Print out debug about the clicked pixel including the x and y coordinates, YUV color values, and loss value.

Left / Right Arrow Keys	Switch between viewing the original unprocessed image, the loss image, and the post-threshold ball-detection image.
Up / Down Arrow Keys	Adjust the value of the corner loss threshold or ball loss threshold up and down, depending on the mode set by the T and E keys.
Q	Terminate the program and send a shutdown signal to the MSP.
B	Set the left click mouse button to set the ball target color.
R	Set the left click mouse button to set the bottom-right corner target color and position estimate.
L	Set the left click mouse button to set the bottom-left corner target color and position estimate.
T	Set the up and down arrow keys to adjust the corner loss threshold.
E	Set the up and down arrow keys to adjust the ball loss threshold.
0	Disable the loss contrast booster.
Number Keys 1-5	Enable the loss contrast booster, with increasing strength from 1 to 5.
Space	Disable the image output to the screen. This control exists because outputting to the screen is a costly operation that slows down the image processing, so it is disabled during gameplay.
Enter	Send a command to the MSP to exit the waiting state and start gameplay.
W	Send a command to the MSP to exit gameplay and enter the waiting state.

Path Planning

The input to the path planning algorithm is the output of the ball detection: ball position, ball velocity, and a flag that indicates whether a ball was found in the image. First, the planning code checks whether the ball position is within the physical bounds of the table to ensure the image processing did not mistakenly identify an object outside the table as a ball. If the ball is out of bounds, or if there was no ball detected, the planning code instructs both the offense and defense rods to return to a default position in the middle of the table in a blocking position.

If the planning receives a valid ball position, the first step is to decide which general region of the table the ball is in relative to each rod by comparing the ball's x position to the known x position of the rod. Figure 6 shows the regions relative to the offense rod. If the ball is behind the rod by farther than the players can reach, it is in the Ready (green) region, and the players must be rotated horizontally so a shot from the defense rod will not be blocked by the offense rod. If the ball is ahead of the rod by farther than the players can reach, it is in the Block (blue) region, and the players must be rotated straight down to block a shot from the opposing team. If the ball is within reach of the players, it is in the Shoot (red) region, and the players must kick the ball

towards the opposing team's goal. An exception is made if the ball is in the Shoot region, but moving quickly towards the rod, in which case the rod must block instead of attempting to shoot. The logic is identical for the defense rod, but with the regions being defined in relation to the defense x position instead. The planning algorithm is capable of differentiating between a ball in the Shoot region that is in front of the players versus behind them, so that the players can avoid kicking a ball backwards while winding up. However, due to time restrictions, this was not implemented in the control algorithm, so functionally, there is only one Shoot region.

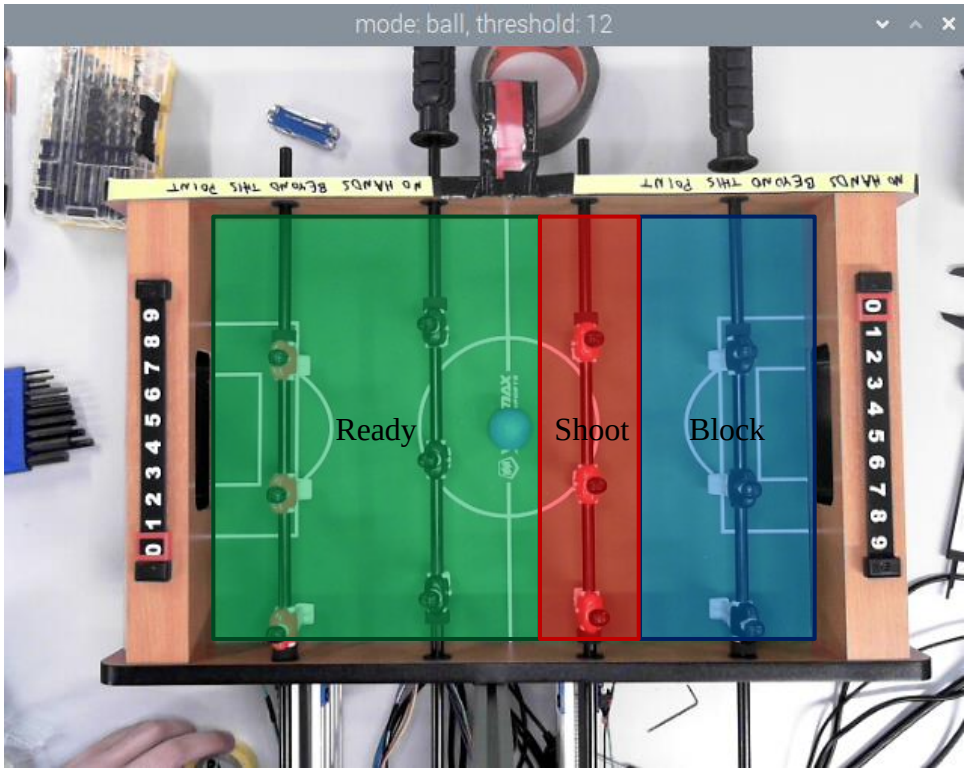


Figure 6: General Regions of the Table Relative to the Offense Rod

After deciding the general region, the planning algorithm calculates the y position to move the players to. In the Shoot region, the players aim at the current y position of the ball. In the Block and Ready regions, the target y position depends on the ball's velocity. If the ball is moving quickly towards the rod, the planning algorithm uses the ball's current position and velocity to extrapolate the ball's path and calculate the y position at which the ball will reach the rod, assuming the velocity remains constant. Otherwise, the target y position is simply the current y position of the ball.

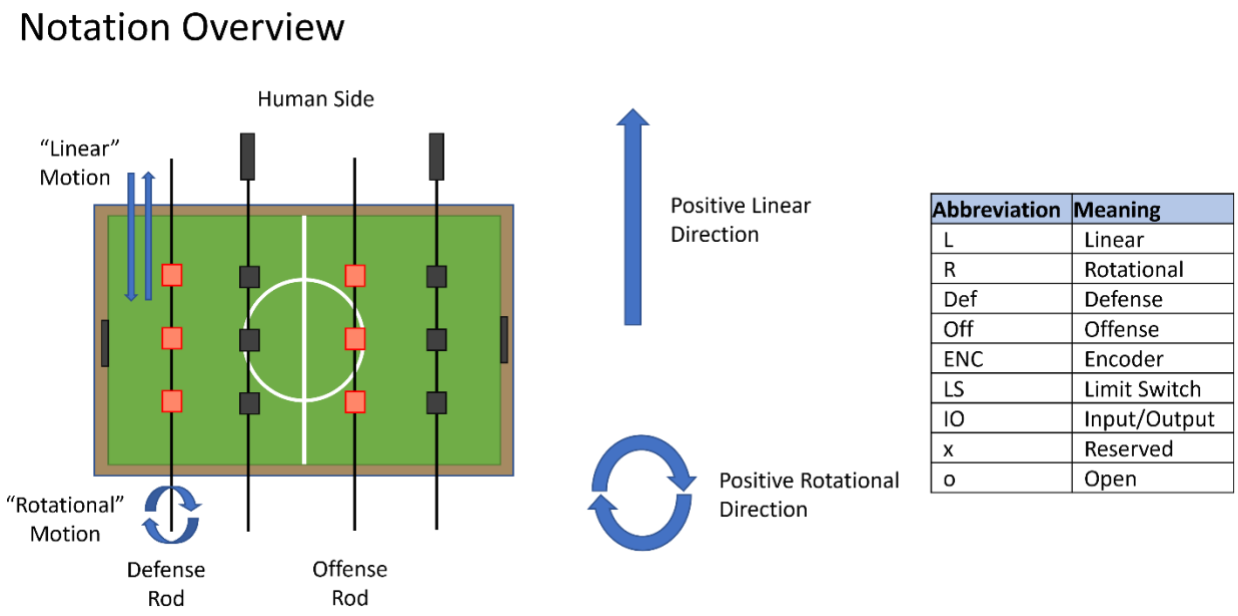
To move the players to a target y position, the planning code must first decide which player on the rod to use. The program uses the physical distance between each player on a rod and the physical distance the rod can move to calculate a range of y values that each player can reach. The planning code records the last player used for each rod, and first checks whether the target y value falls within that player's range and selects that player if it does. If not, it checks the remaining player ranges to find one that can reach the target y value, favoring the center player.

Once a player has been selected, the code calculates the distance the rod must move from its home position by subtracting the target y position from the selected player's base y position.

Finally, the planning code must send a message over USB to the MSP432 containing the desired y position and rotational action state (block, ready, or shoot) for each rod. To do so, it needs to convert the desired y values from millimeters to motor encoder counts. In the initialization of the planning code, it waits for the MSP432 to finish its encoder calibration routine and reads in the maximum encoder counts returned from the MSP432. Then, in the planning loop, it creates a conversion ratio from the maximum encoder counts and the maximum physical travel of the rod in millimeters, which it applies to the desired y positions to calculate desired encoder counts. For each rod, it transmits the desired encoder count over 3 bytes and the rotational state over 1 byte, as explained in further detail in the next section.

Embedded Code

By necessity, a standard naming convention was established to represent the table and game, both conceptually and in software. An overview of this naming system is shown in Figure 7. These names will appear throughout the embedded, hardware, and mechanical sections.



Abbreviation	Meaning
L	Linear
R	Rotational
Def	Defense
Off	Offense
ENC	Encoder
LS	Limit Switch
IO	Input/Output
x	Reserved
o	Open

Figure 7: Foosball Naming Convention

The embedded code structure consists of a main gameplay super-loop and interrupt handling for external signals and timers. The MSP header pins and corresponding IO signals are shown in Table 5.

Table 5: Embedded Pinout for MSP

PORT	BIT 0	BIT 1	BIT 2	BIT 3	BIT 4	BIT 5	BIT 6	BIT 7
1	X	x	x	x	x	o	RDef_ENC_A	RDef_ENC_B
2	X	x	x	o	o	o	ROff_ENC_A	ROff_ENC_B
3	O	x	o	o	x	o	LDef_ENC_A	LDef_ENC_B
4	RDef_IN1	RDef_IN2	ROff_IN1	ROff_IN2	R_SLP	T_OFF	LOff_ENC_A	LOff_ENC_B
5	RLSDef	RLSOff	o	x	LLSDef1	LLSDef2	LLSOff1	LLSOff2
6	LDef_IN1	LDef_IN2	x	x	LOff_IN1	LOff_IN2	LDef_INH	LOff_INH

*IO suffix omitted from pin names for readability

Interrupts are generated by UART transmit (Tx) and receive (Rx) signals, rising and falling edges on the encoder signals, the stall watchdog timer, and the PWM generation timer.

UART

The enhanced universal serial communication interface 0 (eUSCI0) on the MSP is configured in UART mode with a baud rate of 9600. The interrupt register of the eUSCI is configured to trigger interrupts when new data is loaded into the Rx register, and the flag is manually set when loading data into the Tx register. Subsequent outgoing data in the Tx register will reset the flag until no data is left to send. Upon entering the shared Tx and Rx interrupt service routine (ISR), the interrupt flag is first checked to determine whether Tx or Rx should be serviced. The MSP transmits data to the Pi infrequently, sending the max linear encoder count ranges following a calibration sequence. The ISR Tx handling sends the calibration data from a buffer one byte a time, sending first the linear defense max encoder range then offense. During gameplay, the Pi constantly sends the desired game state data in frames, shown in Figure 8. This information is processed and stored by the ISR Rx handling.

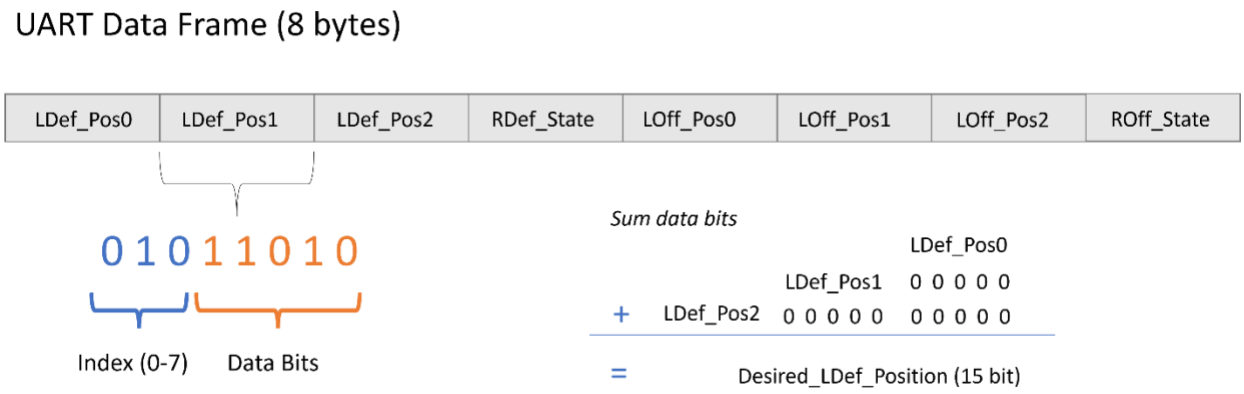


Figure 8: Structure of UART Protocol from Pi to MSP

Each byte in the frame has a three-bit index which indicates what information is contained in its five data bits. Three bytes of the frame are allocated for each desired linear motor position, measured in encoder counts. The new position will only be stored if all three bytes for that position are received in order, and the final calculation for the new position is shown in Figure 8. The desired states for the rotational motors are each sent in a single byte, where the data bits map to a rotational state enumerated type (enum). ROff_State, byte position seven of the frame indexed from zero, can also be used to send magic number sequences to the MSP. These are checked separately by the Rx ISR and set the main state of the MSP manually, Figure 11. A full, successfully transmitted data frame provides all the information needed in the main gameplay loop to achieve the desired game state calculated by the planning algorithm.

PWM Generation

Interrupts from Timer A0 of the MSP are used to generate four simultaneous PWM signals for motor control. Each of these four signals must also be capable of switching between two pins, to run the motors in both forward and reverse. Timer A0 is configured in up mode, set to compare mode, and uses the sub main clock (SMCLK) which runs at 12 MHz. The timer interrupt registers are configured such that two unique interrupts occur – one when the timer counts to capture control register 0 (CCR0) and one when the timer count reaches a value in CCR one to four (CCRn). This results in the desired PWM period of 1ms.

A value proportional to the desired duty cycle (from 0-11,998) for each motor is loaded into a corresponding CCRn register of the timer. When the CCR0 interrupt is triggered, the current PWM output pins for the motor are set high. When the CCRn interrupt is triggered, the ISR checks which register caused the interrupt and sets the output corresponding to that motor low. A special case checks if the duty cycle is zero and never sets the output for that motor high. Figure 9 shows a simplified PWM generation with one signal highlighted. Output pin switching to support direction is handled by bitwise logic using variables defined outside the ISRs – this minimizes the computation occurring within the ISRs. Table 6 outlines which CCRs and pins correspond to each motor and direction.

Table 6: PWM CCR and Output Pins

Motor	CCR Index	Forward MSP Pin	Reverse MSP Pin
LDef	1	6.0	6.1
LOff	2	6.4	6.5
RDef	3	4.0	4.1
ROff	4	4.2	4.3

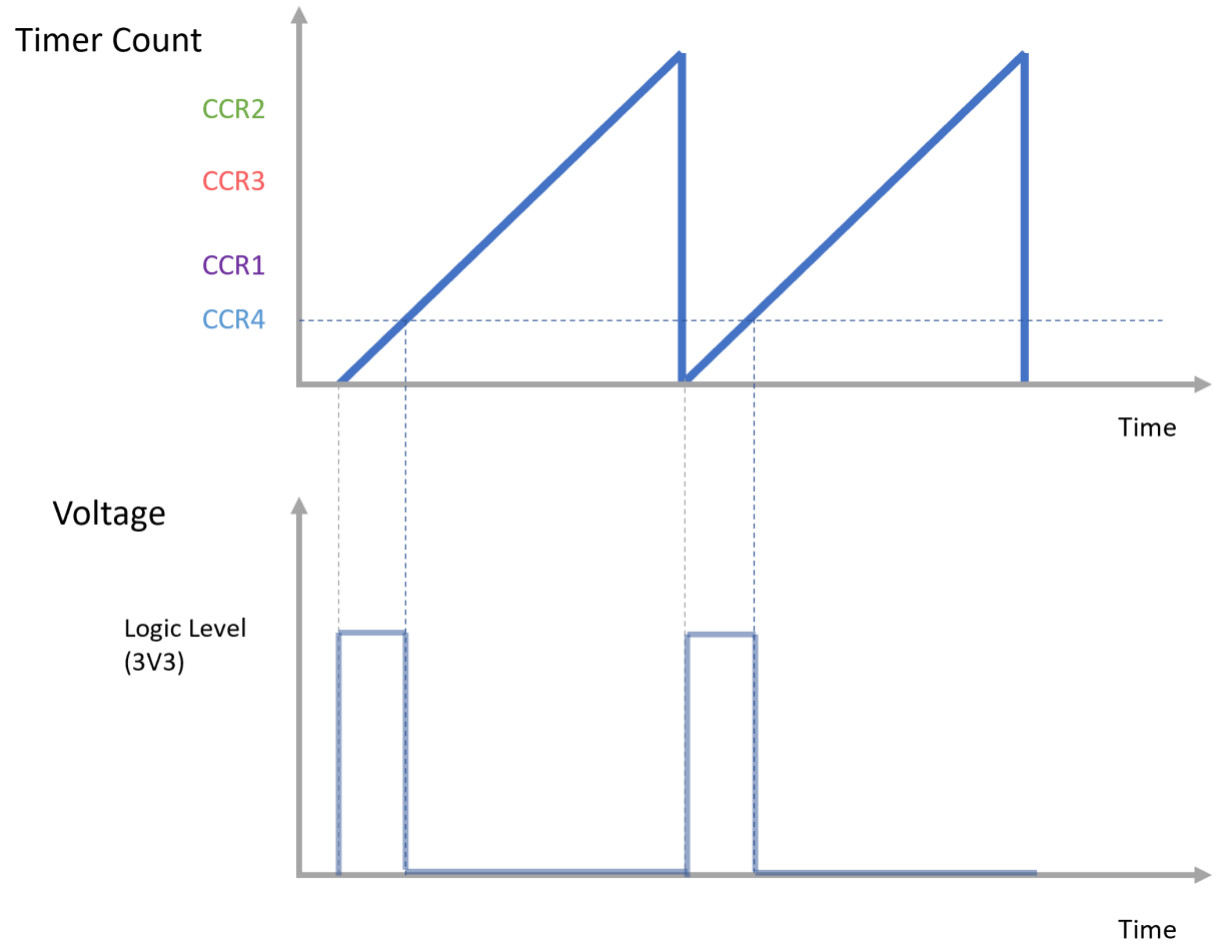


Figure 9: PWM Signal Generation

Encoder Signals

After processing, pairs of logic level encoder signals for each motor connect to pins 6 and 7 of ports 1-2 (rotational motors) and 3-4 (linear motors) of the MSP. An interrupt is triggered on each port for both rising and falling edges, and the encoder count for the motor at the respective port is incremented or decremented. Figure 10 shows the stateful logic used to determine the encoder count operation.

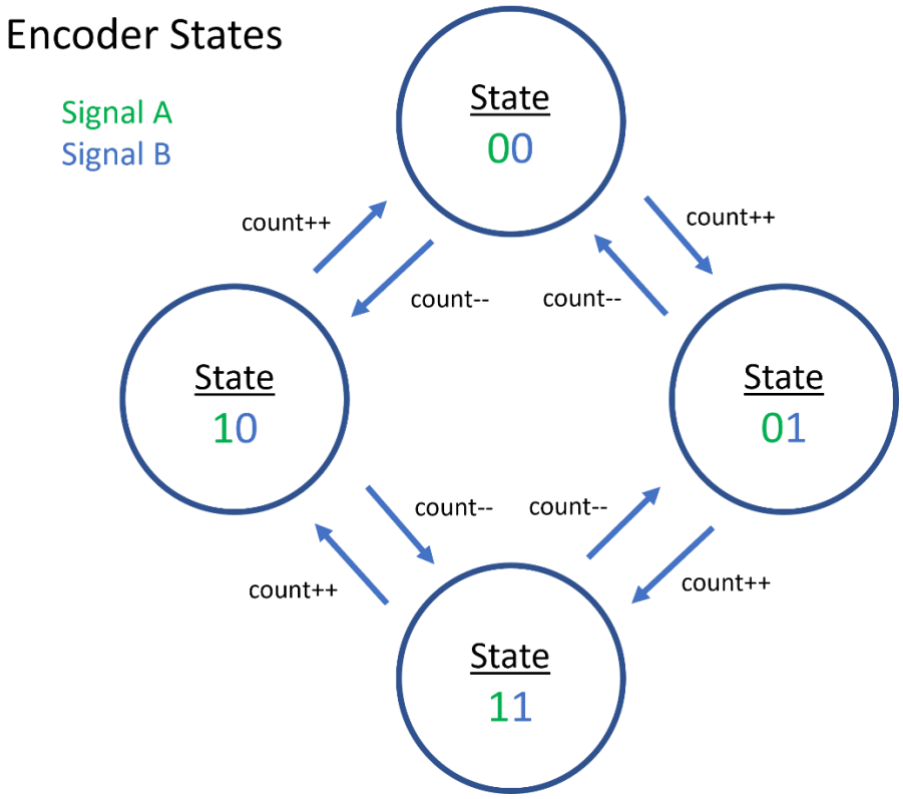


Figure 10: Encoder States

For the particular MSP model used in the project, general purpose input output (GPIO) interrupts triggered by a given pin are supported for either rising or falling edges but not both. The encoder state logic requires that each edge on both signals is counted – to overcome this, bitwise toggling of the interrupt edge select (IES) register was performed using the XOR of the previous and new encoder states. For example, upon receiving a rising edge on signal A, this operation will toggle IES such that a falling edge on signal A will now be caught, but the behavior for B is unchanged. Because these encoder signals can occur at up to 10 kHz at max motor speed, the encoder ISR was optimized and tested for speed. On average, the ISR runs in approximately 2.9 µs. With all four motors running at max speed, encoder ISR handling will require approximately 11.6% of the MSP processing time, which is acceptable as a worst-case condition.

Stall Watchdog Timer

Timer A1 was configured in up mode using the SMCLK with a divider of 8. CCR0 was loaded with the value 30,000 such that the timer produces interrupts every 200 ms. Within the CCR0 ISR, if a motor has a positive duty cycle the current encoder count is compared to the previous encoder count (at the time of the last stall watchdog ISR execution). If the current and previous counts are within some tolerance, a stall count variable is incremented. If the stall count variable reaches 10, the motor is stopped and the main state of the MSP transitions to stall recovery. If the current and previous encoder counts are more than the tolerance apart, the stall count is reset. The current encoder count for each motor is then saved to be used in the next check of the watchdog timer. This system essentially checks if a motor is being supplied power but is not

moving, stopping the motor before damage occurs. Because motors regularly get stuck on the ball during gameplay, having a system that stops stalled motors and sets a flag for a recover routine is essential.

Main State Super-loop

While not servicing interrupts, the MSP is executing within a main super-loop. The behavior of this super-loop depends on a main state, depicted in Figure 11. Simple, tunable positional control functions were implemented to move motors to desired positions.

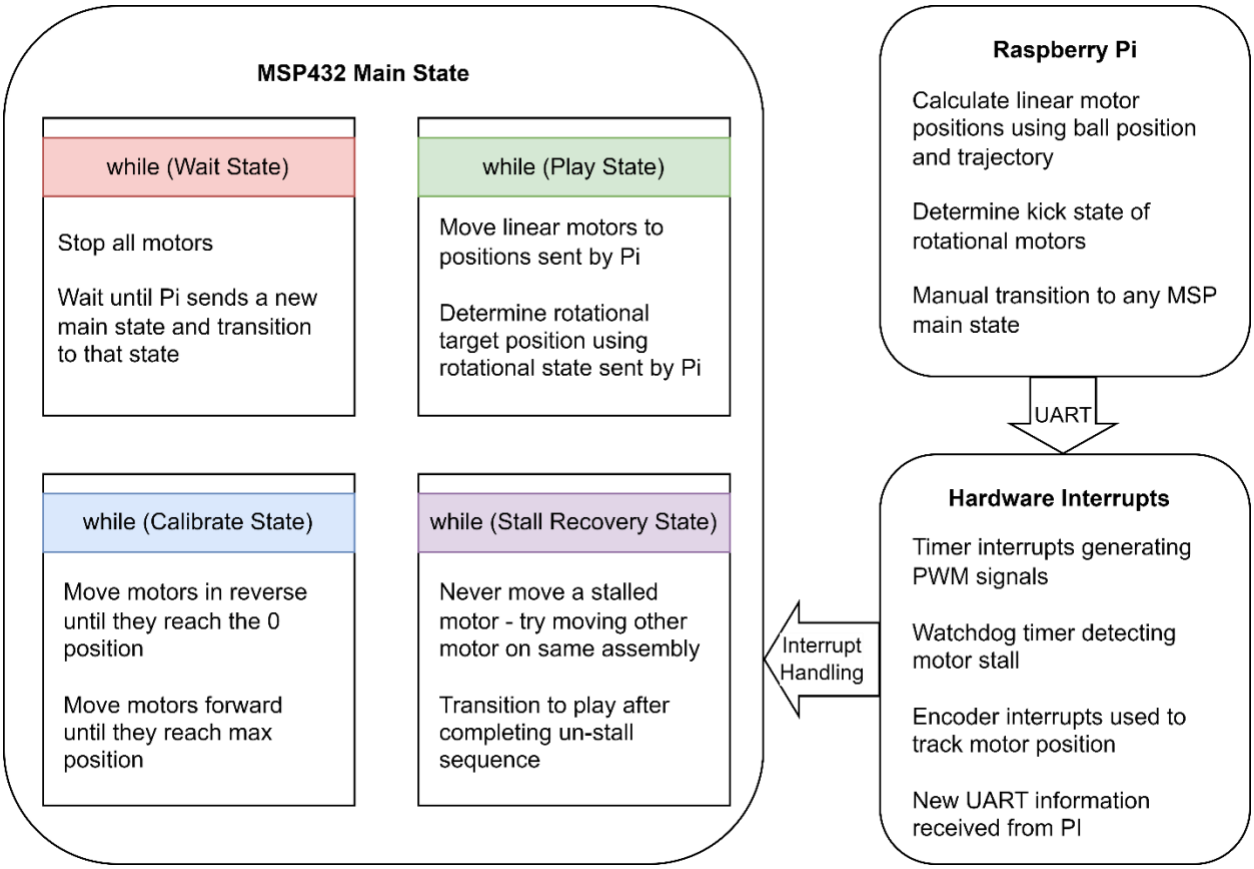


Figure 11: Main State Embedded Code Structure

While in the wait state, all motors are turned off and the MSP awaits a signal from the Pi indicating which state to transition into. In the calibrate state, all motors move in reverse until activating their corresponding zero position switch. The encoder counts are set to zero upon reaching this position, and the motors are run in reverse. Upon hitting the max position switches, the current encoder values are saved as the max ranges. The motors then return to a default state and the MSP transmits the linear max encoder counts to the Pi over UART. While in the play state, desired state information transmitted by the Pi is used to inform motor control. An expanded view of the play state is shown in Figure 12.

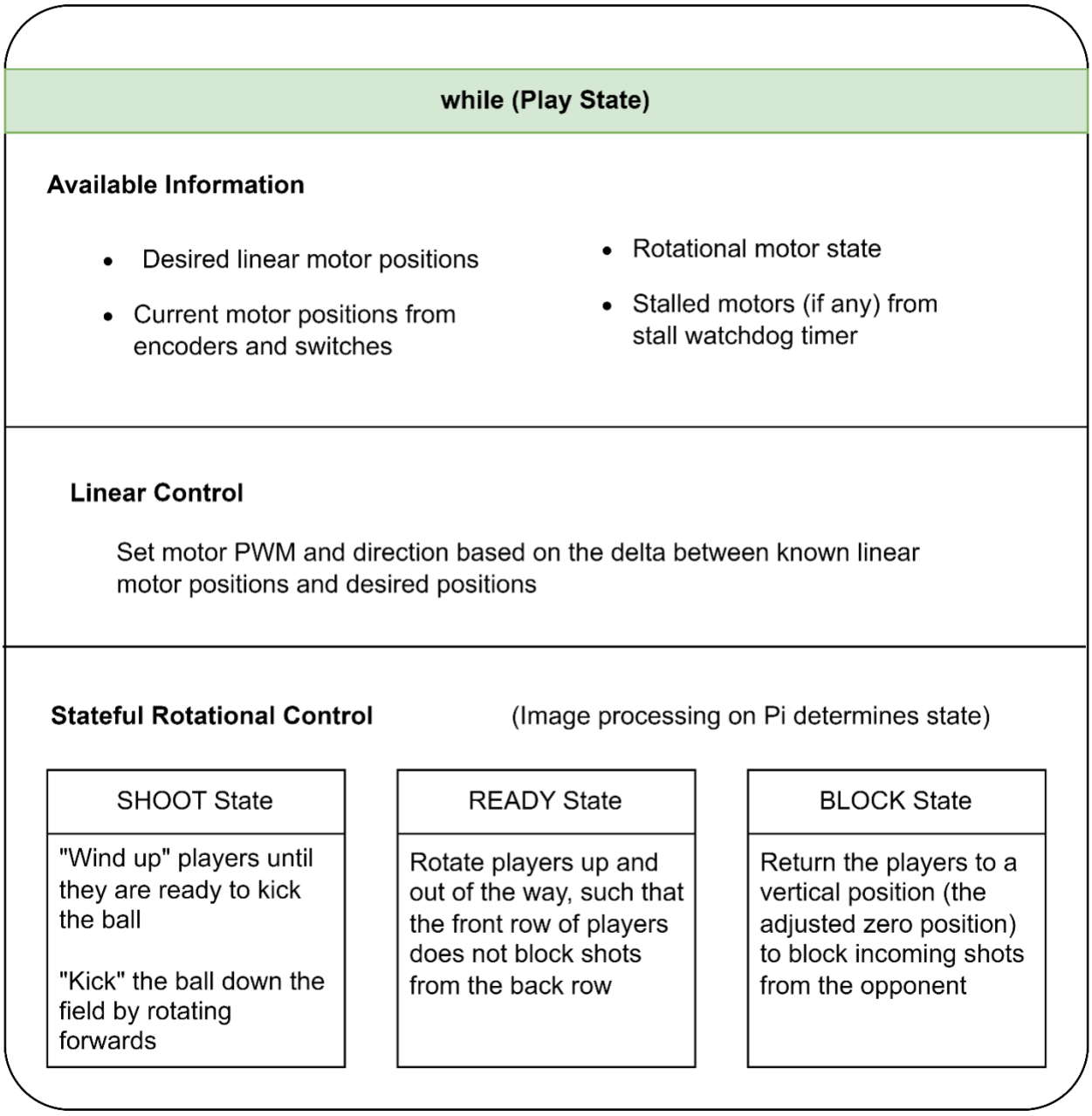


Figure 12: Embedded Code Play State Structure

While in play state, each iteration of the super-loop polls the limit switches and resets encoder counts if pressed. The linear motor control is called with the desired positions sent by the Pi – the linear motors move directly to those positions with no additional stateful logic. The control of the rotational motors is determined by a state sent by a Pi rather than a position. If sent the shoot state, the rotational motor will “wind up” until at a 45-degree angle. Once at a 45-degree angle, the motor will “kick”, turning forwards until reaching a certain threshold. If the motor is still in shoot state after kicking, it will return to “wind up” and try again. If sent the block state, the rotational motors will move to a vertical position in order to block the ball. If sent the ready state, the rotational, motors will move to a 90-degree position such that the ball can pass underneath

them when moving in the desired direction. A final spin state exists for the rotational motors that can only be entered with manual keyboard input to the Pi. In this state, the motors spin in reverse constantly.

At any time during the main super-loop, the stall watchdog timer can trigger a transition to the stall recovery state. In order to prevent damage to motors, a stalled motor is never moved in this state; rather, the other motor on the same assembly (offense or defense) is moved. Motors usually stall when caught on the ball – this routine attempts to dislodge the ball. For example, if the rotational defense motor is stalled, the linear defense motor will move a set distance. At this point, the main state will transition back to play. If the stall was not resolved, the watchdog timer will detect sequential stalls of one motor (or an edge case where multiple motors stall at the same time). This will transition to the main state to wait and requires manual resolution. In several hours of testing, the stall recovery never failed to resolve a stall before a double stall was detected.

Combined, these embedded structures move the motors to the desired game state sent by the Pi over the UART connection.

Hardware

The system uses two PCBs to drive motors and connect peripherals: one signal level header board, and one high current driver board. The header board sits on top of the MSP and contains only logic level signals. The header board is connected to the driver board using a ribbon cable. The driver board handles power distribution and includes the motor driver circuitry.

Header Board Design

All input and output signals for the embedded code travel through the header board. The signal to pin interconnect header is shown in Figure 13. Motor driver circuit control signals from the MSP are needed on the driver board, and motor encoder signals and power from the driver board are needed on the MSP. The ribbon cable between these two boards carries these signals.

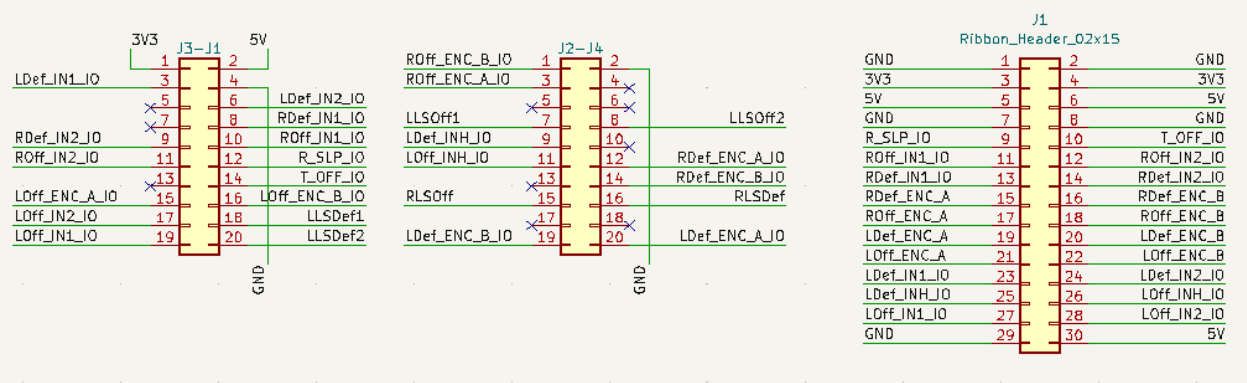


Figure 13: Header Board Interconnects to MSP (left) and Driver Board (right)

The header board has shrouded interconnects for four external physical limit switches. The circuitry for each switch includes a pull-up resistor, a current limiting resistor in series with the MSP pin, and a simple resistor capacitor (RC) low pass filter to protect from electrostatic discharge (ESD).

The header board has interconnects for two external slotted optical switches, which are a packaged combination of OPB830 and OPB840 photodiode and phototransistor devices [41]. The RC input protection and biasing resistors are shown in Figure 14. The 3.3kOhm resistor at the phototransistor emitter provides a 3.3V drop at the MSP pin given a fully on transistor state. The forward voltage across the photodiode is configured using a resistor divider to be approximately 1.7V, and the current through the diode is approximately 20mA. This results in the device behaving like a logic level switch.

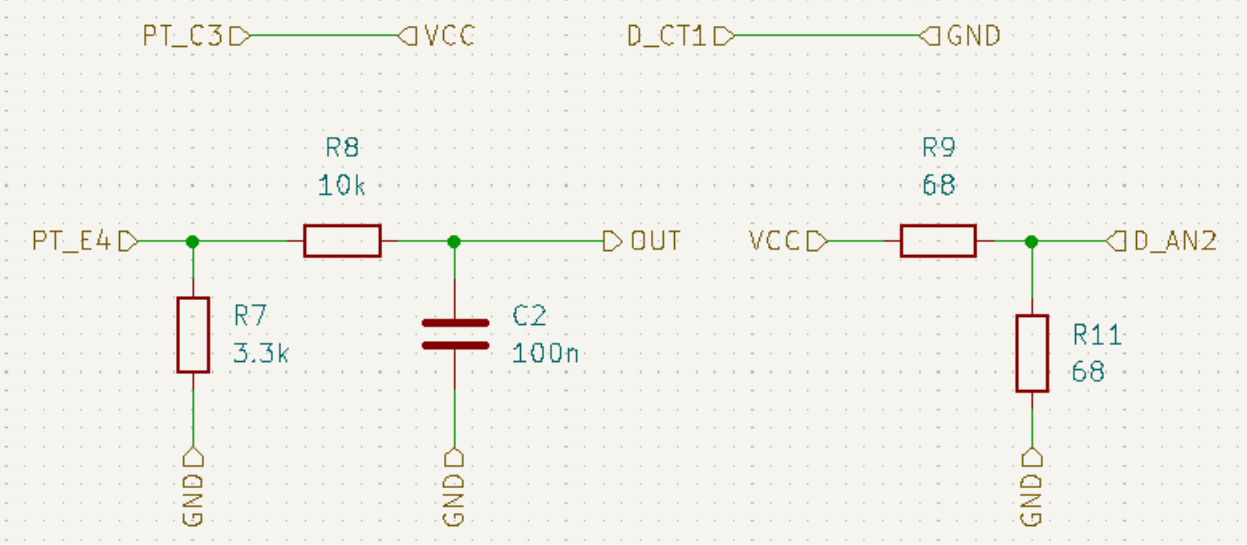


Figure 14: Photodiode Biasing and Input Protection

Any off-board signals also have simple ESD input protection before connection to the MSP pin. It was verified that the corner frequency of the RC low pass input protection circuits for the encoder signals had a sufficiently high cutoff frequency, such that the square wave encoder signals were not clipped.

Header Board Layout

Because everything on the header board is signal level, the routing of signals to the MSP header interconnects was determined by what would be most convenient for interrupt handling in the embedded software. Similarly, the ribbon layout was selected to optimize the driver board layout.

The ribbon cable header is located directly between the MSP headers to minimize the distance between the input protection circuitry and the MSP pins for off-board signals. As a result of this, the switch interconnects were placed around the remaining perimeter of the board to allow space for convenient wiring. In this configuration, the protection circuitry was grouped with the switch

interconnects rather than close to the MSP pins which would have been preferable. The final layout for the header board is shown in Figure 15.

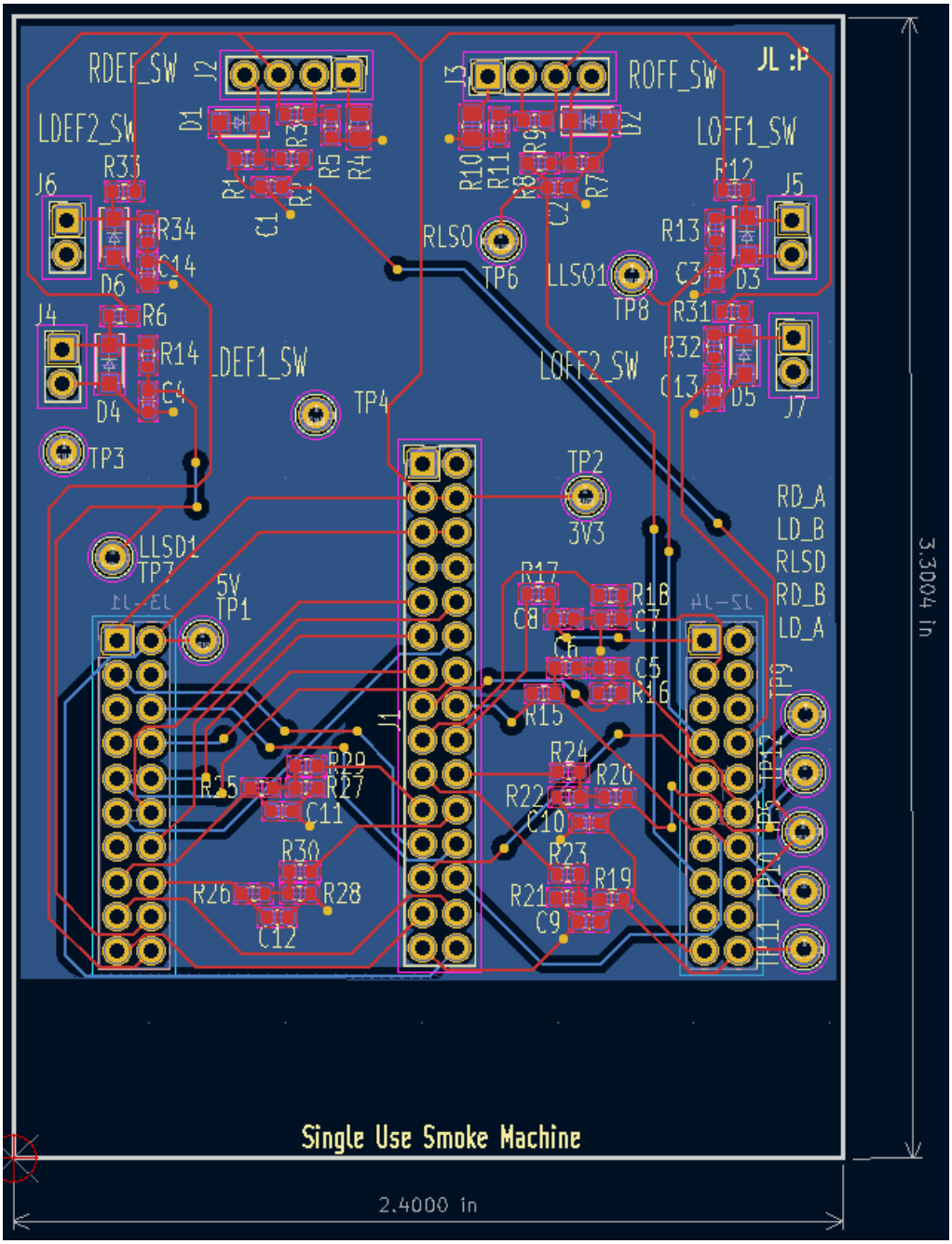


Figure 15: Header Board Layout

Driver Board Design

The driver board supplies power to the other boards and contains the motor driver circuitry. The board connects to an external DC 12V 16.5A switching power supply, which in turn connects to 120V AC wall power [42]. The hot AC power line of this supply was appropriately fused. A robust rocker switch with an internal LED allows power to the board to be turned on and off, and a surface mounted fuse holder and transient voltage suppression (TVS) diode protects against overcurrent or transient voltage spikes from the supply. The supply interconnect and circuitry is shown in Figure 16.

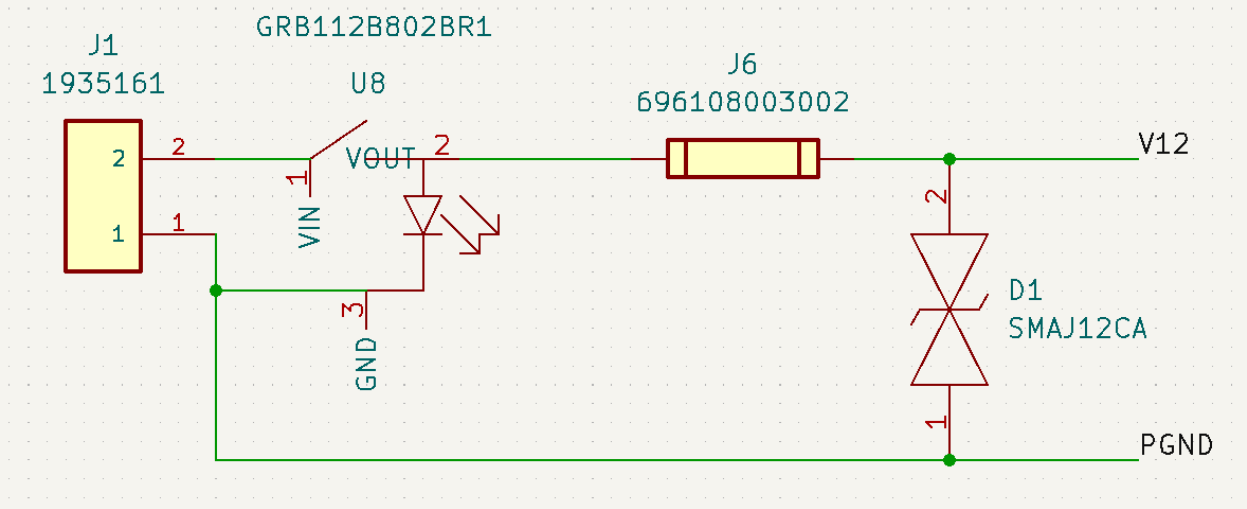


Figure 16: Driver Board Power Supply and Protection

The driver circuit for the linear motors uses four BTN8962TA half-bridge high-current driver chips, while the rotational motors driver design uses a single quad half-bridge DRV8935 chip [43], [44]. The datasheet recommended passive elements are used. For the rotational driver, a resistor divider at the VREF pins configures a trip current of 1.6 amps. This is slightly higher than the stall current of the motor and provides an additional overcurrent protection. Both the rotational and linear driver circuits take in four PWM input signals (two for each motor, a forward and a reverse) and produce four scaled power level 12V PWM output signals. Additionally, each driver circuit requires a logic high enable signal to enable the devices. Under normal operating conditions, each linear driver circuit can drive 5.5A of current and each rotational driver circuit can drive up to 1.5A of current. To account for dips in the voltage level of the power supply, bulk capacitance was added to each driver circuit. 100uF was selected for the rotational driver circuit, and 470uF was selected for the linear drivers—these values were upsized from the datasheet recommendations, because a relatively inexpensive power supply is used. The driver circuit layouts are shown in Figure 17 and Figure 18.

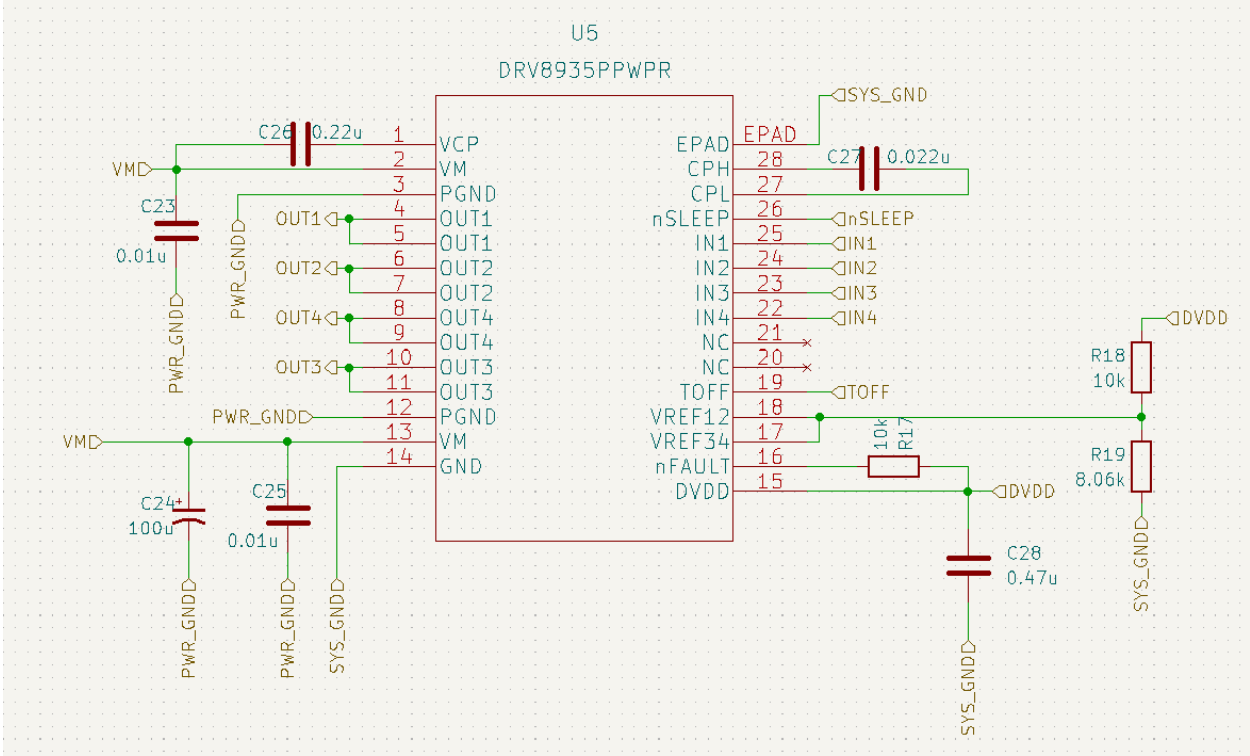


Figure 17: Rotational Motor Driver Circuit (both motors)

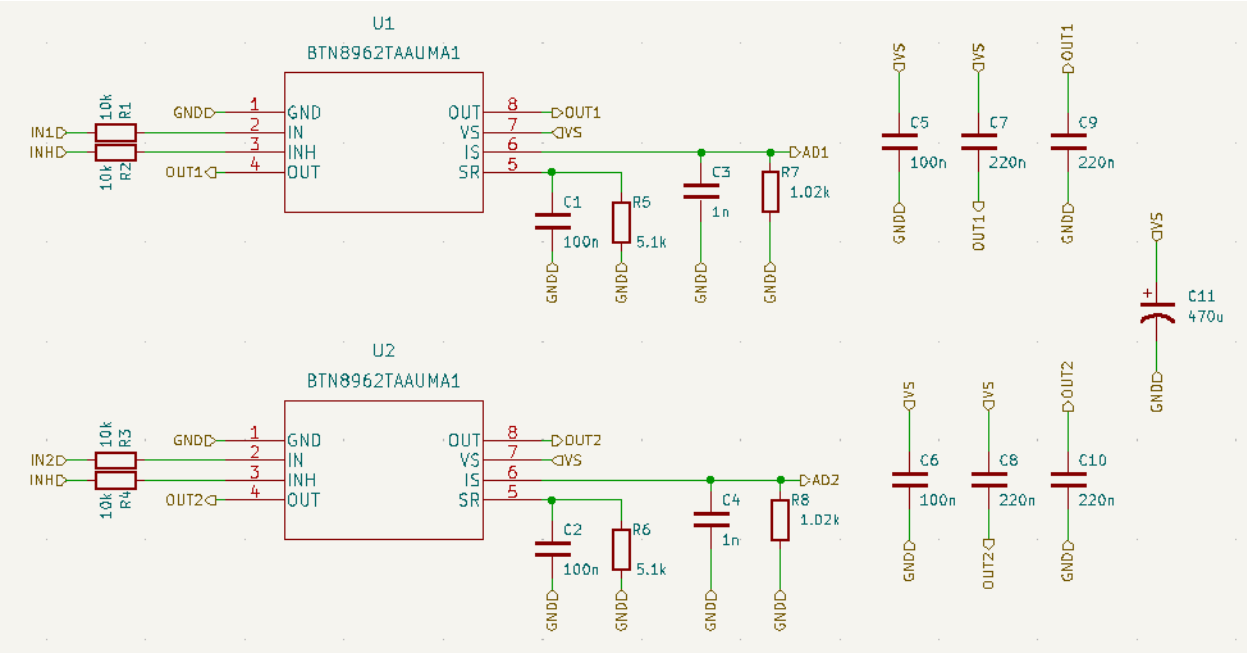


Figure 18: Linear Motor Driver Circuit (single motor)

The driver board also supplies 5V power to the linear motor encoders and 3.3V power to the MSP. A CRE1S1205SC DC-DC converter is used to produce 1W of power at 5V [44]. Because the device requires a minimum amount of power consumption, a small 249 Ohm resistor is used

to provide the required minimum of ~0.1W. The power ground is also shorted to logic ground at this point, such that the inductance of large high current traces has minimal impact on logic level signals. The 5V output is stepped down to 3.3V by the AP2125N-3.3TRG1[46]. This 3.3V output is run over the ribbon cable to power the MSP. Indicator lights show whether each voltage level is being successfully produced. These subcircuits are shown in Figure 19.

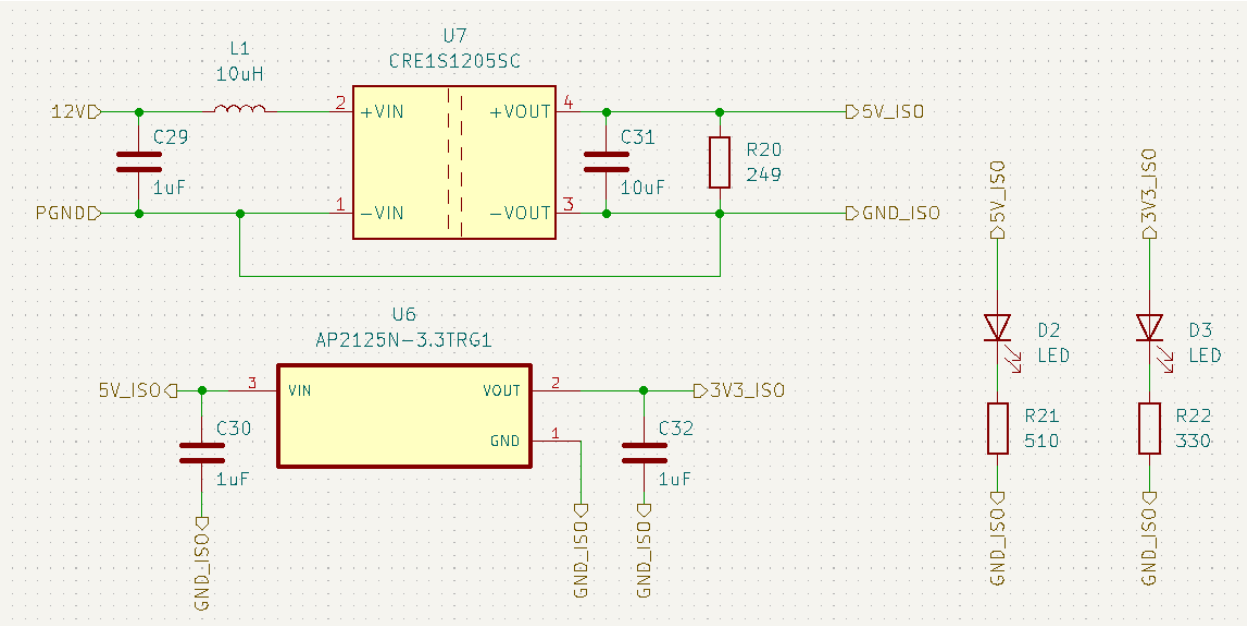


Figure 19: 12V to 5V DC Converter (top) and 5V to 3V (bottom)

Driver Board Layout

The primary consideration when laying out the driver board was 14.5A maximum current needed if all motors were running simultaneously. To support this amount of current, large power planes on the copper top layer connect 12V signals whenever possible. Notably, the large planes are used between the power supply and driver circuit 12V input as well as the driver circuit outputs and the motor interconnects. The driver IC datasheet recommendations are closely followed with regard to positioning of passive elements. The overall layout of the driver board is shown in Figure 20.

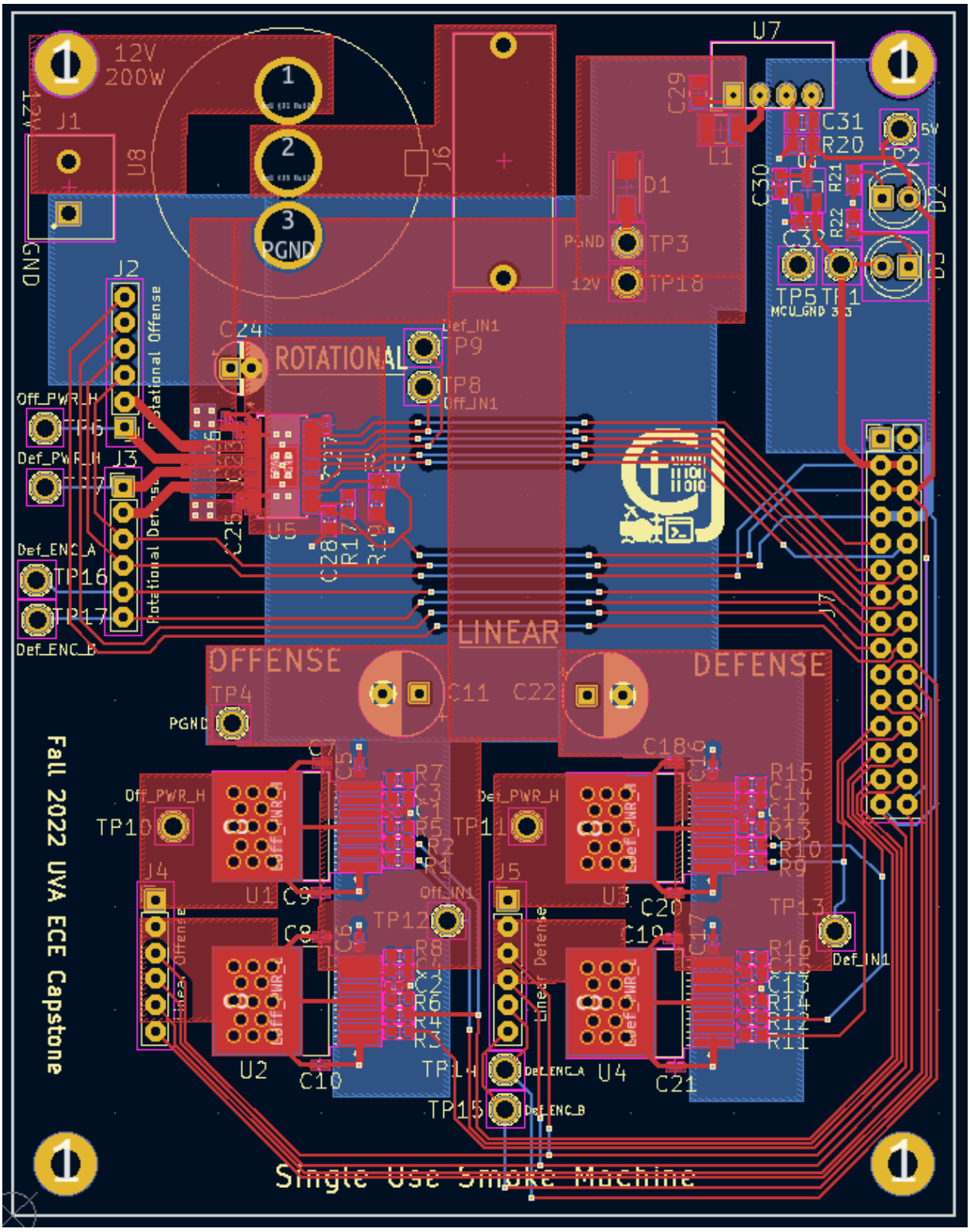


Figure 20: Driver Board Layout

Figure 21 shows a zoomed view of a single BTN chip. To aid in heat dissipation, an array of thermal vias connects the large output pad to an exposed plate (electrically unconnected) of copper on the back of the chip. It can also be seen that surface area of the V12 pin is maximized with the power plane.

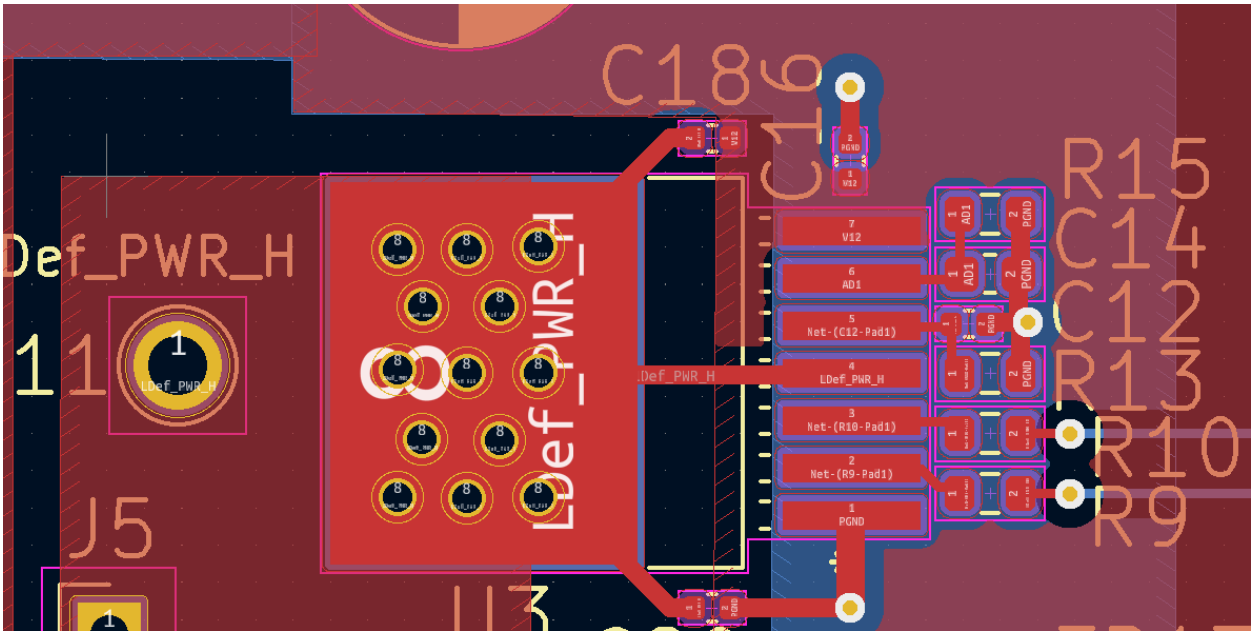


Figure 21: Zoomed View of High Current Area (Linear Driver H-Bridge)

Mechanical Assembly

The mechanical assembly can be broken up into three main components: linear actuation, rotational actuation, and the frame. A rendered 3D model of the full assembly can be seen in Figure 22.

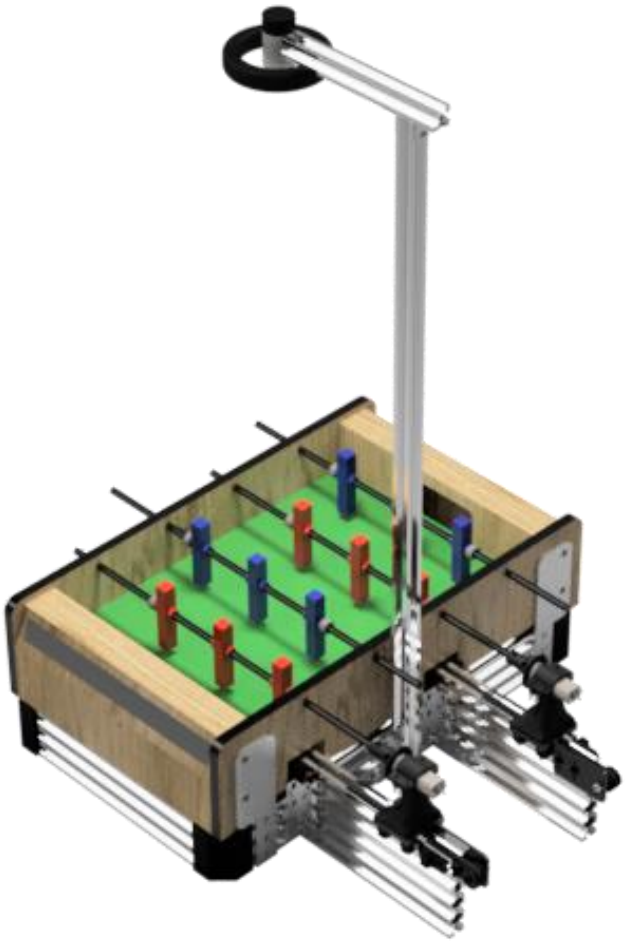


Figure 22: Full Assembly Model Render

Linear Actuation

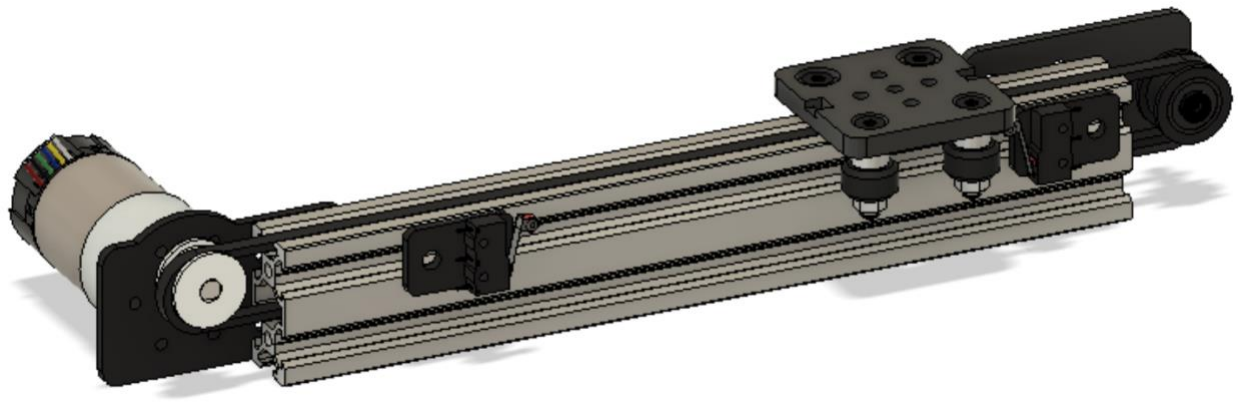


Figure 23: Linear Actuation Assembly Model

To allow for linear actuation a DC brushed motor with an encoder is mounted to 20x60mm aluminum extrusion with a custom 3D printed plate shown in Figure 24. This motor drives a belt and pulley system comprised of a pulley on the motor shaft, an idler pulley at the end of the rail, and a belt attached to an OpenBuilds Mini V Gantry Kit [47]. On the side of the rail, two limit switches [48] are attached with another 3D printed plate as shown in Figure 25. This plate extends past the edge of the limit switch to serve as a mechanical stop to prevent the gantry from slamming into the switch and exceeding the overtravel limit. These switches are used to set the minimum and maximum encoder counts to home and calibrate the motor and planning algorithm.

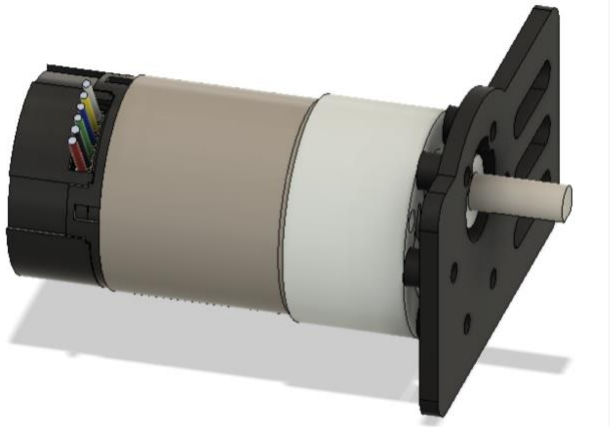


Figure 24: Linear Actuation Motor Mount

Figure 25: Limit Switch Mount

The specifications of this assembly were chosen to theoretically be able to react to a maximum ball speed of 1.3 m/s, which was measured from a normal game of foosball. Assuming that the system would need to respond to this speed of shot being shot from the opposing back row, travel its maximum linear distance of 78 mm from one edge of the field to the other, and react with a worst case command delay of 100 ms to take into account camera frame capture, image processing, UART, and microcontroller computation, the maximum linear speed was calculated using the script in Figure 26:

```
max_rod_travel = 78  # mm
end_player_distance = 274  # mm

ball_max_speed = 1300 #284/0.23  # mm/s
fps = 30

worst_case_command_delay = 0.1  # s
reaction_time = end_player_distance/ball_max_speed - worst_case_command_delay
avg_linear_speed = max_rod_travel/reaction_time  # mm/s
max_linear_speed = 2*avg_linear_speed # speed at peak of triangle
```

```
Reaction Time: 0.11076923076923076 s
Reaction Frames: 3.323076923076923
Average Linear Speed: 704.1666666666667 mm/s
Max Linear Speed: 1408.3333333333335 mm/s
```

Figure 26: Linear Speed Calculations

From these calculations, the system must be capable of achieving a maximum linear speed of approximately 1408.33 mm/s with a triangular speed profile. Using the distance per revolution for each pulley available from OpenBuilds [49] the rotations per minute (RPM) specification that the motor would need to achieve was calculated using the script in Figure 27.

```
pulley_options = [(3,20), (2,14), (2,30), (2,20)]  # pitch (mm), teeth

for pulley in pulley_options:
    belt_pitch = pulley[0]
    num_teeth = pulley[1]
    dpr_belt = num_teeth * belt_pitch  # mm/r
    desired_rpm_belt =  max_linear_speed/ dpr_belt * 60  # rpm
Pitch: 3 mm, Teeth: 20, RPM: 1408.3333333333335
Pitch: 2 mm, Teeth: 14, RPM: 3017.857142857143
Pitch: 2 mm, Teeth: 30, RPM: 1408.3333333333335
Pitch: 2 mm, Teeth: 20, RPM: 2112.5
```

Figure 27: Pulley RPM Calculations

Because this application requires high torque to accommodate rapid acceleration, the 3 mm pitch pulley with 20 teeth and corresponding 3mm pitch belt were selected as they require a lower RPM, reducing the required tradeoff between RPM and torque. The 2 mm pitch pulley with 30 teeth would require the same RPM, but a more aggressive pitch is more suitable for high torque applications, and the slight decrease in precision is negligible given that the radius of a foosball player's foot is approximately 6.5 mm.

To reach the maximum speed within the desired reaction time, the required torque was calculated using the method outlined in [50]. It is assumed that the rod starts at rest, the motor provides constant acceleration, the motion follows a symmetric triangular speed profile, the belt and pulleys have negligible mass and inertia (primarily because this data is not listed by the supplier and the parts had not been ordered at the time of these calculations), and the efficiency of the system is 75%. The script used to calculate the torque with the specific values can be seen in Figure 44 which utilizes the following equations to find the root mean square (RMS) torque:

$$T_c = \frac{F_a * r_1}{1000 * \eta}$$

T_c = torque required during constant velocity (Nm)
F_a = total axial force (N)
r_1 = radius of drive pulley (mm)
η = efficiency of belt drive system

$$F_a = m * g * \mu$$

m = mass of moved load (external load plus belt) (kg)
g = gravity (m/s^2)
μ = coefficient of friction of guide

$$T_a = T_c + T_{acc}$$

T_a = total torque required during acceleration (Nm)
T_{acc} = torque required due to acceleration (Nm)

$$T_{acc} = J_t * \alpha$$

J_t = total inertia of the system (kgm^2)
α = angular acceleration (rad/s^2)

$$J_t = J_m + J_c + J_{p1} + J_{p2} + J_l$$

J_m = inertia of motor (provided by manufacturer) (kgm^2)
J_c = inertia of coupling (provided by manufacturer) (kgm^2)
J_{p1} = inertia of drive pulley (provided by manufacturer, or calculate) (kgm^2)
J_{p2} = inertia of idler pulley (provide by manufacturer, or calculate) (kgm^2)
J_l = inertia of load (kgm^2)

$$\mathrm{J_l} = (\mathrm{m_l} + \mathrm{m_b}) * \mathrm{r_1^2} * 10^{-6}$$

m_l = mass of external load (kg)
m_b = mass of belt (kg)
r_1 = radius of drive pulley (mm)

$$\alpha = \frac{2\pi * N}{60 * t}$$

N = maximum angular velocity (rpm)
t = time for acceleration (s)

$$T_d = T_c - T_{acc}$$

T_d = torque required during deceleration (Nm)

$$T_{RMS} = \frac{\sqrt{T_a^2 * t_a + T_c^2 * t_c + T_d^2 * t_d}}{t_{total}}$$

T_{RMS} = root mean square (continuous) torque (Nm)
t_a = time for acceleration (s)
t_c = time for constant velocity (s)
t_d = time for deceleration (s)
t_{total} = total time for move (including any idle time between moves) (s)

Using these equations, the RMS torque was found to be 0.146 Nm or 1.493 kg*cm. Based on these calculated specifications of 1408.33 RPM and 1.493 kg*cm, the Pololu 6.3:1 Metal Gearmotor [47] was selected with a no-load performance of 1600 RPM and a stall extrapolation torque of 3.0 kg*cm.

Rotational Actuation

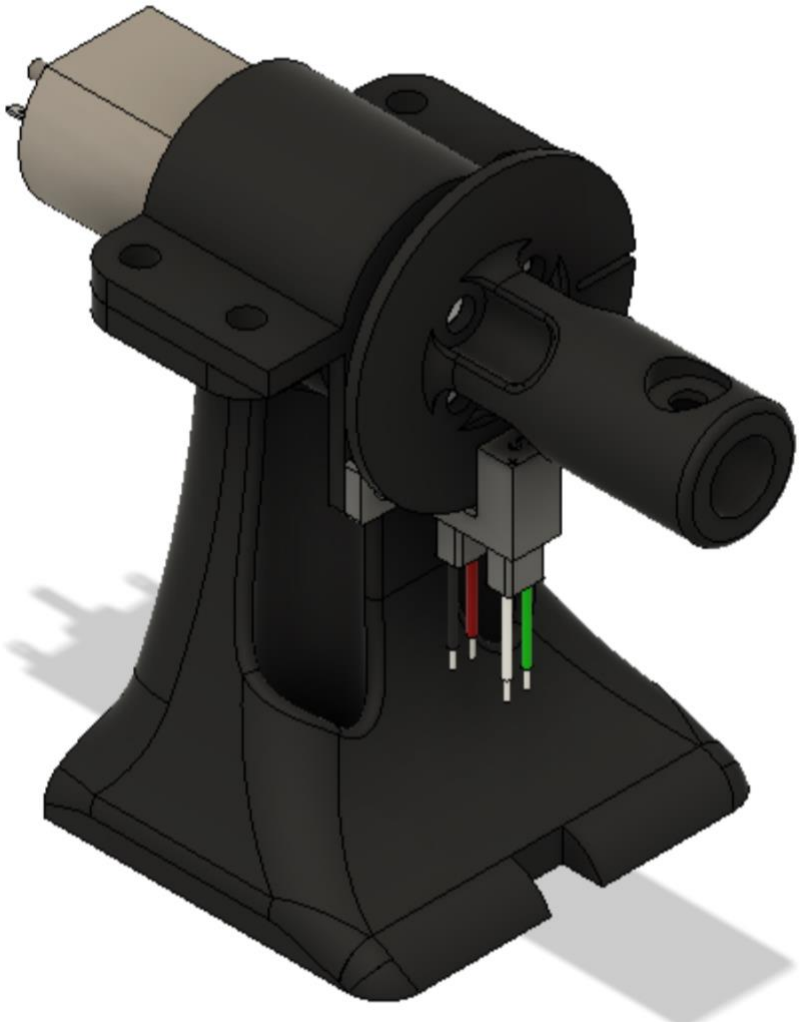


Figure 28: Rotational Actuation Assembly Model

Mounted to the gantry plate of the linear actuation assembly is a two-piece 3D printed holder that clamps to hold a small DC brushed motor with an encoder. The top piece also serves as a mount for an optical interrupter [41]. A 3D printed shaft coupling is attached to the motor shaft with a metal shaft collar. This attaches the motor's shaft to a foosball player rod like the one shown in Figure 29 using a screw that was previously used to attach a handle to the rod. It also serves as a homing mechanism by passing a disk with a notch in it through the optical interrupter. When the notch passes through the slot in the optical interrupter, light from the light emitting diode (LED) can pass into the phototransistor which allows current to flow, outputting a logic level signal to the microcontroller which is used to reset the encoder count for relative positioning.

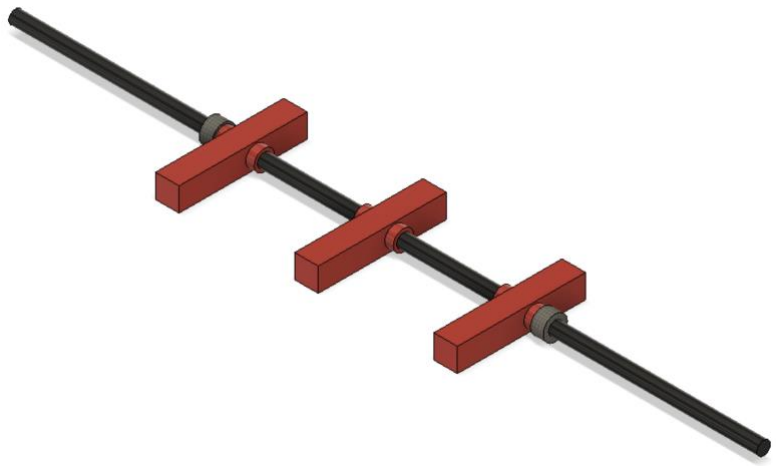


Figure 29: Foosball Player Rod

To select the DC brushed motor, the RPM and torque were calculated to achieve a ball speed of 1.3 m/s, the maximum speed achieved by a human player measured in the initial baseline games, from a 90-degree position, which is the maximum position allowed under the official rules of foosball. It was assumed that the rod starts at rest, that the motor provides constant angular acceleration, and that the player's foot contacts the ball instantaneously at the end of its swing. The players were treated as one rod rotating about its end for the purposes of moment of inertia calculations. The script used to calculate the torque with the specific values can be seen inFigure 44 Figure 44 which utilizes the following equations to find the RPM and torque:

$$v = w * r_{foosman}$$

v = linear speed
w = angular/rotational speed (rad/s)
$r_{foosman}$ = radius of foosball player measured from rod to bottom of foot

$$I_{rod} = \frac{1}{12} m_{rod} * l_{rod}^2$$

I_{rod} = moment of inertia of the rod (kgm^2)
m_{rod} = mass of the rod (kg)
l_{rod} = length of the rod (m)

$$I_{player} = \frac{1}{3} m_{player} * l_{player}^2$$

I_{player} = moment of inertia of the player (kgm^2)
m_{player} = mass of the player (kg)
l_{player} = length of the player (m)

$$I_{total} = I_{rod} + 3 * I_{player}$$

I_{total} = total moment of inertia (kgm^2)

$$\Delta_w = w_f - w_i$$

Δ_w = change in angular speed from start to end of kick (rad/s)
w_f = final angular speed (rad/s)

w_i = initial angular speed (rad/s)

$$w_{avg} = \frac{\Delta_w}{2}$$

w_{avg} = average angular speed (rad/s)

$$\Delta_t = \frac{\theta}{w_{avg}}$$

Δ_t = change in time from start to end of kick (s)
θ = starting kick angle (rad)

$$\alpha = \frac{\Delta_w}{\Delta_t}$$

α = angular acceleration (rad/s^2)

$$T = I_{total} * \alpha$$

T = required to produce the required angular acceleration

```
Rotational RPM: 292.09613085100796 rpm
Rotational Torque: 3.2485774288998206 kg*cm, Theta: 0.7853981633974483 rad
Rotational Torque: 1.6242887144499103 kg*cm, Theta: 1.5707963267948966 rad
```

Figure 30: Rotational RPM and Torque Calculations

As shown in Figure 30, the required speed is approximately 292.096 RPM and the torque required to achieve a 1.3 m/s ball speed from 45 degrees is 3.249 kg*cm and from 90 degrees is 1.624kg*cm. With these numbers, the Pololu 31:1 Metal Gearmotor 20D [52] was selected with a no-load performance of 450 RPM and a stall extrapolation of 2.4 kg*cm.

Frame

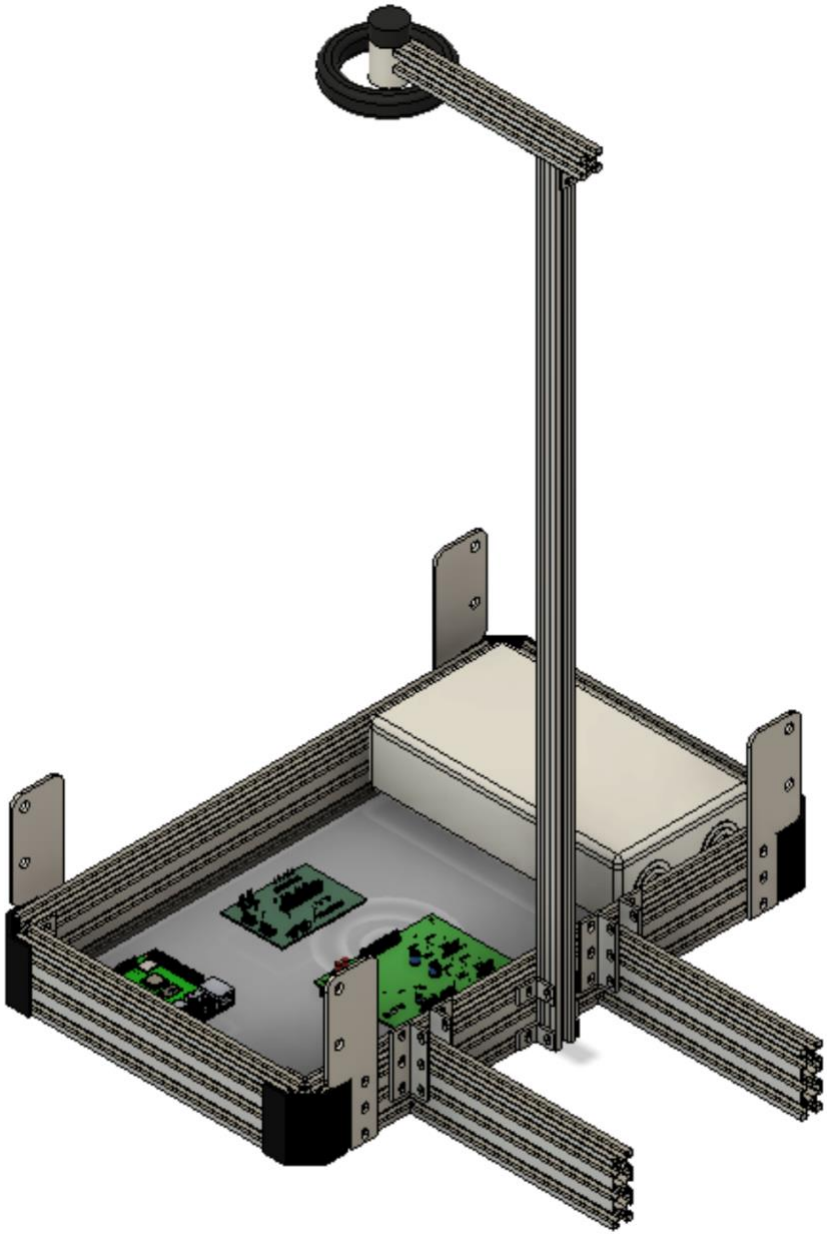


Figure 31: Frame Assembly Model

The frame of the device, shown in Figure 31: Frame Assembly Model was constructed out of 20x60mm aluminum extrusion with a 5mm slot width. The walls were connected with internal L-brackets and external corner brackets using M5 bolts and t-nuts. To mount the foosball table to the frame, custom steel brackets were cut on a waterjet and connected to the bolts on the side of the foosball table and to the frame with more M5 bolts and t-nuts. The camera was mounted to the frame with 20x20mm aluminum extrusion and L-brackets.

The Raspberry Pi [9] and circuit boards were mounted inside the frame with 10mm plastic hex standoffs and M3 bolts on a polycarbonate sheet slid into the slots of the aluminum extrusion and held in place with 3D printed brackets.

To comply with standards, the power supply needed to be enclosed in a NEMA 1 enclosure. One such enclosure was purchased from Amazon but was too large to fit inside the frame [53]. Subsequently it was cut and put back together using epoxy. Then it was mounted to the polycarbonate with hex standoffs similarly to the circuit boards.

Project Time Line

As evident in our original Gantt Chart in Figure 32, the core tasks of each subsystem were highly parallelizable. Individuals planned to work on lighting, ball sensing, path planning, detection, PCB, and assembly in parallel. A few tasks remained serializable, notably the parts of the mechanical assembly and full system assembly. Of course, within each category tasks became serializable. For example, ball detection required a working camera rig for proper development. See the Statement of Work section for a detailed description of which teammates were responsible for these tasks. Also depicted in this chart is how our timeline was informed by several key dates as discussed on the first day of class. These include the proposal deadline, the poster session, and of course the final demonstration.

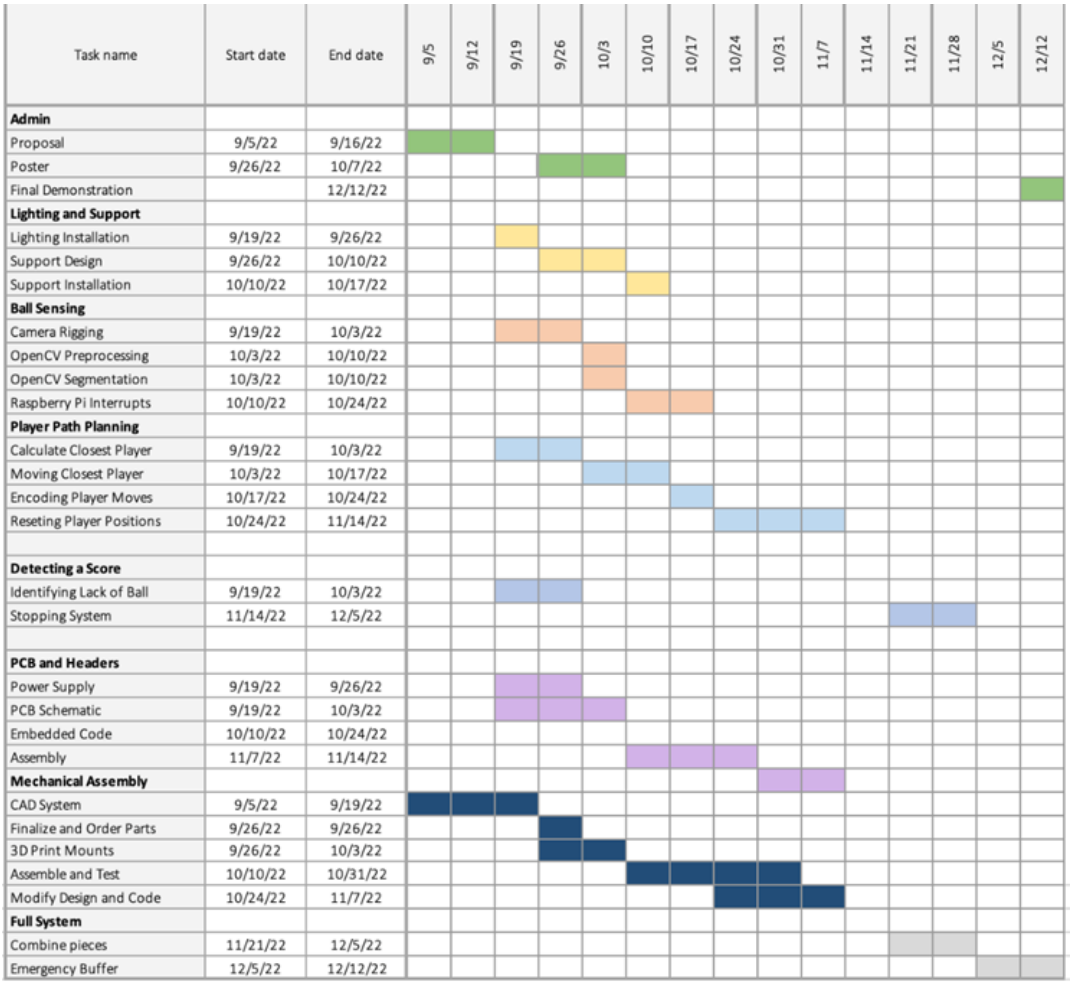


Figure 32: Original Gantt Chart

As the project developed throughout the semester, the timelines of various subtasks changed in response to shipping delays, bug fixes, difficulty underestimation, and other roadblocks. Our final Gantt chart, which reflects the actual timeline of the project, can be found in Figure 33. In comparing these two figures, one might notice a few substantial differences. These include dedicating much more time to perfecting ball detection and assembling the full project later than expected. Neither of these changes to our schedule presented major problems; our original timeline had enough built-in flexibility that these adjustments still kept us on schedule for the final demonstration.

Task name	Start date	End date	9/5	9/12	9/19	9/26	10/3	10/10	10/17	10/24	10/31	11/7	11/14	11/21	11/28	12/5	12/12
Admin																	
Proposal	9/5/2022	9/16/2022															
Poster	9/26/2022	10/7/2022															
Final Demonstration		12/12/2022															
Lighting and Support																	
Lighting Installation	9/19/2022	10/31/2022															
Support Design	9/26/2022	11/14/2022															
Support Installation	10/10/2022	11/21/2022															
Ball Sensing																	
Camera Rigging	9/19/2022	10/3/2022															
OpenCV Preprocessing	10/3/2022	10/24/2022															
OpenCV Segmentation	10/3/2022	10/24/2022															
v4l2 Preprocessing	10/10/2022	10/24/2022															
v4l2 Segmentation	10/10/2022	10/24/2022															
Raspberry Pi Interrupts	10/31/2022	11/14/2022															
Player Path Planning																	
Calculate Closest Player	9/19/2022	10/3/2022															
Moving Closest Player	10/3/2022	10/17/2022															
Encoding Player Moves	10/17/2022	10/24/2022															
Reseting Player Positions	10/24/2022	11/14/2022															
Detecting a Score																	
Identifying Lack of Ball	9/19/2022	10/3/2022															
Stopping System	11/14/2022	12/5/2022															
PCB and Headers																	
Power Supply	9/19/2022	9/26/2022															
PCB Schematic	9/19/2022	10/3/2022															
Embedded Code	10/3/2022	10/24/2022															
Assembly	11/7/2022	11/14/2022															
Mechanical Assembly																	
CAD System	9/5/2022	9/19/2022															
Finalize and Order Parts	9/26/2022	9/26/2022															
3D Print Components	10/3/2022	11/7/2022															
Assemble and Test	10/10/2022	10/31/2022															
Modify Design and Code	11/14/2022	11/28/2022															
Full System																	
Combine pieces	11/21/2022	12/5/2022															
Emergency Buffer	12/5/2022	12/12/2022															

Figure 33: Final Gantt Chart

Test Plan

The test plan for this project was divided into three groups along subsystem lines to reduce debugging time needed for the combined system. These three groups were hardware and firmware, UART communication, and image processing. Figure 34 shows the hardware and firmware test plan.

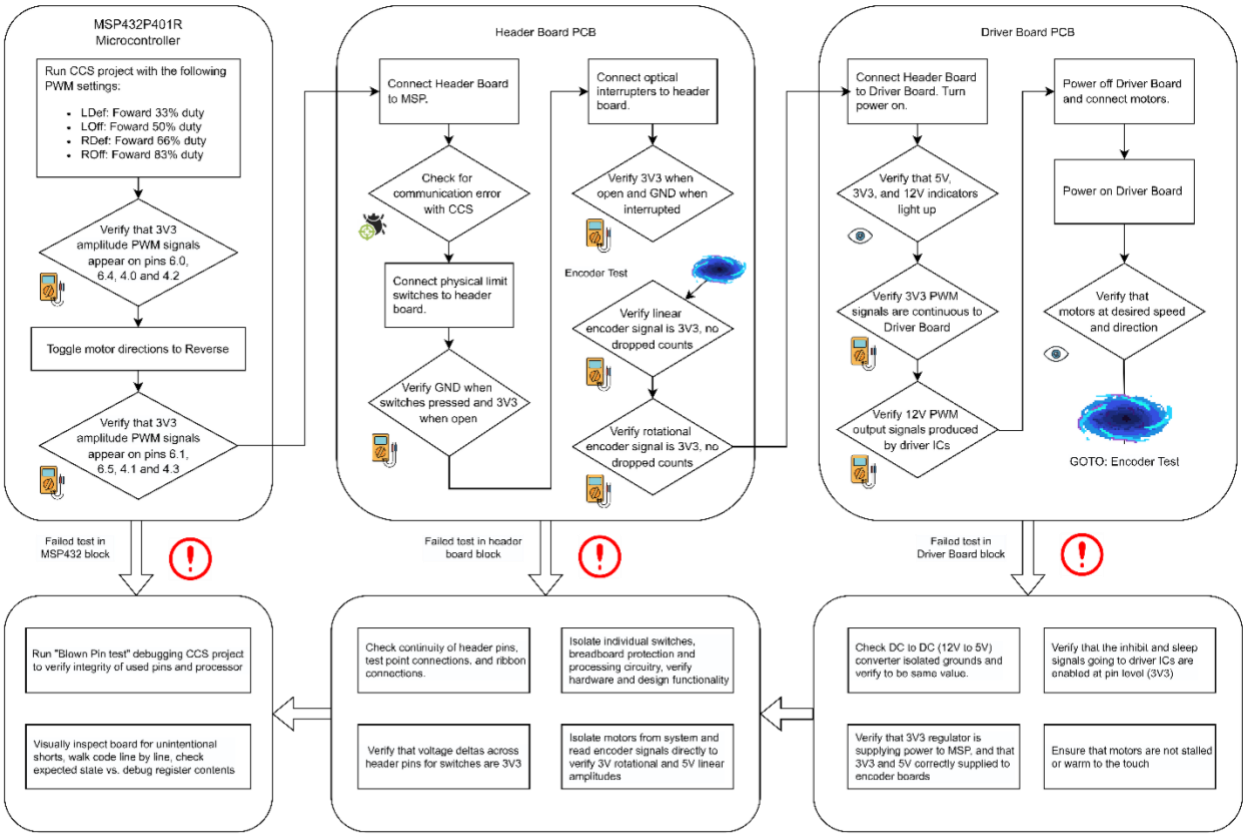


Figure 34: Hardware and Embedded Test Plan

For hardware and firmware verification, tests were structured such that all dependent signals were verified before continuing to the next test. Using this structure, a failed test indicated that the current component under test was responsible for failure. First, the logic level outputs of the MSP432 were tested with the MSP isolated. It was verified that a pulse width modulation (PWM) signal could be produced at each of the pins connected to the motor driver circuitry. It was verified using a multimeter that the software defined duty cycle and logic-level amplitudes of each PWM signal matched the expected values. The header board was then added to the MSP with the CCS debugger open to verify that communication between the laptop running CCS and the MSP was not lost. Each of the six switches (four physical limit switches and two optical interrupters) were then connected to the header board and tested to confirm that the switches were active low and logic level otherwise. With the functionality of the header board and the MSP confirmed, the driver board and 12V power supply were connected to the MSP and header board. The power indicator light emitting diodes (LEDs) were checked for each voltage level, and it was verified that the MSP could run using power from the driver board. The outputs of the motor driver ICs were tested to verify that they had the expected 12V amplitude and matched the duty cycle of their logic level inputs. Motors were connected, and it was confirmed that their speed, direction, and encoder signal outputs matched the design. Passing these tests in sequence indicated that the designed printed circuit boards (PCBs) and low-level embedded abstractions were working as intended.

During this process, several hardware design flaws were revealed. While testing the limit switch outputs, the inactive voltage at the switch output was found to be lower than logic level. It was identified that this problem resulted from the Zener diodes used in the input protection circuit conducting more current than anticipated. The diodes were removed. During the switch tests, it was also revealed that the initial design for the optical interrupter biasing network did not allow sufficient current through the internal photodiode. This was corrected by replacing the biasing network with smaller resistor values and shorting one of the surface mount pads with a 0 Ohm surface mount device (SMD). While testing the supply voltages on the driver board, it was found that the isolated grounds on the 12V and 5V sides of the DC-to-DC converter circuit had different values. A wire was soldered between the two grounds, minimizing contact forcing the grounds to have equivalent values. Finally, while testing the linear motor encoder signals, the processed encoder signal was found to be below logic level. The values in the resistor divider were originally too low for the encoder to drive sufficient current. These resistors were swapped for higher values.

The test plan for the UART communication between the MSP and Raspberry Pi is shown in Figure 35.

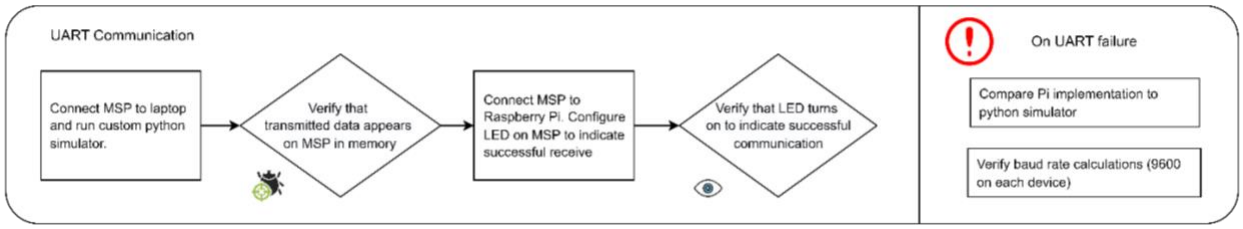


Figure 35: UART Communication Test Plan

Generally, the communication test plan uses the simple and easily verifiable behavior of the python simulator to test the transmit and receive UART configuration on the MSP before combining with the more complex Raspberry Pi implementation. The python simulator was run on a laptop connected to the MSP, using the CCS debugger to verify the contents of the data being sent and received. The MSP was then configured to toggle an LED when receiving valid data and tested with the Raspberry Pi. It was confirmed that the transmission from the Pi toggled the LED as expected. This testing process was repeated several times when initially configuring the UART communication protocol. Some minor software adjustments were required to allow the communication (COM) port to open before transmitting.

The test plan for the image processing code is shown in Figure 36. Because the image processing operates at a higher level of abstraction than the rest of the system, testing consisted mostly of viewing debug output and iterating.

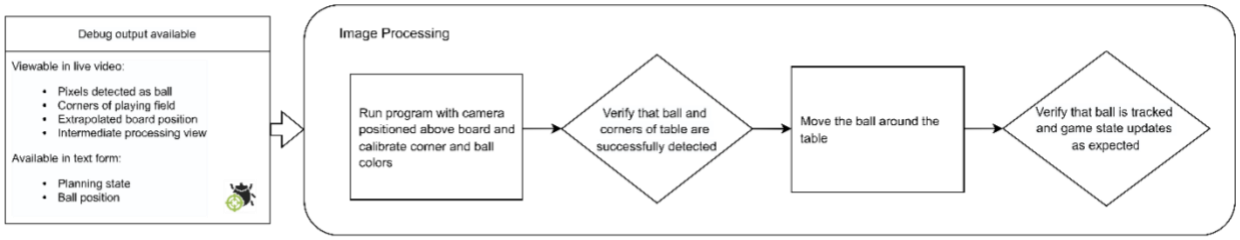


Figure 36: Image Processing Test Plan

To test an iteration of the image processing program, the program was run with the camera positioned above the board. The program was then calibrated to identify the corners of the board and the color of the ball. Debug output was used to verify that the ball position and center was being correctly identified, the size of the table was being correctly extrapolated, and the corners of the table were found correctly. The ball was moved to various positions on the table to observe the state output of the planning algorithm and the tracked position of the ball. Executing each of these individual test plans independently allowed for relatively high confidence moving into the full system level tests. Once combined, the remaining testing mostly pertained to gameplay algorithm tuning – this tuning would have been difficult to reduce into a consistent test plan. Overall, the subsystem level testing was effective at finding and eliminating bugs in the system.

Final Results

Our project resulted in a functioning, playable robotic foosball table that entertained dozens of visitors at the Capstone Fair. A picture of the entire system is provided in Figure 37.

Figure 37: Final Assembly

We also provide a picture of the table's insides, which contains the power supply, PCBs, Raspberry Pi, and MSP432, in Figure 38.

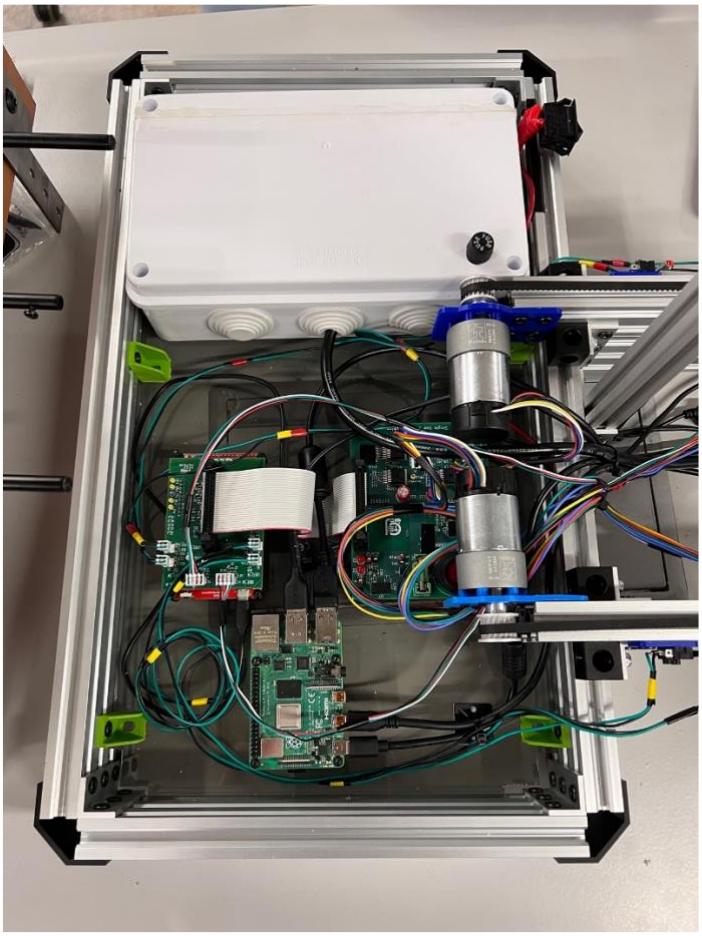


Figure 38: Final Internal Circuitry

In our proposal, we outlined eight goal criteria for our robotic foosball table. For each of these, we honestly assess whether these were met and provide specific examples of performance.

1. *Raspberry Pi collects video frames and processes them at 30Hz.*

   Our table uses a camera that collects video frames at 30Hz. The Raspberry Pi reads these frames, processing each one in series. At 30Hz, each frame needs to be processed in 33ms or less to prevent lost data. The processing time encompasses everything from the moment after a frame buffer is dequeued to the moment before that buffer is re-enqueued, including the ball detection, player planning, and transmission to the MSP432. We modified our Raspberry Pi code to measure and output this elapsed time on every frame, then played several minutes of foosball against the table. We gathered processing time data for 15,694 frames, a histogram of which is shown in Figure 39. The mean processing time was 2.9 ms, 99.7% of frames took 6 ms or less, and the very longest any frame took to process was 8.5 ms. Since every frame in a large sample size comfortably beat the 33

ms deadline, we consider the deterministic 30 Hz image processing requirement to be soundly met.

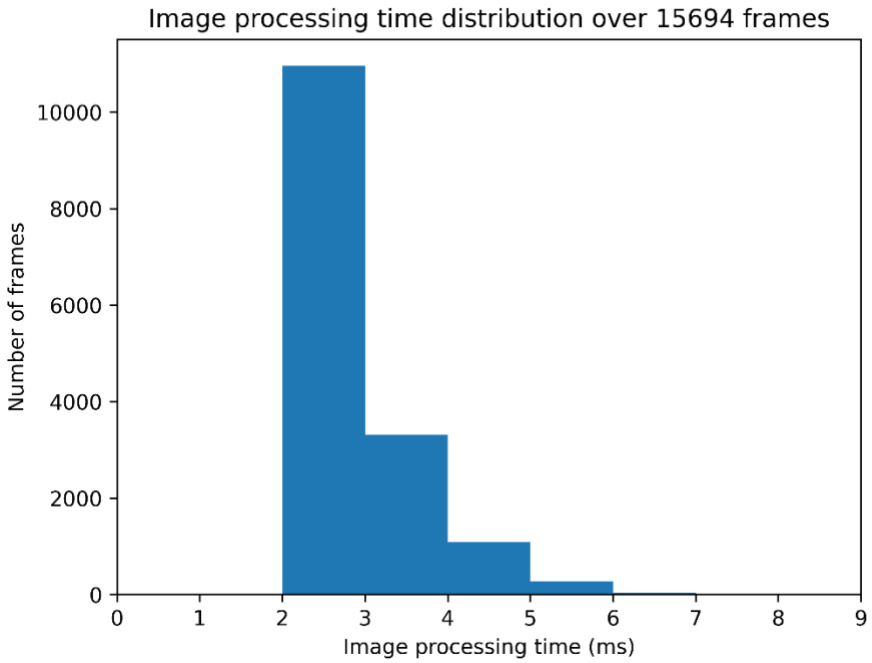


Figure 39: Image Processing Time Distribution

2. *System stops playing after a goal or when the ball is otherwise not on the field.*

   Whenever the camera cannot detect the ball, the system automatically positions the players to a default ready state. In this state, the players are centered on the goal line and in the vertical position. Scoring the ball or removing it from the table in any way triggers this behavior. If the ball becomes visible again, the system once again begins playing without human intervention. In all our play testing, this specification is consistently met.

3. *Players return to a lateral default blocking state after moving to block the ball.*

   When a player blocks the ball, two behaviors can happen. First, if the ball is not obstructed (for example, by a horizontal player), then the system continues predicting its trajectory and position its players accordingly. Alternatively, if the ball is obstructed, the system returns to a default state as described in the second specification. In our proposal, we anticipated having enough time between processed frames to return the blockers to a home position each time. However, because we were able to process information so quickly, it is more efficient to have the players perpetually ready to block a perceived ball. This means that, while we do not meet the original specification outlined in our proposal, it is only because our actual implementation uses a more effective strategy.

4. *Players return to a vertical default blocking state after hitting the ball.*

After completing the shooting routine consisting of a backwards windup and a forward kicking swing, if the image processing sees that the ball is still within range to kick (i.e., the robot missed or hit the ball weakly), the robot continues repeating the shooting routine to attempt to kick the ball again. If the robot successfully hits the ball, the image processing will detect that the ball is ahead of the players, so they will return to the default vertical blocking position. In our playtesting, the players always returned to this vertical default position after successfully hitting the ball downfield, so this requirement was met.

5. *System is capable of deliberately hitting the ball towards the goal to score.*

   When the ball is close enough to the players to kick it, the robot performs a back-and-forth motion with the rotational motors in order to strike the ball downfield towards the opposing goal. Although the simple back-and-forth logic means that the robot is also capable of hitting the ball towards its own goal, for most shots this does not happen because the ball started in front of the players (i.e., it just blocked a shot by the opposing team) or the ball came from far enough behind the players that they transitioned straight from the ready position to the shooting routine, so no initial windup was necessary. In our playtesting, the robot consistently attempted to shoot the ball when it was in range, it usually kicked the ball in the correct direction, and many of its shots successfully went into the opponent's goal, so this requirement was successfully met.

6. *System does not block itself when its back row of players is hitting.*

   Whenever the ball is too far behind a row of players to hit, that row moves the players to a horizontal position (parallel to the table) so that the ball can pass under them. Therefore, if the back row of players kicks the ball, the front row will already be in position to allow the shot past. In our playtesting, we confirmed that the front row always allowed a shot by the back row past, so this condition was met.

7. *System hits the ball hard enough that it is reasonably challenging for a human to block it.*

   To assess how difficult it is for a human to block the ball, we perform an analysis on 120 shots from gameplay with three different group members. In total, this data is obtained from a collection of approximately 5,400 frames in aggregate, each of which was manually tabulated for data collection. Of these 120 shots, 70 were the robot shooting against the human. Figure 40 shows the histogram of these shots that were successfully blocked by a human. In total, 31 of the robot's shots were blocked.

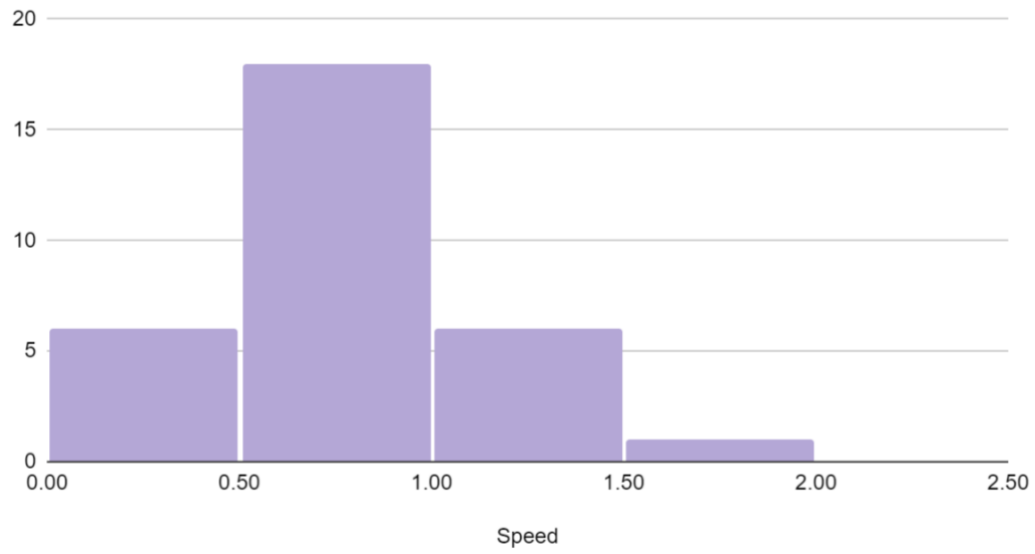


Figure 40: Histogram of Robot Shots Blocked by Human

We compare this with Figure 41, which shows the histogram of shots that were *not* blocked by a human. Note that 39 shots were not blocked, meaning the robot performed a shot that the human could not block 56% of the time. Furthermore, the robot shot the ball at a max speed of 2.1 m/s, which is significantly faster than the top human speed of 1.3 m/s we determined in our proposal. As a result, we conclude that the robot is capable of hitting the ball hard enough that it is reasonably challenging for a human to block it.

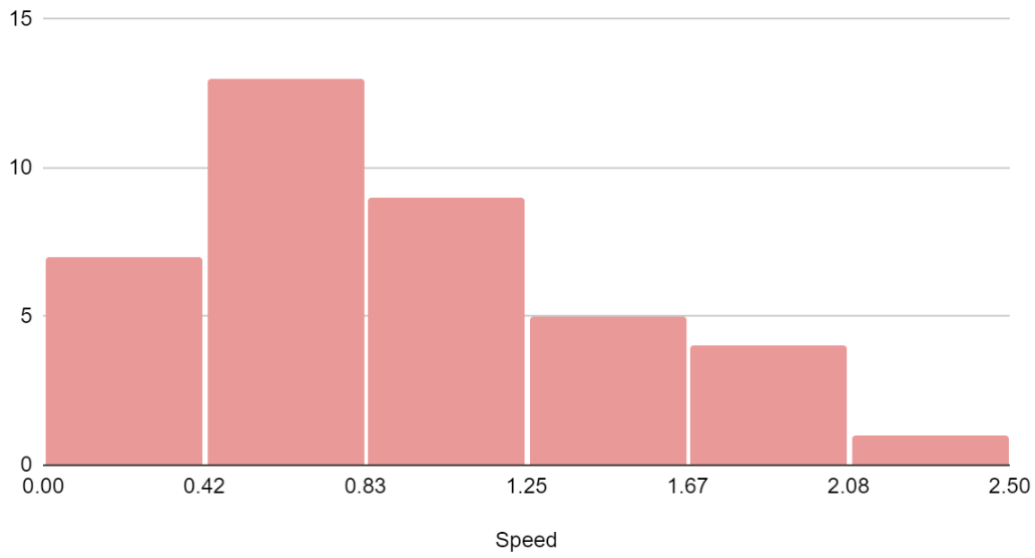


Figure 41: Histogram of Robot Shots Not Blocked by Human

8. *System blocks a ball that is hit with reasonable power from a player of medium skill level.*

   From the onset of our project, the cornerstone of our strategy has been defense. We reasoned that a sophisticated defense would give us the time needed to score goals, and that if the human could never score on the system then eventually they would make a mistake and the system would score on them. Furthermore, we understood that predicting the ball's trajectory and blocking it was an easier task than aiming it, leading us to rely on defense instead of offense.

   Because of this emphasis on defense, the system's blocking mechanism performs well against human opponents. In our proposal we mention "reasonable power" from players of "medium skill." While these terms are poorly defined, our working project can block most shots regardless of power. In our extensive testing early on, we determined that the maximum speed a human could hit the ball on our table was 1.3m/s. Given the time it takes to process frames, our system is easily capable of reacting to speeds much higher than that.

   The data collected from the study mentioned in specification seven, above, support this claim. Figure 42 shows the histogram of shots from the human that the robot does not block.

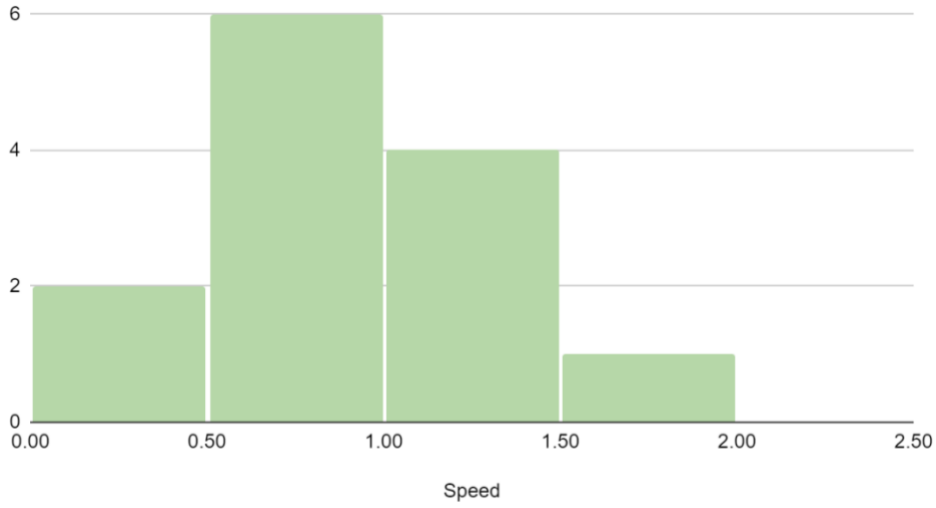


Figure 42: Histogram of Human Shots Not Blocked by Robot

We compare this data with that from Figure 43, which shows the histogram of human shots the robot did block. Notably, the robot successfully blocks 70% of human shots. We contrast this with the data from the human blocking the robot's shots, where the human blocks only 44% of the robot's shots.

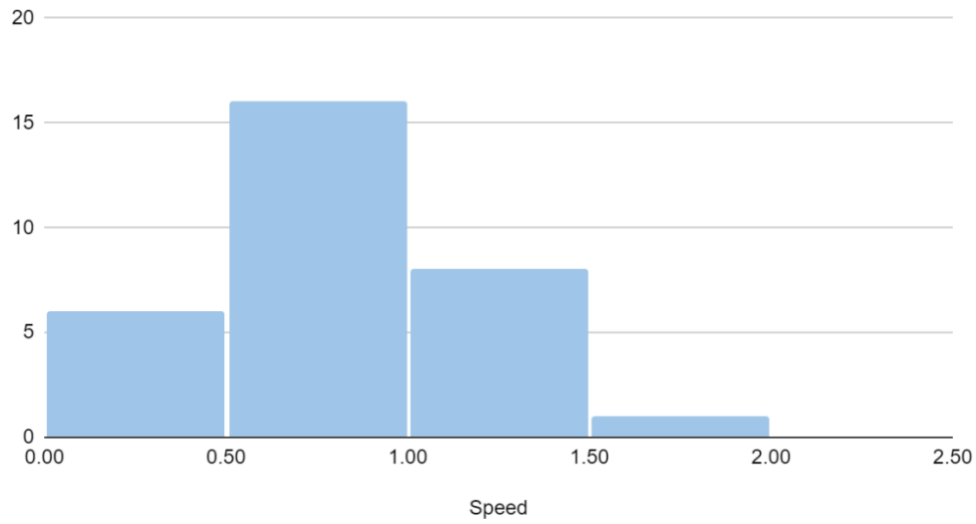


Figure 43: Histogram of Human Shots Blocked by Robot

Additionally, the robot's shots have an average speed of 0.89 m/s, whereas the humans' shots have an average speed of 0.76 m/s. This means on average the robot shoots 17% faster than the human.

There does not appear to be a relationship between the situations when the robot does not block a shot; no common speed, angle, wait time, or other factors we could think of. Despite our best efforts, there are still slight inaccuracies in how the system estimates the future position of the ball and moves to intercept it. These likely arise from the table shaking during play, distortion from the camera lens, and noisy image processing. However, we believe that the data still show that the robot is capable of blocking a shot hit with reasonable power from a player of medium skill.

To conclude, the system successfully meets (and in some cases, far exceeds) the specifications laid out in our original proposal. Perhaps the best metric of success, however, is the robot's performance at the Capstone Fair. At the Fair, we recorded over thirty matches with a wide variety of human opponents. The scores are recorded in Table 12; the robot beat the humans 208-94 and only lost to 3 people. For these reasons, we propose that this project succeeded.

Costs

The cost overviews can be seen in Table 2 and Table 3 in the Cost Constraints section. $431.38 of the $500 class budget was used and $524.54 was spent external to the class budget. Table 9 in the Appendix shows the detailed list of all components purchased through the class budget. Table 10 shows the estimated value of all parts external to the class budget used for the project, which total to $732.46. This includes the estimated value of everything that was already owned and items that were purchased specifically for the project. Thus, the total estimated value of everything that was used in the project is $1,163.84. This is broken down by category in Table 7.

Table 7: Total Estimated Value by Category

Mechanical	$444.73
PCBs	$266.13
Motors	$224.70
Processing & Control	$128.94
Other Electrical	$57.35
Foosball Table	$41.99
Total	$1,163.84

The amounts spent specifically for the project, which excludes anything that was already owned, can be seen in Table 8.

Table 8: All Costs Incurred for Project

Mechanical	$365.75
PCBs	$266.13
Motors	$224.70
Processing & Control	$0.00
Other Electrical	$57.35
Foosball Table	$41.99
Total	$955.92

If this were to be manufactured as a product, there would be many ways to save cost in mass production. The cost to manufacture 10,000 units of the current design is estimated in Table 11 at $812.50 per unit. The main savings included here are from the decreased component and assembly costs at scale. This price could be decreased further by ordering similar mechanical parts that are less niche and not from specialized suppliers. Instead of using an MSP432 [8] development board, the processer Integrated Circuit (IC) alone could be purchased and integrated onto one of the boards. The 3D printed parts could be recreated with plastic molds which would be cheaper and much faster. All of these would contribute to making production cheaper per unit and faster.

Future Work

Though the table is complete and generally meets all the specifications we originally outlined, there are several ways we would improve the project if we had more time. Perhaps most obvious are additional safety features. Given more time, we would build a transparent enclosure around the motors to prevent anyone from touching them. We would also add permanent markings warning of potential danger, for example a sign that indicates people should not reach inside the table. A better on switch that was attached to the table would also be safer (and more presentable).

When it comes to the mechanical assembly, many of our parts were 3D printed. This means they are different colors (reducing presentability) and weaker than manufactured metal parts. While 3D printing was a viable strategy for a semester-long project, and we thoroughly vetted the designs and parts for potential failures, professionally designed parts would still improve the durability and professionalism of the project.

If we had more time, the most effective way we could improve the project would be with better gameplay algorithms. Though the system plays decently well, the underlying gameplay logic is rudimentary and has a few glaring flaws. First, the robot behaves exactly the same when trying to kick a ball that is behind its players as it does for a ball in front of them. This causes it to have a chance of accidentally kicking the ball backwards while winding up to kick a ball behind it, so it occasionally scores on itself. While the image recognition and planning algorithms are capable of identifying a ball behind the robot's players, we would still need to implement embedded motor control logic to avoid hitting the ball backwards when that is the case. Second, while the robot intentionally kicks the ball downfield, it does not currently adjust its angle to aim for the goal laterally, so many shots miss the goal and bounce off the far wall. Further work could improve on this by deliberately striking the ball off-center to angle the shot towards the goal, requiring less attempts to score.

Other potential improvements could include the use of more advanced algorithms for trajectory planning or predicting opponent actions, potentially even incorporating machine learning. We could also upgrade the camera to a higher frame rate to be able to track the ball's position more precisely and quickly. To implement a more sophisticated aiming capability, the system could be upgraded to track the locations of the human-controlled players, either with a camera or some form of encoder, so that the system could aim around them.

Throughout the course of this project, we have also encountered several roadblocks that might have been avoided. For one, we quickly realized that supplying power to a stalled motor will quickly ruin it. We were also reminded of the importance of precise measurements, especially when some parts had to be printed multiple times to correct for sub-millimeter differences. At the beginning, we underestimated how complicated even simple gameplay would become, and we also allotted less time than necessary for assembly, testing, and generally fixing last-minute problems we did not anticipate.

References

[1] R. P. Ltd, "Raspberry Pi," *Raspberry Pi*. https://www.raspberrypi.com/ (accessed Sep. 25, 2022).

[2] "Automation, robotics, and the factory of the future | McKinsey," Sep. 12, 2022. https://www.mckinsey.com/business-functions/operations/our-insights/automation-robotics-and-the-factory-of-the-future (accessed Sep. 12, 2022).

[3] T. Hornyak, "Entertainment robots the latest craze worldwide as the pandemic rages on," *CNBC*, Sep. 12, 2022. https://www.cnbc.com/2021/01/09/entertainment-robots-popularity-continues-to-grow-amid-covid-19.html (accessed Sep. 12, 2022).

[4] cole P. F. de Lausanne, "A Robot to Beat Humans at Foosball," Sep. 12, 2022. https://cacm.acm.org/news/167371-a-robot-to-beat-humans-at-foosball/fulltext (accessed Sep. 12, 2022).

[5] M. C. T. Hollingshead, “BYU-created A.I.-powered foosball table goes head to head with humans,” *News*, Apr. 25, 2016. https://news.byu.edu/news/byu-created-ai-powered-foosball-table-goes-head-head-humans (accessed Sep. 12, 2022).
[6] S. Bambach and S. Lee, “Real-Time Foosball Game State Tracking,” p. 5.
[7] “Why We Built a Neuromorphic Robot to Play Foosball,” *IEEE Spectrum*, Feb. 27, 2022. https://spectrum.ieee.org/robotic-foosball-table (accessed Sep. 12, 2022).
[8] “SIMPLELINK-MSP432-SDK Software development kit (SDK) | TI.com.” https://www.ti.com/tool/SIMPLELINK-MSP432-SDK (accessed Sep. 25, 2022).
[9] Raspberry Pi Ltd, “Raspberry Pi 4 Model B specifications,” *Raspberry Pi*. https://www.raspberrypi.com/products/raspberry-pi-4-model-b/ (accessed Dec. 09, 2022).
[10] “SIMPLELINK-MSP432-SDK Software development kit (SDK) | TI.com,” Sep. 26, 2022. https://www.ti.com/tool/SIMPLELINK-MSP432-SDK (accessed Sep. 25, 2022).
[11] “C documentation — DevDocs.” https://devdocs.io/c/ (accessed Dec. 10, 2022).
[12] “Linux.org,” *Linux.org*, Sep. 26, 2022. https://www.linux.org/ (accessed Sep. 25, 2022).
[13] “Real Time Linux,” Sep. 27, 2022. https://wiki.linuxfoundation.org/realtime/start (accessed Sep. 26, 2022).
[14] “Part I - Video for Linux API — The Linux Kernel documentation.” https://www.kernel.org/doc/html/v4.9/media/uapi/v4l/v4l2.html (accessed Dec. 10, 2022).
[15] mattzzw, “Yet another YUV viewer.” May 28, 2019. Accessed: Dec. 11, 2022. [Online]. Available: https://github.com/mattzzw/yay
[16] “Simple DirectMedia Layer - Homepage.” https://www.libsdl.org/ (accessed Dec. 11, 2022).
[17] “Welcome to Python.org,” *Python.org*, Sep. 26, 2022. https://www.python.org/ (accessed Sep. 25, 2022).
[18] “MATLAB - MathWorks.” https://www.mathworks.com/products/matlab.html (accessed Dec. 10, 2022).
[19] “CCSTUDIO IDE, configuration, compiler or debugger | TI.com,” Sep. 12, 2022. https://www.ti.com/tool/CCSTUDIO (accessed Sep. 12, 2022).
[20] “KiCad EDA,” Sep. 12, 2022. https://www.kicad.org/ (accessed Sep. 12, 2022).
[21] “FreeDFM - A Service of Advanced Circuits.” https://www.my4pcb.com/net35/FreeDFMNet/FreeDFMHome.aspx (accessed Dec. 10, 2022).
[22] “Fusion 360 | 3D CAD, CAM, CAE, & PCB Cloud-Based Software | Autodesk,” Sep. 12, 2022. https://www.autodesk.com/products/fusion-360/overview?term=1-YEAR&tab=subscription (accessed Sep. 12, 2022).
[23] “Ultimaker Cura: Powerful, easy-to-use 3D printing software,” *https://ultimaker.com*. https://ultimaker.com/software/ultimaker-cura (accessed Dec. 10, 2022).
[24] “Ultimaker 3,” *https://ultimaker.com*. https://ultimaker.com/3d-printers/ultimaker-3 (accessed Dec. 10, 2022).
[25] “Replicator+ Desktop 3D Printer,” *MakerBot*. https://www.makerbot.com/3d-printers/replicator/ (accessed Dec. 10, 2022).
[26] “Lacy Student Experiential Center (Lacy Hall) | Makergrounds,” Sep. 27, 2022. https://makergrounds.virginia.edu/location/lacy-student-experiential-center-lacy-hall (accessed Sep. 26, 2022).
[27] “Footprint of a Microcontroller - STMicroelectronics,” Sep. 12, 2022. https://www.st.com/content/st_com/en/about/st_approach_to_sustainability/sustainability-

priorities/sustainable-technology/eco-design/footprint-of-a-microcontroller.html (accessed Sep. 12, 2022).
[28] O. US EPA, “Cleaning Up Electronic Waste (E-Waste),” Mar. 18, 2014. https://www.epa.gov/international-cooperation/cleaning-electronic-waste-e-waste (accessed Sep. 26, 2022).
[29] J. Nyika, F. M. Mwema, R. M. Mahamood, E. T. Akinlabi, and T. Jen, “A five-year scientometric analysis of the environmental effects of 3D printing,” *Adv. Mater. Process. Technol.*, vol. 8, no. sup2, pp. 564–574, Sep. 2022, doi: 10.1080/2374068X.2021.1945267.
[30] J. Nyika, F. M. Mwema, R. M. Mahamood, E. T. Akinlabi, and T. Jen, “Advances in 3D printing materials processing-environmental impacts and alleviation measures,” *Adv. Mater. Process. Technol.*, vol. 8, no. sup3, pp. 1275–1285, Oct. 2022, doi: 10.1080/2374068X.2021.1945311.
[31] asdf asdf, “The Lifespan & Recyclability Of Aluminium And Steel,” *Metals Warehouse*, Apr. 09, 2021. https://www.metalswarehouse.co.uk/lifespan-recyclability-aluminium-steel/ (accessed Dec. 10, 2022).
[32] D. M. Threlkel, “NEMA Enclosure Types,” no. 17, p. 9.
[33] “IPC Board Design Standards,” *IPC International, Inc.*, Oct. 01, 2021. https://www.ipc.org/ipc-board-design-standards (accessed Sep. 26, 2022).
[34] “1910.212 - General requirements for all machines. | Occupational Safety and Health Administration.” https://www.osha.gov/laws-regs/regulations/standardnumber/1910/1910.212 (accessed Sep. 26, 2022).
[35] “NFPA 70®: National Electrical Code®.” https://www.nfpa.org/codes-and-standards/all-codes-and-standards/list-of-codes-and-standards/detail?code=70 (accessed Sep. 12, 2022).
[36] webmaster, “Embedded C Coding Standard,” May 26, 2016. https://barrgroup.com/embedded-systems/books/embedded-c-coding-standard (accessed Dec. 13, 2022).
[37] T. C. Starker, “Broadcast-ready Table Sports System,” US 11395975 B2, Jul. 26, 2022
[38] G. Curry, “Camera-based tracking and position determination for sporting events using event information and intelligence data extracted in real-time from position information,” US9185361B2, Nov. 10, 2015
[39] Ajay, Jain, “Probabilistic Prediction Of Dynamic Object Behavior For Autonomous Vehicles,” US 11521396 B1, Dec. 06, 2022
[40] “2.10. YUV Formats — The Linux Kernel documentation.” https://www.kernel.org/doc/html/v4.8/media/uapi/v4l/yuv-formats.html (accessed Dec. 10, 2022).
[41] “Slotted Optical Switch OPB830 and OPB840 (L and W).” https://www.ttelectronics.com/TTElectronics/media/ProductFiles/Datasheet/OPB830_OPB840.pdf (accessed Dec. 13, 2022).
[42] “Amazon.com: DC 12V Power Supply 16.5A AC 96V-240V Converter DC Adapter Universal Regulated Switching Power Supply 12Volt 200W LED Power Supply for LED Strip,CCTV, Radio, Computer Project (12V 16.5A 200W) : Electronics.” https://www.amazon.com/96V-240V-Converter-Universal-Regulated-Switching/dp/B01HO76O0G?th=1 (accessed Dec. 13, 2022).
[43] “DRV8935 Quad Half-Bridge Driver With Integrated Current Sense.” https://www.ti.com/lit/ug/sprugp1/sprugp1.pdf?ts=1670721846559&ref_url=https%253A%252F%252Fwww.google.com%252F (accessed Dec. 10, 2022).

[44] "BTN8962TA High Current PN Half Bridge." https://www.infineon.com/dgdl/BTN8962TA-Data-Sheet-rev10-Infineon.pdf?folderId=db3a30431ff98815012060aedcd46179&fileId=db3a30433fa9412f013fbe2d247a7bf5&ack=t (accessed Dec. 13, 2022).
[45] "CRE1 Series Isolated 1W Single Output Isolated DC-DC Converters." Accessed: Dec. 13, 2022. [Online]. Available: https://www.murata.com/products/productdata/8807029407774/kdc-cre1.pdf
[46] "AP2125 300mA, HIGH SPEED, EXTREMELY LOW NOISE CMOS LDO REGULATOR." https://www.diodes.com/assets/Datasheets/AP2125.pdf (accessed Dec. 13, 2022).
[47] "Mini V Gantry Kit," *OpenBuilds Part Store*. https://openbuildspartstore.com/mini-v-gantry-kit/ (accessed Dec. 13, 2022).
[48] "HiLetgo 10pcs Micro Limit Switch KW12-3 AC 250V 5A SPDT 1NO 1NC Micro Switch Normally Open Close Limit Switch with Roller Lever Arm Black: Amazon.com: Industrial & Scientific." https://www.amazon.com/HiLetgo-KW12-3-Roller-Switch-Normally/dp/B07X142VGC/ (accessed Dec. 13, 2022).
[49] "OpenBuilds Part Store," Sep. 27, 2022. https://openbuildspartstore.com/ (accessed Sep. 26, 2022).
[50] D. Collins, "How to calculate motor drive torque for belt and pulley systems," *Linear Motion Tips*, Jun. 07, 2019. https://www.linearmotiontips.com/how-to-calculate-motor-drive-torque-for-belt-and-pulley-systems/ (accessed Dec. 13, 2022).
[51] "Pololu - 6.3:1 Metal Gearmotor 37Dx65L mm 12V with 64 CPR Encoder (Helical Pinion)." https://www.pololu.com/product/4757 (accessed Dec. 12, 2022).
[52] "Pololu - 31:1 Metal Gearmotor 20Dx41L mm 12V CB with Extended Motor Shaft." https://www.pololu.com/product/3487 (accessed Dec. 12, 2022).
[53] "MAKERELE Outdoor Junction Box Waterproof ABS Plastic Enclosure Electrical Project Boxes White 10"×7.9"×3.1"inch (255×200×80mm) - - Amazon.com." https://www.amazon.com/dp/B097XBZZV1 (accessed Dec. 13, 2022).

Appendix

Table 9: Class Budget Detailed Usage

Order #	Item	Supplier	Unit Cost	Quantity	Total Cost
1	296-DRV8935PPWPRCT-ND	Digikey	$4.56	1	$4.56
1	BTN8962TAAUMA1INCT-ND*	Digikey	$0.00	4	$0.00
				Order #1	$4.56
2	4757 - 6.25:1 Metal Gearmotor	Pololu	$51.95	1	$51.95
2	3487 - 31.25:1 Metal Gearmotor	Pololu	$29.95	1	$29.95
2	3499 - Magnetic Encoder Pair	Pololu	$8.95	1	$8.95
2	550 - Smooth Idler Pulley Kit	OpenBuilds	$5.99	1	$5.99
2	570 - Idler Pulley Plate	OpenBuilds	$6.99	1	$6.99
2	285-LP - V-Slot 20x40 Linear Rail	OpenBuilds	$3.99	1	$3.99
2	1185-Set - Mini V Gantry Kit	OpenBuilds	$34.99	1	$34.99
2	712 - Timing Pulley	OpenBuilds	$7.99	1	$7.99
2	626-By-the-Foot - Timing Belt	OpenBuilds	$3.49	4	$13.96
				Order #2	$164.76
3	P8062S-ND	Digikey	$0.49	1	$0.49
				Order #3	$0.49
4	365-1729-ND	Digikey	$4.66	2	$9.32
4	2449-SM3CQF3502L00-ND	Digikey	$1.12	4	$4.48
4	1276-6720-1-ND	Digikey	$0.10	8	$0.80
4	1276-1018-1-ND	Digikey	$0.10	4	$0.40
4	1276-1176-1-ND	Digikey	$0.10	8	$0.80
4	732-8604-1-ND	Digikey	$0.28	2	$0.56
4	1276-1051-1-ND	Digikey	$0.10	2	$0.20
4	732-8598-1-ND	Digikey	$0.11	1	$0.11
4	1276-1176-1-ND	Digikey	$0.10	1	$0.10
4	1276-1537-1-ND	Digikey	$0.10	1	$0.10
4	1276-1247-1-ND	Digikey	$0.10	1	$0.10
4	490-10675-1-ND	Digikey	$0.32	1	$0.32
4	1276-1942-1-ND	Digikey	$0.10	2	$0.20
4	1276-1096-1-ND	Digikey	$0.10	1	$0.10
4	SMAJ12CALFCT-ND	Digikey	$0.43	1	$0.43
4	754-1870-ND	Digikey	$0.42	2	$0.84
4	277-1667-ND	Digikey	$0.54	1	$0.54
4	A34825-ND	Digikey	$0.64	4	$2.56
4	A19433-ND	Digikey	$0.45	4	$1.80
4	732-11376-ND	Digikey	$0.68	1	$0.68

4	F1728-ND	Digikey	$0.42	3	$1.26
4	811-2473-1-ND	Digikey	$0.80	1	$0.80
4	P10.0KHCT-ND	Digikey	$0.05	10	$0.48
4	P5.10KHCT-ND	Digikey	$0.10	4	$0.40
4	P1.02KHCT-ND	Digikey	$0.10	4	$0.40
4	P8.06KHCT-ND	Digikey	$0.10	1	$0.10
4	RNCP0805FTD249RCT-ND	Digikey	$0.10	1	$0.10
4	P510HCT-ND	Digikey	$0.10	1	$0.10
4	P330HCT-ND	Digikey	$0.10	1	$0.10
4	CW103-ND	Digikey	$4.25	1	$4.25
4	36-5010-ND	Digikey	$0.42	3	$1.26
4	36-5011-ND	Digikey	$0.42	3	$1.26
4	36-5014-ND	Digikey	$0.42	4	$1.68
4	36-5012-ND	Digikey	$0.42	8	$3.36
4	AP2125N-3.3TRG1DICT-ND	Digikey	$0.39	1	$0.39
4	811-3198-ND	Digikey	$2.98	1	$2.98
4	478-1239-1-ND	Digikey	$0.10	6	$0.60
4	399-C0603C102K5RAC7867CT-ND	Digikey	$0.10	4	$0.40
4	311-3971-1-ND	Digikey	$0.11	4	$0.44
4	3757-PZ1AL3V6B_R1_00001CT-ND	Digikey	$0.50	6	$3.00
4	MHS30N-ND	Digikey	$5.06	2	$10.12
4	A1922-ND	Digikey	$0.28	2	$0.56
4	S6106-ND	Digikey	$1.26	2	$2.52
4	A113510-ND	Digikey	$0.74	4	$2.96
4	A129543CT-ND	Digikey	$0.10	2	$0.20
4	RNCP0603FTD10K0CT-ND	Digikey	$0.06	12	$0.73
4	2019-RK73H2ATTD44R2FCT-ND	Digikey	$0.10	2	$0.20
4	RMCF0603FT4K53CT-ND	Digikey	$0.10	2	$0.20
4	541-CRCW0603100KJNEBCT-ND	Digikey	$0.10	4	$0.40
4	RMCF0603JJ1K00CT-ND	Digikey	$0.10	4	$0.40
4	RNCP0603FTD180RCT-ND	Digikey	$0.10	4	$0.40
4	RMCF0603FT100RCT-ND	Digikey	$0.10	4	$0.40
4	36-5005-ND	Digikey	$0.42	2	$0.84
4	36-5001-ND	Digikey	$0.42	2	$0.84
4	36-5004-ND	Digikey	$0.42	4	$1.68
4	36-5002-ND	Digikey	$0.42	4	$1.68
4	A30978-ND	Digikey	$0.23	4	$0.92
4	A30980-ND	Digikey	$0.29	2	$0.58
				Order #4	$73.43
5	Custom PCB		$33.00	2	$66.00

				Order #5	$66.00
6	Driver Board Parts	WWW	$0.50	73	$36.50
6	Driver Board Fixed Cost	WWW	$10.00	1	$10.00
6	Header Board Parts	WWW	$0.50	63	$31.50
6	Header Board Fixed Cost	WWW	$10.00	1	$10.00
				Order #6	$88.00
7	P10KGCT-ND	Digikey	0.1	6	$0.60
7	311-18.0KHRCT-ND	Digikey	0.1	6	$0.60
7	RMCF0603FG1K00CT-ND	Digikey	0.1	6	$0.60
7	3757-PZ1AL3V6B_R1_00001CT-ND	Digikey	0.5	2	$1.00
7	2019-RK73Z1JTTDCT-ND	Digikey	0.1	4	$0.40
7	A130402CT-ND	Digikey	0.14	6	$0.84
7	311-22KGRCT-ND	Digikey	0.1	6	$0.60
				Order #7	$4.64
8	Parts to desolder	WWW	$0.50	20	$10.00
8	Parts to solder	WWW	$0.50	19	$9.50
8	Board fixed cost	WWW	$10.00	1	$10.00
				Order #8	$29.50
	*purchased by ECE department			Final Total	$431.38

Table 10: Estimated Values External to Budget

Item	Est. Unit Cost	Quantity	Total value
Raspberry Pi 4B (2 GB)	$45.00	1	$45.00
MSP432P401R	$43.95	1	$43.95
Ring Light	$14.99	1	$14.99
Aluminum 80/20 to hold camera	$11.99	1	$11.99
Microsoft LifeCam Webcam	$39.99	1	$39.99
WIN.MAX Mini Foosball Table	$41.99	1	$41.99
DC 12V 16.5A Power Supply	$31.99	1	$31.99
Power Supply Enclosure	$19.49	1	$19.49
Fuse Holder	$3.82	1	$3.82
Fuses	$3.90	1	$3.90
Limit Switches	$5.99	1	$5.99
6.25:1 Metal Gearmotor	$51.95	2	$103.90
31.25:1 Metal Gearmotor	$29.95	1	$29.95
Aluminum Mounting Hub for Shaft (2)	$8.49	1	$8.49
Black 14 AWG Wire	$6.63	1	$6.63

Red 14 AWG Wire	$6.63	1	$6.63
Clear Epoxy	$15.16	1	$15.16
Rocker Switch	$4.38	1	$4.38
Steel bracket material	$2.00	1	$2.00
3d print material	$50.00	1	$50.00
Nylon Inert Hex Locknut - M5 (10 Pack)	$0.99	1	$0.99
L Bracket Triple	$1.99	8	$15.92
L Bracket Single	$1.29	6	$7.74
M5 x 8mm Screws (10 Pack)	$0.99	11	$10.89
M5 Tee Nuts (10 Pack)	$2.99	11	$32.89
Double Tee Nut	$0.69	2	$1.38
Cube Corner Connector	$3.49	4	$13.96
Nylon Inert Hex Locknut - M3 (10 Pack)	$0.99	4	$3.96
V Slot 20x60 Linear Rail 500mm	$9.99	4	$39.96
M3 x 6mm Screws (10 pack)	$0.99	5	$4.95
M3 hex standoffs (4 pack) Female	$0.29	3	$0.87
M3 hex standoffs (4 pack) Female Male	$0.29	1	$0.29
Rubber Feet Set (4 pack)	$7.99	2	$15.98
Inside Outside Corner Bracket 60mm	$2.99	4	$11.96
Nylon Spacers	$2.09	1	$2.09
Aluminum Spacers	$2.49	1	$2.49
M5 x 10mm screws (10 pack)	$1.09	1	$1.09
M5 x 8mm screws (10 pack)	$0.99	2	$1.98
M5 x 6mm screws (10 pack)	$0.89	2	$1.78
M5 x 25mm screws (10 pack)	$1.39	1	$1.39
M5 Tee Nuts (10 pack)	$2.99	1	$2.99
Idler Pulley Plate	$6.99	1	$6.99
Double Tee Nut	$0.69	4	$2.76
Smooth Idler Pulley Kit	$5.99	1	$5.99
Mini V Gantry Kit	$34.99	1	$34.99
3GT Timing Pulley	$7.99	1	$7.99
V-Slot 20x40 Linear Rail 250mm	$3.99	1	$3.99
M3 x 6mm screws (10 pack)	$0.99	1	$0.99
M3 x 10mm screws (10 pack)	$0.99	3	$2.97
	Total Estimated Value		$732.46

Table 11: 10,000 Unit Cost

Item	Supplier	Unit Cost	Quantity	Total Cost
296-DRV8935PPWPRCT-ND	Digikey	2.43600	1	$2.44
BTN8962TAAUMA1INCT-ND	Digikey	3.60762	4	$14.43
365-1729-ND	Digikey	4.15250	2	$8.31
2449-SM3CQF3502L00-ND	Digikey	0.59122	4	$2.36
1276-6720-1-ND	Digikey	0.00271	8	$0.02
1276-1018-1-ND	Digikey	0.00497	4	$0.02
1276-1176-1-ND	Digikey	0.00464	8	$0.04
732-8604-1-ND	Digikey	0.16100	2	$0.32
1276-1051-1-ND	Digikey	0.00273	2	$0.01
732-8598-1-ND	Digikey	0.06000	1	$0.06
1276-1176-1-ND	Digikey	0.00573	1	$0.01
1276-1537-1-ND	Digikey	0.00428	1	$0.00
1276-1247-1-ND	Digikey	0.01247	1	$0.01
490-10675-1-ND	Digikey	0.06328	1	$0.06
1276-1942-1-ND	Digikey	0.00780	2	$0.02
1276-1096-1-ND	Digikey	0.01413	1	$0.01
SMAJ12CALFCT-ND	Digikey	0.08138	1	$0.08
754-1870-ND	Digikey	0.08302	2	$0.17
277-1667-ND	Digikey	0.36550	1	$0.37
A34825-ND	Digikey	0.29864	2	$0.60
A19433-ND	Digikey	0.16859	4	$0.67
732-11376-ND	Digikey	0.39300	1	$0.39
F1728-ND	Digikey	0.17044	1	$0.17
811-2473-1-ND	Digikey	0.30922	1	$0.31
P10.0KHCT-ND	Digikey	0.00436	10	$0.04
P5.10KHCT-ND	Digikey	0.00475	4	$0.02
P1.02KHCT-ND	Digikey	0.00475	4	$0.02
P8.06KHCT-ND	Digikey	0.00541	1	$0.01
RNCP0805FTD249RCT-ND	Digikey	0.00832	1	$0.01
P510HCT-ND	Digikey	0.00541	1	$0.01
P330HCT-ND	Digikey	0.00541	1	$0.01
CW103-ND	Digikey	2.76336	1	$2.76
AP2125N-3.3TRG1DICT-ND	Digikey	0.08181	1	$0.08
811-3198-ND	Digikey	2.21001	1	$2.21
478-1239-1-ND	Digikey	0.00731	6	$0.04
399-C0603C102K5RAC7867CT-ND	Digikey	0.00921	4	$0.04
311-3971-1-ND	Digikey	0.01571	4	$0.06
MHS30N-ND	Digikey	2.93574	2	$5.87

A1922-ND	Digikey	0.10233	2	$0.20
S6106-ND	Digikey	0.67470	2	$1.35
A113510-ND	Digikey	0.31044	4	$1.24
A129543CT-ND	Digikey	0.00778	2	$0.02
RNCP0603FTD10K0CT-ND	Digikey	0.00559	12	$0.07
2019-RK73H2ATTD44R2FCT-ND	Digikey	0.00428	2	$0.01
RMCF0603FT4K53CT-ND	Digikey	0.00170	2	$0.00
541-CRCW0603100KJNEBCT-ND	Digikey	0.00319	4	$0.01
RMCF0603JJ1K00CT-ND	Digikey	0.00134	4	$0.01
RNCP0603FTD180RCT-ND	Digikey	0.00609	4	$0.02
RMCF0603FT100RCT-ND	Digikey	0.00149	4	$0.01
A30978-ND	Digikey	0.09636	4	$0.39
A30980-ND	Digikey	0.11392	2	$0.23
P10KGCT-ND	Digikey	0.00426	4	$0.02
RMCF0603FG1K00CT-ND	Digikey	0.00149	4	$0.01
2019-RK73Z1JTTDCT-ND	Digikey	0.00185	2	$0.00
A130402CT-ND	Digikey	0.01447	4	$0.06
311-22KGRCT-ND	Digikey	0.00173	4	$0.01
		PCB Components		$45.70
Driver Board PCB	Pcbcart est.	1.25000	1	$1.25
Header Board PCB	Pcbcart est.	0.65000	1	$0.65
Driver Board Assembly	Est for size	0.83000	1	$0.83
Header Board Assembly	Est for size	0.40000	1	$0.40
		PCB & Assembly		$3.13
4757 - 6.25:1 Metal Gearmotor	Pololu	47.79000	2	$95.58
3487 - 31.25:1 Metal Gearmotor	Pololu	27.55000	2	$55.10
3499 - Magnetic Encoder Pair	Pololu	8.23000	2	$16.46
		Motors and Encoders		$167.14
550 - Smooth Idler Pulley Kit	OpenBuilds	5.99000	2	$11.98
570 - Idler Pulley Plate	OpenBuilds	6.99000	2	$13.98
285-LP - V-Slot 20x40 Linear Rail	OpenBuilds	3.99000	2	$7.98
1185-Set - Mini V Gantry Kit	OpenBuilds	34.99000	2	$69.98
712 - 3GT Timing Pulley	OpenBuilds	7.99000	2	$15.98
626-By-the-Foot - Timing Belt	OpenBuilds	3.49000	4	$13.96
		Extending Gantries		$133.86
Limit Switches		$5.99	1	$5.99
Aluminum Mounting Hub for Shaft (2)		$8.49	1	$8.49
Black 14 AWG Wire		$0.27	1	$0.27

Red 14 AWG Wire	$0.27	1	$0.27
Clear Epoxy	$4.00	1	$4.00
Rocker Switch	$4.38	1	$4.38
Steel bracket material	$2.00	1	$2.00
Plastic mold material	$5.00	1	$5.00
Nylon Inert Hex Locknut - M5 (10 Pack)	$0.99	1	$0.99
L Bracket Triple	$1.99	8	$15.92
L Bracket Single	$1.29	6	$7.74
M5 x 8mm Screws (10 Pack)	$0.99	10	$9.90
M5 Tee Nuts (10 Pack)	$2.99	10	$29.90
Double Tee Nut	$0.69	2	$1.38
Cube Corner Connector	$3.49	4	$13.96
Nylon Inert Hex Locknut - M3 (10 Pack)	$0.99	4	$3.96
V Slot 20x60 Linear Rail 500mm	$9.99	4	$39.96
M3 x 6mm Screws (10 pack)	$0.99	5	$4.95
M3 hex standoffs (4 pack) Female	$0.29	3	$0.87
M3 hex standoffs (4 pack) Female Male	$0.29	1	$0.29
Rubber Feet Set (4 pack)	$7.99	2	$15.98
Inside Outside Corner Bracket 60mm	$2.99	4	$11.96
Nylon Spacers	$2.09	1	$2.09
Aluminum Spacers	$2.49	1	$2.49
M5 x 10mm screws (10 pack)	$1.09	1	$1.09
M5 x 8mm screws (10 pack)	$0.99	2	$1.98
M5 x 6mm screws (10 pack)	$0.89	2	$1.78
M5 x 25mm screws (10 pack)	$1.39	1	$1.39
M5 Tee Nuts (10 pack)	$2.99	1	$2.99
Double Tee Nut	$0.69	4	$2.76
M3 x 6mm screws (10 pack)	$0.99	1	$0.99
M3 x 10mm screws (10 pack)	$0.99	3	$2.97
	Frame and other Mechanical		$208.68
Raspberry Pi 4B (2 GB)	$45.00	1	$45.00
MSP432P401R	$43.95	1	$43.95
Ring Light	$14.99	1	$14.99
Aluminum 80/20 to hold camera	$11.99	1	$11.99
Microsoft LifeCam Webcam	$39.99	1	$39.99
WIN.MAX Mini Foosball Table	$41.99	1	$41.99
DC 12V 16.5A Power Supply	$31.99	1	$31.99
Power Supply Enclosure	$19.49	1	$19.49
Fuse Holder	$3.82	1	$3.82

Fuses	$0.78	1	$0.78
		Misc	$253.99
		Total	$812.50

Figure 44: Motor Calculation Code

```python
# %%
from math import sqrt
import numpy as np
max_rod_travel = 78  # mm
end_player_distance = 274  # mm

ball_max_speed = 1300  # 284/0.23  # mm/s
fps = 30

# s (includes 33ms camera frame, computation, UART rate, and MSP processing)
worst_case_command_delay = 0.1
reaction_time = end_player_distance/ball_max_speed - worst_case_command_delay
avg_linear_speed = max_rod_travel/reaction_time  # mm/s
max_linear_speed = 2*avg_linear_speed  # speed at peak of triangle

print("Reaction Time: {} s".format(reaction_time))
print("Reaction Frames: {}".format(fps*reaction_time))
print("Average Linear Speed: {} mm/s".format(avg_linear_speed))
print("Max Linear Speed: {} mm/s".format(max_linear_speed))


# %%
# [(2,14), (2,30), (2,20), (3,20)]  # pitch (mm), teeth
pulley_options = [(3, 20)]

for pulley in pulley_options:
    belt_pitch = pulley[0]  # mm
    num_teeth = pulley[1]
    dpr_belt = num_teeth * belt_pitch  # mm/r
    desired_rpm_belt = max_linear_speed / dpr_belt * 60  # rpm
    print("Pitch: {}, Teeth: {}, RPM: {}".format(
        belt_pitch, num_teeth, desired_rpm_belt))
```

```
### ROTATIONAL CALCULATIONS ###
# Assumptions:
# - Rod starts at rest
# - Motor provides constant angular acceleration
# - Player has 45 degrees before making contact with ball
# v = w*r

desired_ball_speed = 1300  # mm/s
foosman_radius = 42.5  # mm
desired_rpm_rot = desired_ball_speed/foosman_radius * 60 / (2*np.pi)

print("Rotational RPM: {} rpm".format(desired_rpm_rot))

w_0 = 0
w_f = 10*np.pi  # rad/sec
starting_angles = [np.pi/4, np.pi/2]  # rad
delta_w = w_f - w_0  # rad/sec
w_avg = delta_w/2  # rad/sec

m_rod = 26  # g
m_player = 14  # g

l_rod = 443.5  # mm
l_player = 76  # mm

num_player = 3

I_rod = 1/12*(10**-3*m_rod)*(10**-3*l_rod)**2
I_player = 1/3*(10**-3*m_player)*(10**-3*l_player)**2
I_total = I_rod + I_player*num_player

for theta in starting_angles:
    delta_t = theta/w_avg
    alpha = delta_w/delta_t
    T_rot = alpha*I_total
    T_rot_kgcm = T_rot * 10.19716
    print("Rotational Torque: {} kg*cm, Theta: {} rad".format(T_rot_kgcm, theta))
```

```
### Torque Profile ###
# Assumptions:
# - Rod starts at rest
# - Motor provides constant acceleration
# - Symmetric triangular speed profile
# - Negligible mass and inertia from belt and pulleys (because I can't find the data)


mass_rod_assembly = 68  # g
mass_rotational_motor = 44  # 9.5 # g
encoder_mass = 1.5  # g
# Estimate based on ABS Plastic w/ 50% infill (includes motor to plate and rod to motor)
estimated_3D_print_mass = 15.25  # g
# From Fusion 360 Material Properties
estimated_gantry_mass = 87  # g

total_mass = (mass_rod_assembly + mass_rotational_motor + encoder_mass +
              estimated_3D_print_mass + estimated_gantry_mass) / 1000  # kg
print("Total Mass: {} kg".format(total_mass))


radius_pulley = 9.15  # mm

efficiency = 0.75

gravity = 9.8  # m/s^2

# Estimate based on: https://americanplasticscorp.com/products/delrin/#:~:text=The%20Delrin%2Dsteel%20coefficient%20of,loads%20and%20relative%20surface%20speeds.
rail_friction = 0.3  # coefficient of friction of guide

total_travel_time = max_rod_travel/avg_linear_speed  # s
acceleration_time = total_travel_time/2  # s
print("Acceleration Time: {} s".format(acceleration_time))


# https://www.linearmotiontips.com/how-to-calculate-motor-drive-torque-for-belt-and-pulley-systems/
F_axial = total_mass*gravity*rail_friction
```

```
115
116 inertia = total_mass * radius_pulley**2 * 10**-6  # kgm^2
117 alpha = 2*np.pi * desired_rpm_belt / (60*acceleration_time)  # rad/s^2
118
119 T_constant = F_axial * radius_pulley / (1000 * efficiency)  # Nm
120 T_acceleration = inertia * alpha  # Nm
121 T_total = T_constant + T_acceleration  # Nm
122 T_deceleration = T_constant - T_acceleration  # Nm
123
124 constant_vel_time = 0
125 T_rms = sqrt(T_total**2 * acceleration_time + T_constant**2*constant_vel_time +
126              T_deceleration**2*acceleration_time)/total_travel_time  # Nm
127 T_rms_kgcm = T_rms * 10.19716  # kg*cm
128
129 print("RMS Torque: {} Nm".format(T_rms))
130 print("RMS Torque: {} kg*cm".format(T_rms_kgcm))
```

Table 12: Capstone Fair Game Play Scores

Robot	Humans
1	0
9	1
2	3
9	0
4	3
2	0
3	0
1	0
1	0
9	4
9	1
9	3
2	0
9	5
5	0
6	9
9	3
9	3
7	7
3	0
9	4
9	2
6	0
9	8
5	9

9	7
4	1
9	5
9	3
3	2
9	3
9	4
9	4
208	94