AI Foosball

Foosbot's Final Design Report

Due: 4/29/2022

Michael Thompson – Team Leader

Slayter Teal – Team Purchasing Manager

Cole Mitchell

Johnny Enriquez

Jackson Law- Mechanical Team Leader

Jonathan Harris – Team Planner

Alex Rivera

Kenny Gipson

Hunter Collins

Garrison Locke

1 Table of Contents

2 Introduction and Problem Description

The primary objective of this project is to modify a foosball table so that one side is played by an AI and the other by people. Our goal for this project is for the table to be used as a teaching tool for future OSU students at a variety of different collegiate levels.

The central focus of our design is modularity so that future OSU students can quickly and easily get up to speed on the work we have done and make modifications of their own. We have achieved this by designing the table in such a way that components and software packages are easily interchangeable. This allows for future teams to create more advanced AI software, use a different camera if they want, or even replace the actuators with another configuration.

Below are 2 tables which list the target deliverables broken up into sections showing our electrical system requirements and mechanical requirements. The following report will provide a detailed description of our design, and how we met all the necessary requirements of the project.

Electrical
Goal Sensors
Scoreboard (GUI, Score reporting)
Ball Detection (Location/Velocity vector Prediction)
AI/Brain (Respond to and Kick balls moving at a minimum of 2 m/s)
Controls (Robot player response, correctly follow commands)
Standard Wall Plug-in

Table 1: Electrical Deliverables

Mechanical
Camera Stand
Roto-linear Actuator (360 degree spinning, 352mm Linear movement)
Actuator Table (Actuator assemblies on top, Electronics below)
Acrylic Side walls
Rod Covers
Vinyl field for table (OSU aesthetic)
Replace plastic foosball players (OSU aesthetic)

Table 2: Mechanical Deliverables

To track the position and movement of the foosball, we used an overhead camera and computer vison. For decision making, we used a finite state machine, and a centralized data server. We used CAN communication to send orders to the motors, and we used a novel spline gear assembly to facilitate independent linear and rotational rod movements.

3 Environmental Health and Safety

Though our project is safe by nature, there are still a few safety concerns we must consider. The concerns of highest significance are:

- Impact Hazard from rods moving up to **3.5 m/s**
- Pinching hazard between the rod bumpers and the side wall
- Pinching hazard in unshielded gear assembly.

Though there is a low risk of injury from any of these hazards, we wanted to minimize any risk to a user so that no harmful impact or pinching can occur during the normal play of the game.

Our strategies for mitigating these safety hazards include:

- Installing PVC covers over the exposed ends of the Robot-controlled rods on the user side of the table to prevent the rods from impacting a user.
- Installing acrylic panels in both the camera frame, and over the actuator assembly to prevent anyone placing their hands or fingers where they could be impacted or pinched.
- An emergency Stop Switch which will directly cut off power to the actuator motors.
- A Pause Game button in case the ball gets caught in a dead spot and someone needs to reach in.

Additionally, while fabricating our design, it was important that we followed all guidelines for both PPE and professional conduct in the fabrication shops. We didn't want to damage any University equipment or hurt ourselves through improper use of a tool or machine. We also were very careful to exercise proper use of grounding any time we are working with the electronic components of our design. We have a lot of sensitive and expensive computer components in our design, so it was vital that we followed proper procedures to avoid damaging critical components.

Our design also generated waste during the fabrication process. The most concerning waste is from the many 3D printed parts we will need to make. We always made sure that we disposed of any waste PLA in the proper location to avoid damage to wildlife or the environment.

Foosball is great because it allows people to come together and enjoy an engaging game which is a good distraction from the stresses of daily life. Since our project involves the modification of a popular, well-loved game such as foosball, it has a chance to shake up the recreation market if anyone ever chooses to monetize a design similar to ours. Short term, our table allows someone without a partner to enjoy a game of foosball on their own. Long term, and with the right networking capabilities and proper modifications, it could allow someone to play foosball with a friend on the other side of the world.

The Stakeholders of this project include:

- OSU CEAT
- Foosball Industry
- Mechatronics Hobbyists

These stakeholders were identified because this project is intended for future use and/or modification by future OSU CEAT students. There is no commercially available product like the table we are designing, so there is a potential market, which would impact the foosball industry. Our project has also been attempted before by several different teams and individuals, so what we learn from this project could potentially benefit future mechatronics hobbyists who attempt a similar undertaking.

4 Engineering Principles, Codes and Standards

4.1 Engineering Principles

The engineering principles we used are important in the design process and we have developed them during educational experiences or while working directly in the industry. The principles that were used in this project include many types of software and motion or object analysis. We designed all our CAD models in Solidworks using the 3D design software to accurately recreate our ideas into CAD models to design out our project. We used many types of motion and force analysis by using our knowledge base of statics and dynamics. These calculations are very important when considering the validity of an idea in the design process. Outside of human based calculations, we also used Finite Element Analysis through the software ANSYS. This allowed us to accurately analyze specific situations and gave us the strongest and weakest points in our design. Lastly and most importantly, during documentation we made sure to keep everything as standardized as possible between group members so there is no differentiation between each person's work. This will allow for future groups to easily organize and understand.

4.2 Electrical Standards

All the electrical standards were taken from the NEMA 70 which is currently the most updated version of the electrical standard codes. With hundreds of codes, only a handful of them apply to our project.

- Article 110: All electrical circuits or equipment must follow all the articles in the book
- Article 200.7(B): Any power supplies of less than 50 volts must follow a specific color scheme for its ground
- Article 210.20: All component needs to be rated to handle 125% amps than the amps we are using it at
- Article 250.4 (A): All electrical equipment will be connected to a ground
- Article 430.4: Part winding motors must be used properly
- Article 460: Proper use of capacitors
- Article 470: Proper use of resistors

4.3 Code Standards

With a project of this scale, it is crucial that appropriate software standards are followed during its development. This is necessary both to ensure our own success in meeting our deliverable goals but also to allow continued development by groups outside our own. With the goal of this project being use in a classroom, it is of the utmost importance that those using it see familiar styles and design choices throughout. Making continued design easy while also providing insight into commonly used tools and protocols in industry. While the project is set up to interface with any programming language, the vast majority of the code base will be written in python. In doing so we will be following the PEP 8 style guide which outlines standard practices for naming variables and methods, as well as the use of other python tools. These standards allow for consistent conventions across all files and with the addition of other good practices like proper layout, encapsulation, and documentation will create code that is clean, readable, and easy to follow.

Maintaining a code base of this size introduces several problems in consistency across multiple designers. GitHub will be used to mitigate the risks. GitHub is a version control platform specialized for maintaining code. It stores a remote copy of the program files while allowing multiple users to create local versions that can be edited and tested. When developments have been confirmed as beneficial, it allows for those changes to be added to the main version and accessed by the other members. This not only allows for coordination within our group but would allow future groups to continue collaboration while also having access to the development history of the project, providing more insight into its operation.

Further standards apply to the method passing between our different process operations. Connections between processors will be opened with sockets following TCP protocol. Sockets will be discussed in more detail later, but TCP consists of sending messages in the form of a byte stream with built in error correction, ensuring that the data arrives correctly at the other end. With the use of this protocol, our only concern is specifying the appropriate address and data, underlying systems can then handle the rest. Formatting the data of the messages will use JSON or Java-Script Object Notation. This is a textual representation of objects that can be easily serialized and sent as a byte-stream. All major programming languages provide packages for JSON manipulation and formatting data in this form is all that is required to communicate with any other segment of the software. Additional hardware communication protocols will be utilized, including CAN and UART, each receiving more detail further in the report.

4.4 Mechanical Standards

Codes and standards will be a valuable tool for us as it sets our constraints on the mechanical design of the project. The next value gained from these standards is we are now able to standardize all our documentation. We want future groups to be able to continue where we finish, which is why this is such an important part of our workload. We used a variety of industry standards set by certified organizations such as, The American Society of Mechanical Engineers, American National Standards Institute, and Internal Organization for Standardization. All of these standards listed below can be found or requested from the specific organizations.

- ASME Y14.5-2018(Geometric Dimensioning and Tolerancing)
    - Essential tool for communicating design intent, It establishes symbols, rules, definitions, requirements, defaults, and recommended practices for stating and interpreting GD&T and related requirements for use on engineering drawings, models defined in digital data files, and in related documents.

- ANSI/NEMA MG 1-2016(Motor Selection)

    - Assists users in the proper selection and application of motors and generators. Contains practical information concerning performance, safety, testing, and construction and manufacture of ac and dc motors and generators.

- ASME Y14.1 – 2020(Drawing Practices)

    - This Standard defines sheet sizes and formats for engineering drawings. Helps uniformity in all drawings

- ISO 17396(Belt Selection)
    - Specifies the principal characteristics of Timing belts and pulleys such as nominal belt tooth dimensions, belt tooth pitch spacing, belt length etc.

5 Overall Solution and Subsystems

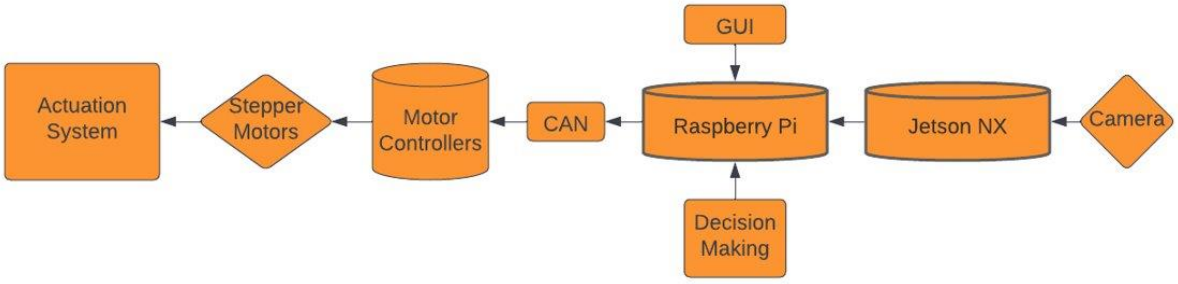


Figure 1: Overall Solution and Subsystem Flowchart

Our solution (shown above) starts with a image from the camera being processed on the Jetson. Once processed, the Jetson sends a ball location and vector to the server located on the Raspberry Pi. The decision making module takes the ball location from the server and generates rod movement commands and then places those commands back on the server. GUI takes user input to start and stop the game and display score. The CAN module takes the rod commands and sends them to the CAN bus if the game is not stopped or paused. It also takes the current position from the CAN bus and places that and the current score on the server. The motor controllers read the commands and then convert those commands to pulse commands for the stepper motors. The actuation system takes the motor movement and converts it to rotation and translation which results in the ball being kicked.

6 Mechanical System

6.1 Linear Actuation

The two biggest mechanical design concerns are how the foosball rods will be articulated both linearly and rotationally, our design is intended to decouple those two motions unlike other commonly found actuation systems. To control the translational motions, a modified belt sled was used. The system consists of a V-slot channel (20mm x 40mm x 500mm) fitted with a XL series timing belt/pulley, attached to the belt is a gantry cart that serves as a base for a custom aluminum arm that links the cart and foosball rod. The belt is powered by a Nema 23 high torque stepper motor that has a stalling torque of 3 newton meters. The aluminum arm houses two collared setscrew ball bearings to allow for the rod spin freely, these are held in place via pressure fitting. The maximum distance that any rod will need to travel is 352 millimeters, by having each system capable of moving up to 500 millimeters we ensured that the range of translational motion is more than covered.

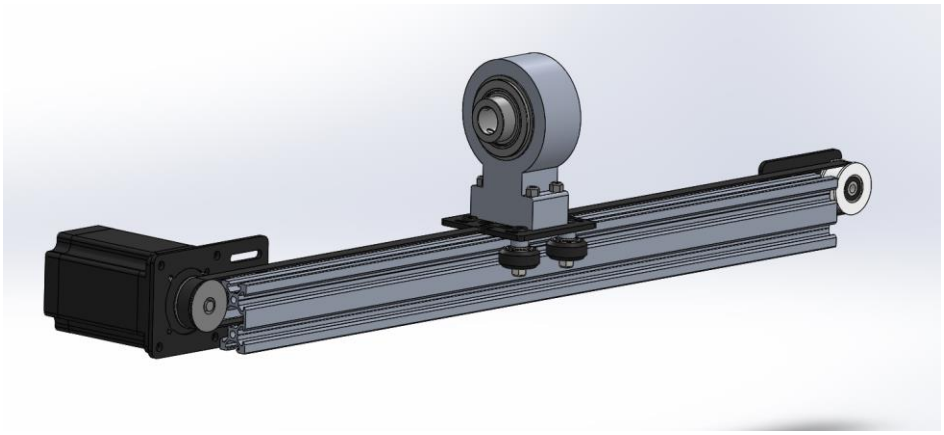


Figure 2: Linear Actuation System

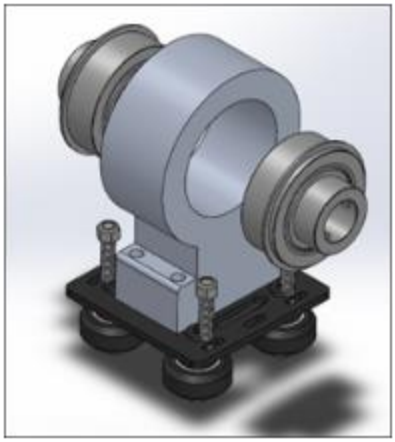


Figure 3: Gantry Cart with Arm Attachment

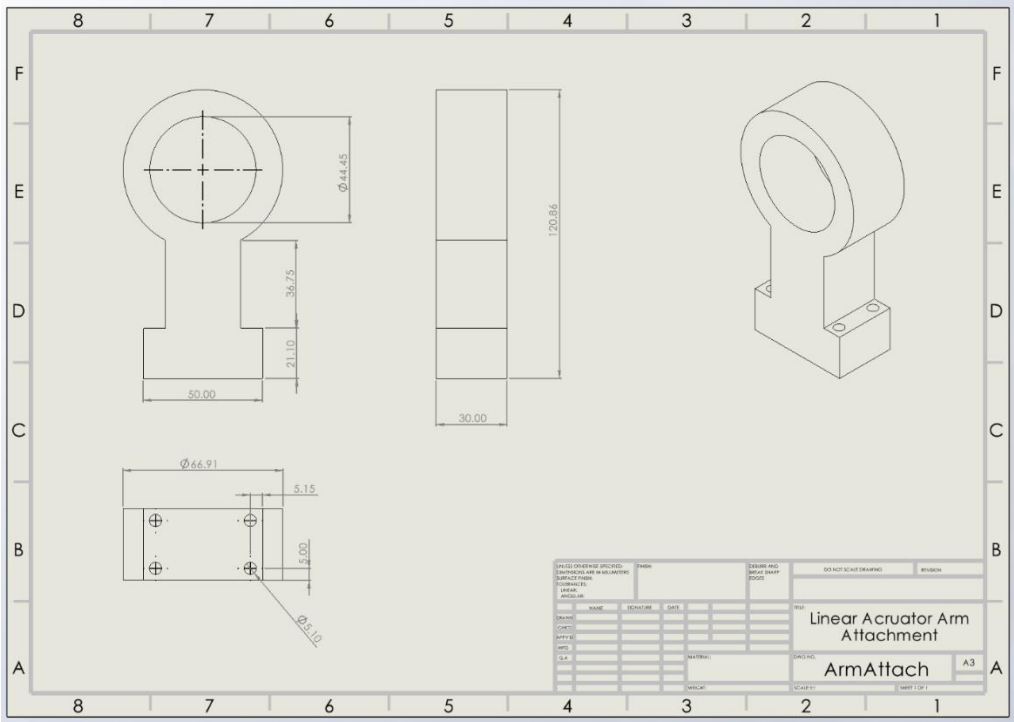


Figure 4: Arm Attachment Engineering Drawing

Additionally, to lower the amount of torque on our motors, the steel AI foosball rods were replaced by lighter aluminum rods. Doing so decreases the weight by nearly two-thirds as well as the torque.

6.2 Rotational Actuation

This is an overview of the rotational assembly. This will cover which actuation method that was decided to go with. How we will use different gears to obtain the motion we need, as well as the material for those gears that will be able to hold the forces, they are experiencing without breaking. How these gears will mesh with each other and the dimensioning of those gears. In the end we will touch on how all these things will be manufactured and put together with the selected material.

The actuation method we decided to go with is a system of three gears. The three gears that were decided for this system is 2 spur gears, one that attaches to the high-torque NEMA 23 motor and the other one slides directly onto the foosball rod itself. The third gear that was decided is a spline gear which essentially is a regular gear that is extruded to 15 inches to allow for the sliding of rod gear.

Getting into the material for the gears there were many things considered when choosing the material. These include but not limited to manufacturability, strength, supply, and cost. The material we decided to go with for the two regular gears are going to be 3D ONYX print that has a carbon fiber filling. This was chosen because they can be easily manufactured locally as well as they are relatively cheap in terms of buying something from McMaster, and they are very strong for the type of forces they will be experiencing. For the spline shaft we have decided to use PLA plastic, the reasoning is because these can be printed easily at the endeavor, and just like the other gears this spline material will be strong enough to withstand the forces experienced. These force

calculations will be shown in the strength section, as well as how many teeth these gears need to work efficiently.

Now to the meshing of the selected gears, this is important because it is imperative that the pressure angle and the pitch of the gears match or else our design will not work. For the two gears (on the rod and on the motor) they will be identical, the pitch diameter will be 3 inches and the number of teeth will be 15. For the spline gear we will be making the pitch diameter 2 inches and the number of teeth will be 10. For the pressure angle of the gears, they will all be set to 20°. We chose this angle because it is an industry standard. In terms of how these gears will be lined up and spaced accordingly we did the research to find that to space gears properly we need to add the two pitch diameters of the gears being looked at and divide that by 2.

$$pitch\ diameter = \frac{N_{teeth}}{P_{diametric}} \quad (1)$$

Lastly, the manufacturability of our rotational assembly is important to consider so that we can make what we have designed. For the spline shaft, both ends will be bored in the center an inch and a half with a diameter of 5/8''. Then we will have 4-inch metal dowel rods that will be pressure fit into those bored out holes in the spline. The other side of the dowel rod will be set into mounted ball bearings, which will then be set screwed. This will allow for a full degree of motion for rotation as well as being very low in friction. For the gear on the motor, we will be also using a set screw to fasten it to the motor drive. For the gear that is going onto the foosball rod, we will be pinning it on to the rod, so a hole will go all the way through the gear hub and the foosball rod so that we can stick a pin all the way through to ensure no movement.

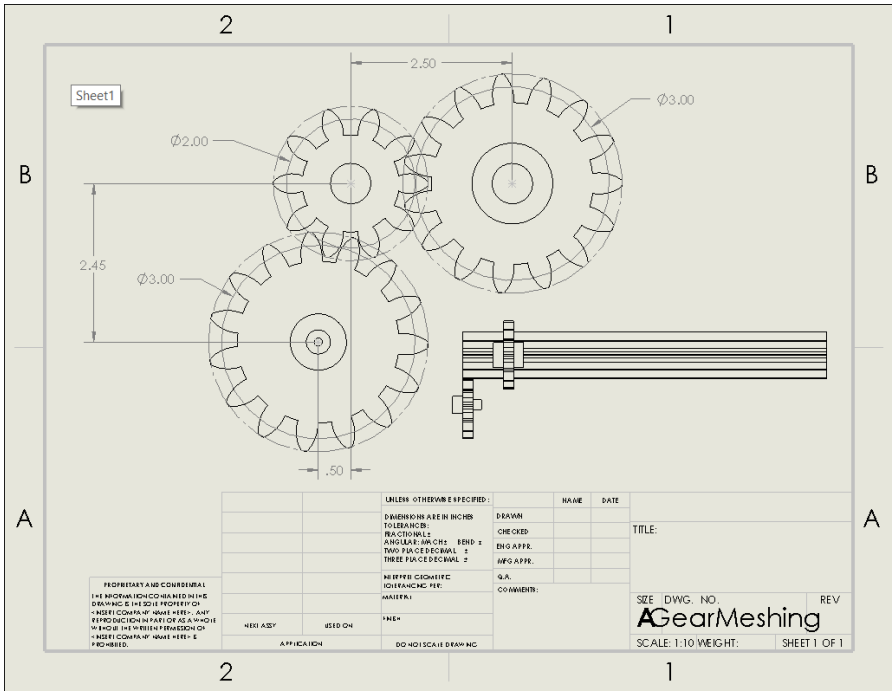


Figure 5: Gear Meshing CAD Drawing

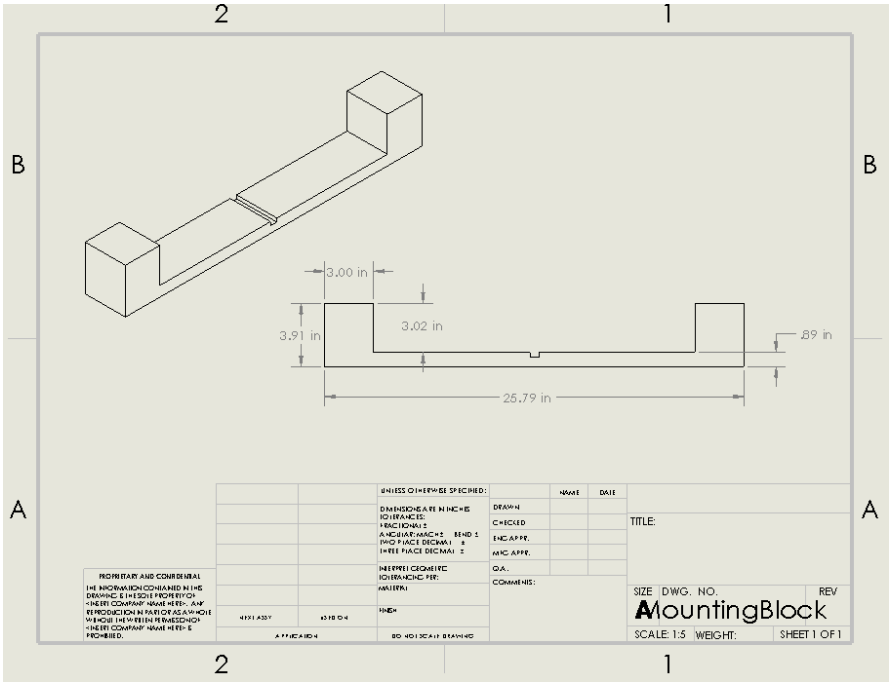


Figure 6: Rotational Gears Mounting Block Drawing

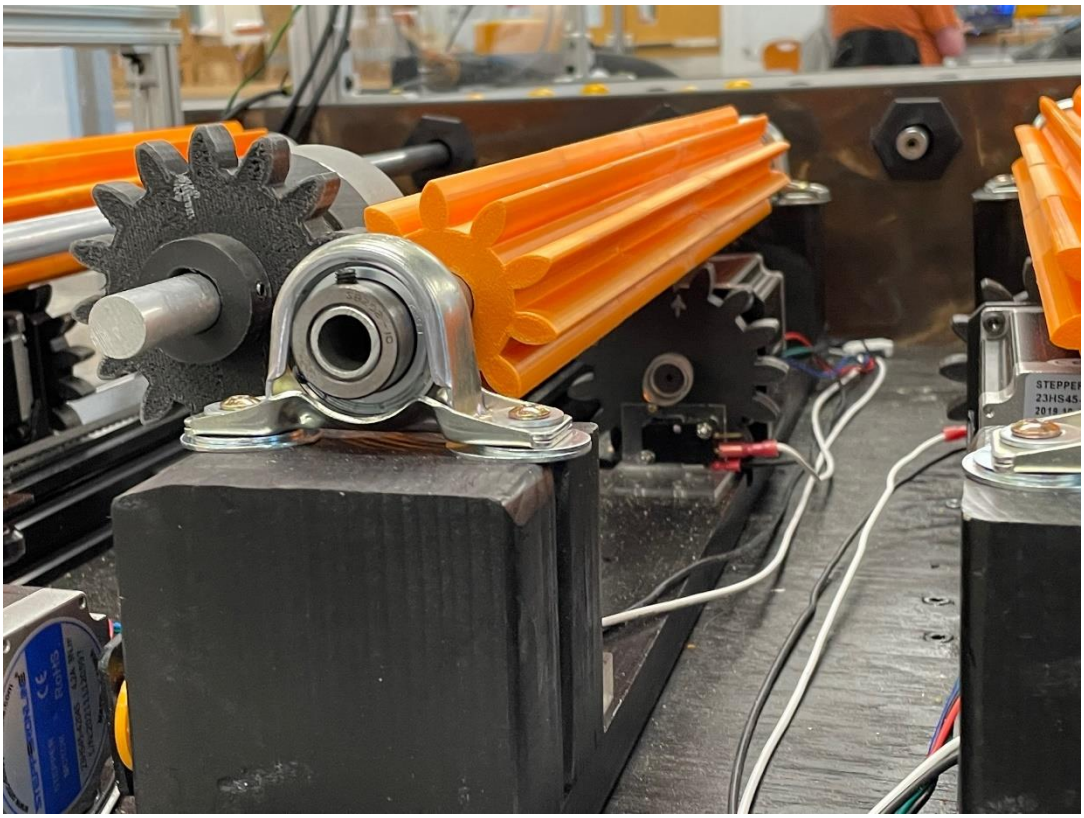


Figure 7: Rotational System as built

6.3 Motor Calibration

To Calibrate the motors accurately, we used bump switches which send a signal to the server showing the motors are at the zero position. For the linear motion we mounted a miniature snap-acting switch on the rotational motor assembly where it would be hit by the linear gantry at the inner limit of its actuation distance. To solve the more difficult problem of rotational zeroing a cam system was designed to attach to rotational motor gears.

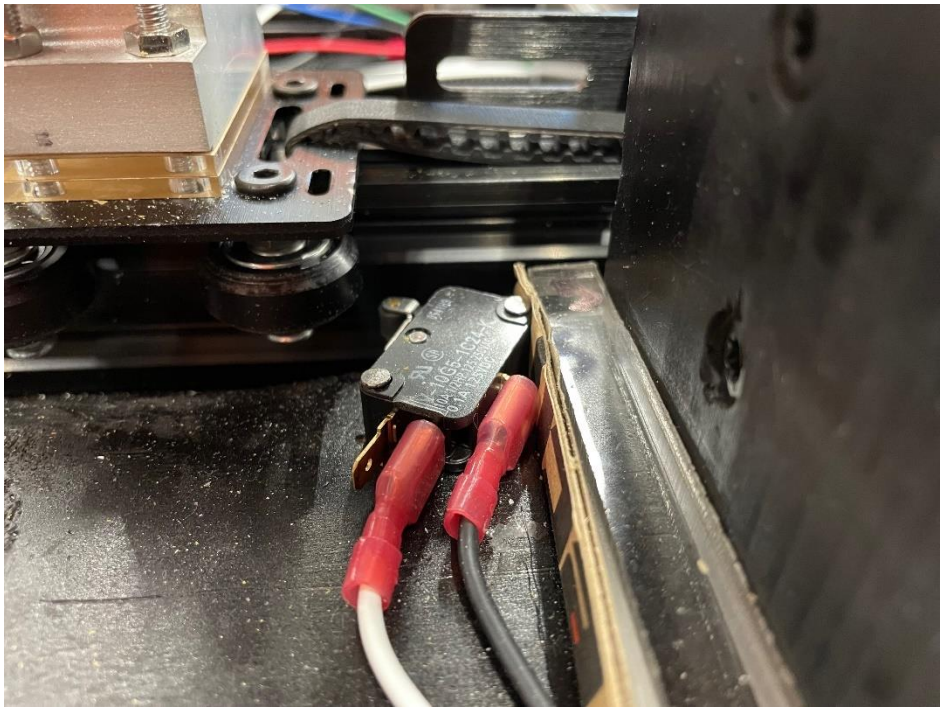


Figure 8: Linear Zeroing Assembly

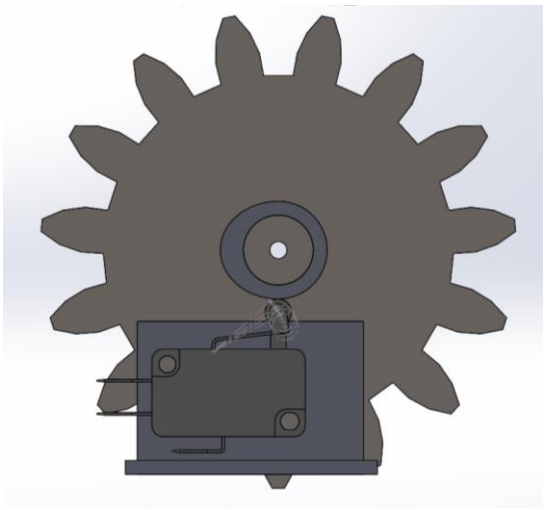


Figure 9: Rotational Zeroing CAD Assembly

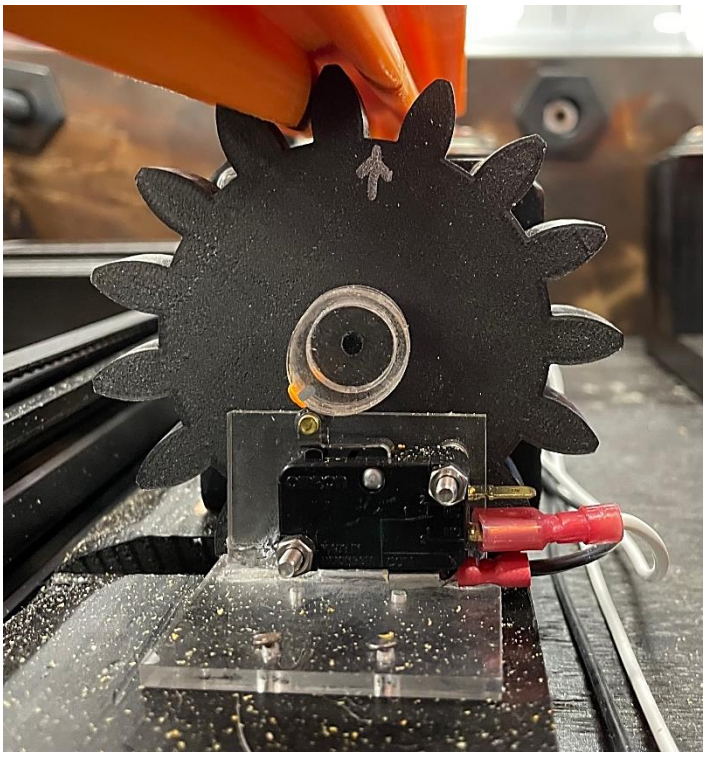


Figure 10: Rotational Zeroing Assembly as Built

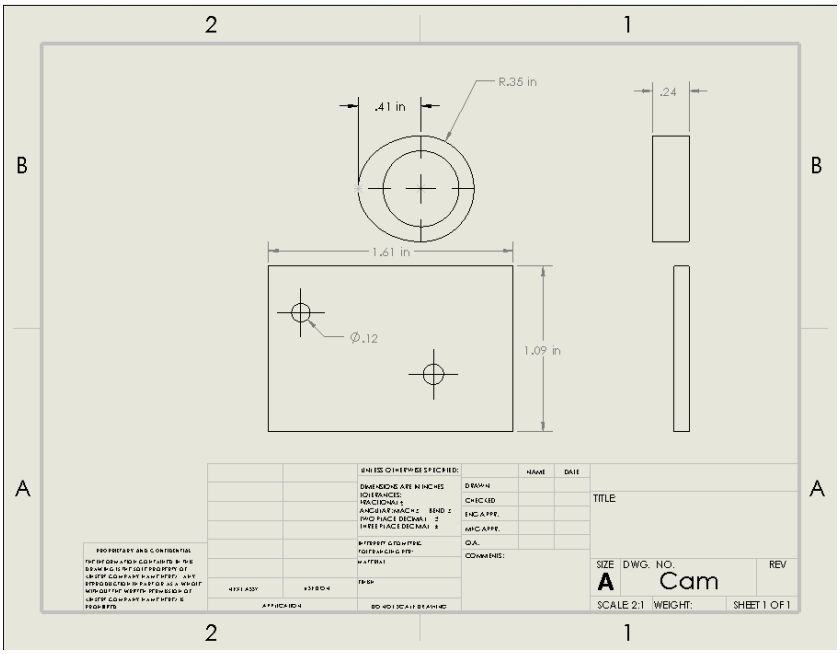


Figure 11: Cam Drawing

6.4 Actuator Table

There were several needs which influenced our decision to build an actuation table instead of mounting the assemblies to the side of the foosball table itself. These factors were:

- Need to minimize camera vibration.
- Need to be able to easily access electronic components
- Space for our large stepper motors and mounting hardware.

Because of these needs, it became clear during our detailed design phase that building a separate actuator table was the most viable path forward.

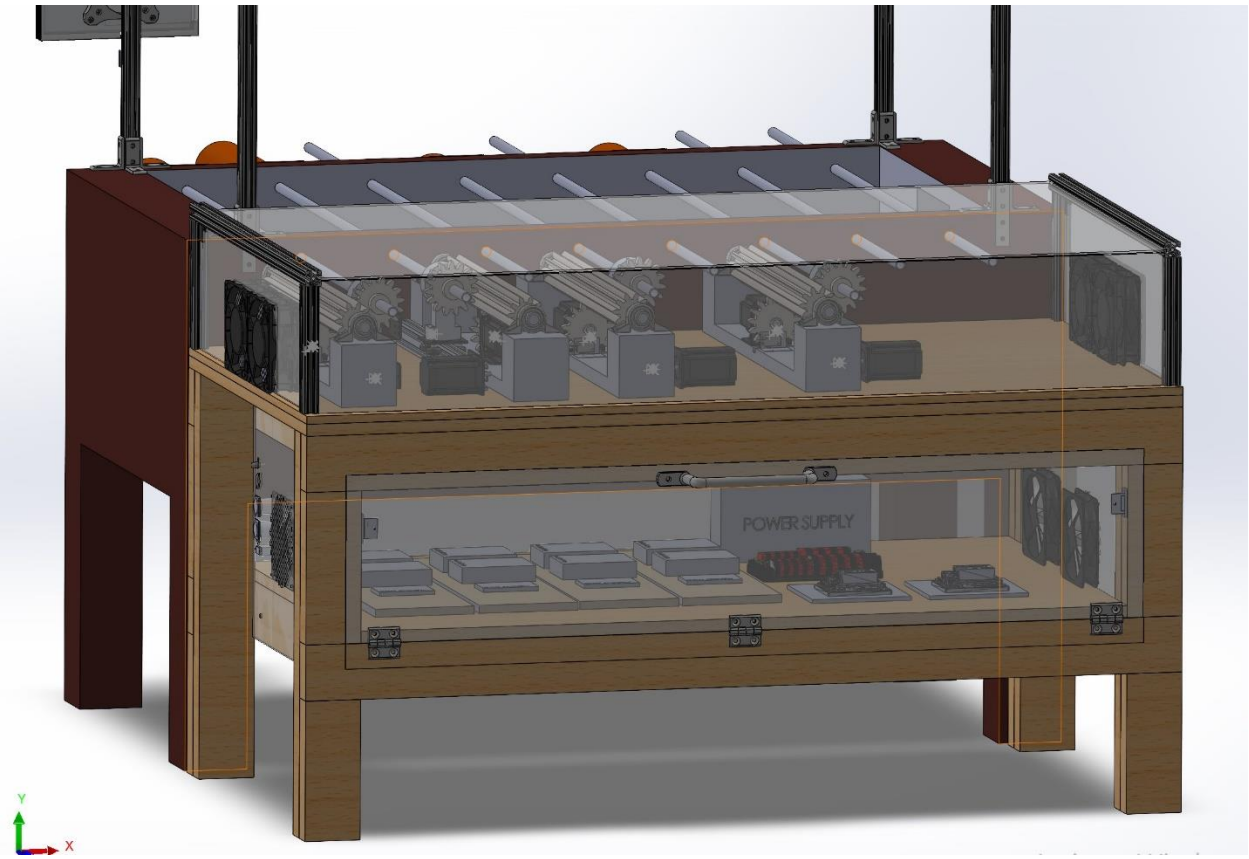


Figure 12: Actuator Table Assembly CAD Model

Figure 13: Constructed Table

Figure 14:Complete Actuator Table as Built

The actuator table frame was constructed from an arrangement of 2”x4” studs and utilized plywood panels for the mounting surfaces and acrylic panels for the side walls. The table houses the actuator assemblies on the top surface and houses all the electronic components in the cabinet below. An acrylic door allows for easy access to all the electronic components, while also allowing the electronics to be visible during the operation of the table. The top portion of the table has an acrylic cover, which sits over the entire actuator assembly, and prevents anyone from reaching into the gears and getting pinched, while also allowing for clear viewing of the actuators during operation.

Additionally, due to the wood construction of the actuator table, its mass is much greater than the moving components in our design. This is great because it significantly reduces any vibration transferred to the camera frame, and thus, the camera. And after thorough testing, the camera has no issues with vibration affecting the picture.

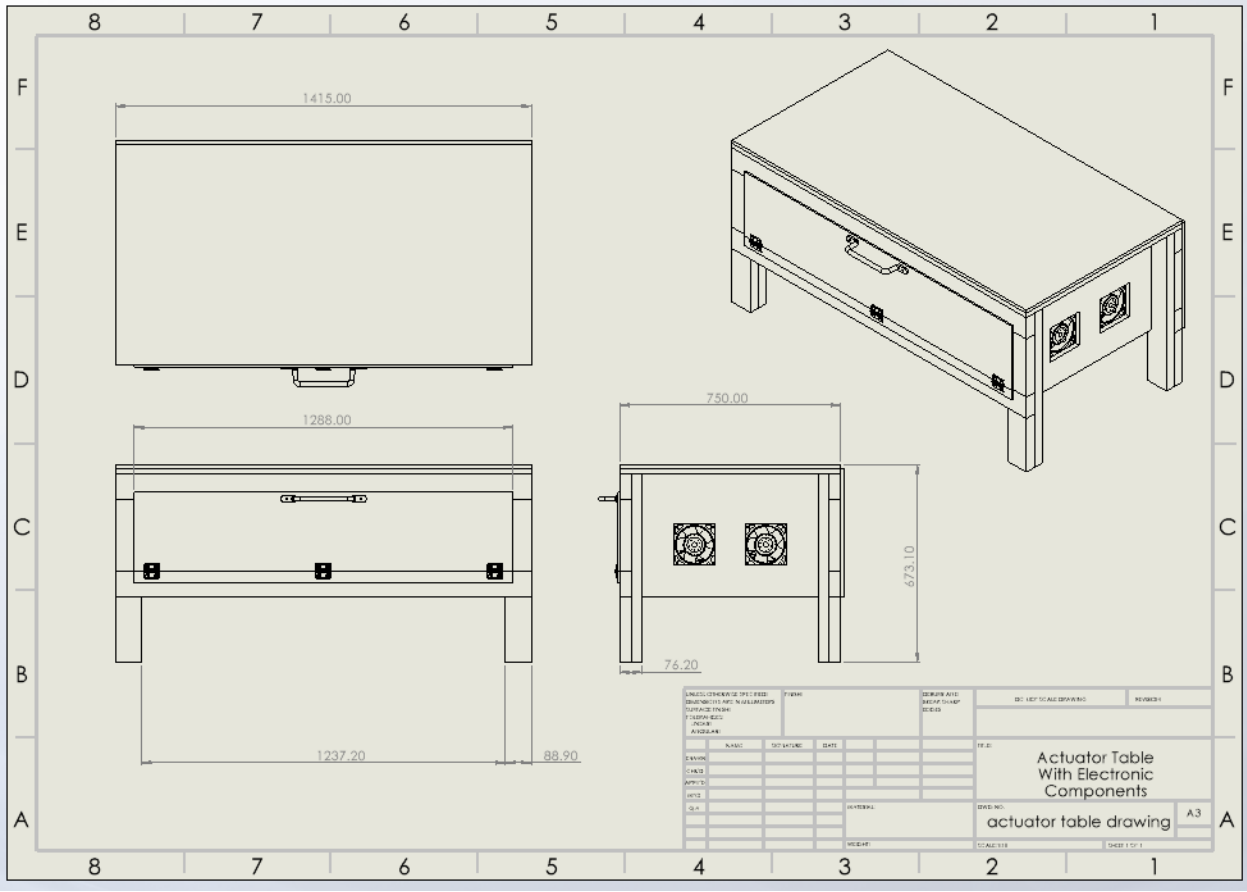


Figure 15: Actuator Table Drawing

6.5 Camera Frame and Touch Screen

It was necessary to design a way to suspend the camera above the table and capture the entire play area so the ball-tracking system can work properly. Additionally, different cameras have different focal lengths, which means the height of the frame must be adjustable to account for these differences between cameras. The frame that has been designed utilizes 1-inch T-Slotted

Aluminum Railing to suspend the camera and not add much weight to the system while maintaining enough structural integrity to prevent the camera from moving an excessive amount. The top portion of the frame has a completely adjustable height that slides up and down the vertical rails to ensure that the frame will work with whatever camera and lens is used, and the 3D Printable camera mount that we designed will be able to press fit into the T-Channel rails. The final camera mount was additionally secured with zip ties from the handles on the sides for added stability.

Figure 16: Camera Frame Model

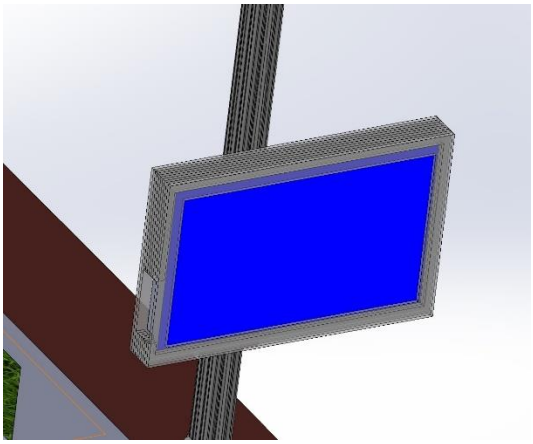


Figure 17: Touch Screen CAD Model

Figure 18: Camera Frame as Built with Touch Screen

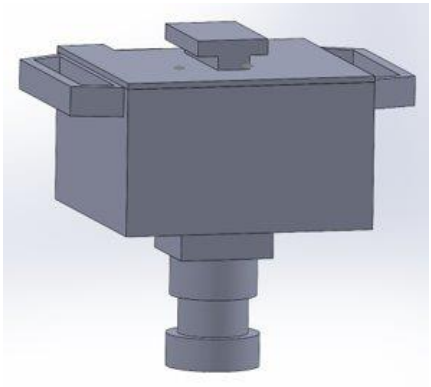


Figure 19: Camera Mount Model

Figure 20: Camera Mount

6.6 Rod Covers

To prevent the robot-side foosball rods from impacting a human user or someone watching the game, we designed covers which will mount to the side of the foosball table and fully enclose the rods. These are the aspect of our design which changed the most since the design phase. After fabricating rod covers from 2.5" square steel tubing, we determined that it would be best if we made our rod covers from a lighter material which would not give users the impression that the rod covers could support a person's weight. We instead decided to go with 3" diameter PVC pipe rod covers of varying lengths, corresponding to the actuation distance of each individual rod. This had the added benefit of providing fewer obstructions during play while also providing the desired level of protection and not giving the impression of load bearing capabilities. We also included labels on the ends reading "DO NOT LEAN" to further discourage users from placing their weight on the rods and damaging the table.

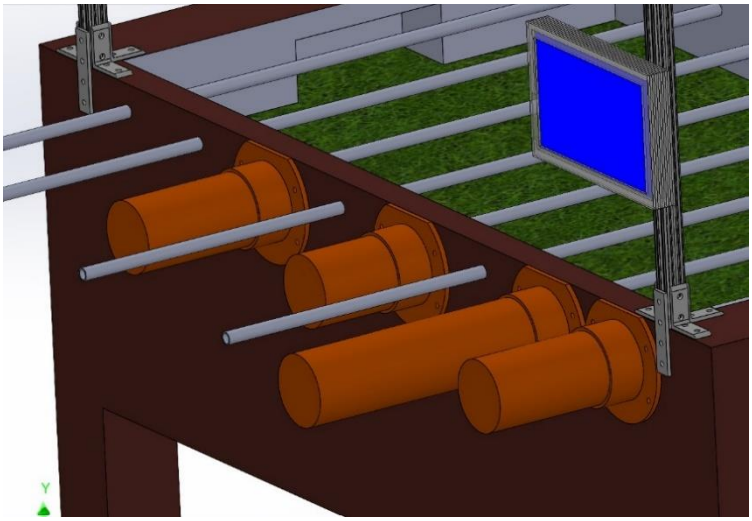


Figure 21:Rod Covers in Full CAD Model

Figure 22: Photo of Rod Covers as Built

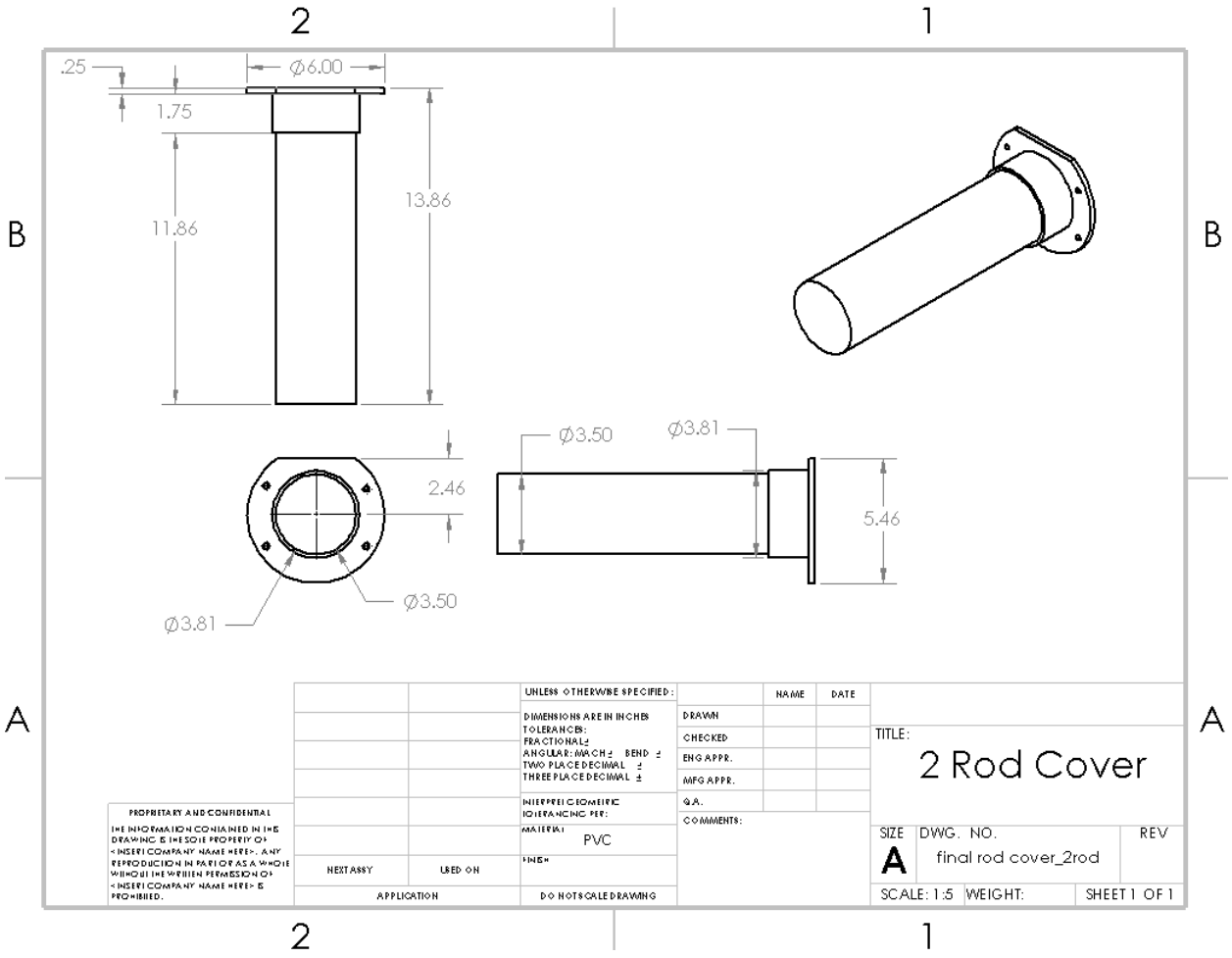


Figure 23: 2-Rod Cover Drawing

7 Electrical System

7.1 Hardware

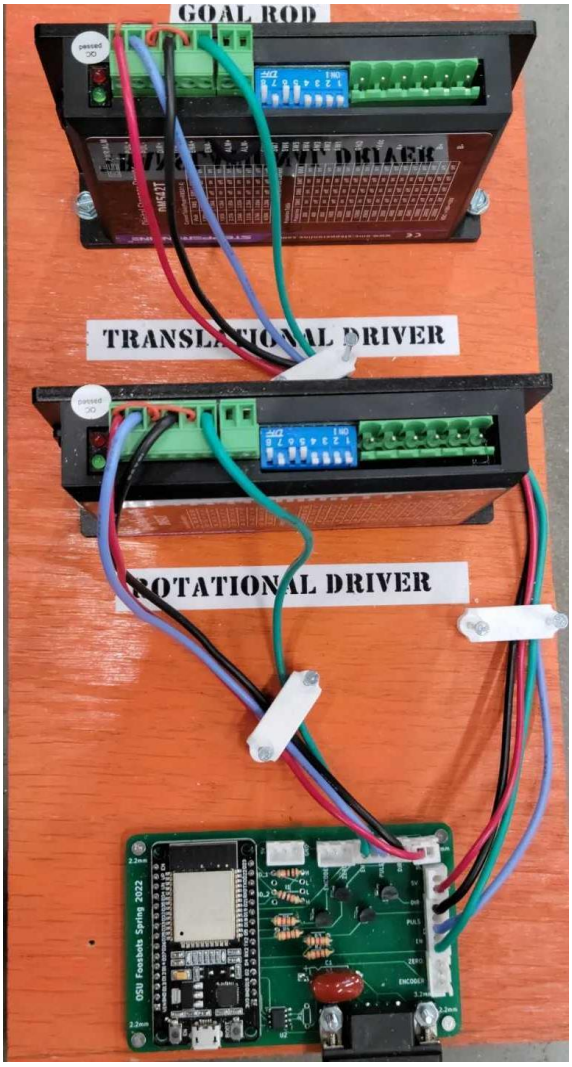


Figure 24: Control Hardware

7.1.1 Micro Controller

Micro Controller	Hardware PWM	Built in CAN	Processor Speed (MHz)	Multi-Tasking	GPIO		Price
					Digital	Analog	
Arduino mega	15	No	16	No	54	16	$40.30
Arduino Micro	7	No	16	No	20	12	$20.70
MyRIO	6	No	667	Yes	32	0	$1,093.00
ESP32 (v1 DevKit)	16	Half	240	Yes	25	15	$5.99
ESP8266	4	No	80	No	17	0	$7.69
Raspberry Pi 4	2	No	1500	Yes	40	0	$75.00

Figure 25: Micro Controller Decision Matrix

The ESP32 has been selected as the micro controller for motor control. It has clock speed of 240Mhz and two cores to allow for parallel processing which results in adequate processing speed for the desired application. Almost all pins on the ESP32 have ADC capability to allow for reading of multiple sensors. It also has CAN 2.0 capability with the use of an off-board transceiver for easy CAN implementation.

7.1.2 Motor Drivers

The motor drivers we used are the DM542Ts. The DM542Ts have 8 dip switches that are used for settings. The first three set the output current according to the table below.

Our stepper motors have a max current pull of 4.2A and the DM542Ts can provide up to 4.5A.

Peak Current	RMS Current	SW1	SW2	SW3
1.00A	0.71A	ON	ON	ON
1.46A	1.04A	OFF	ON	ON
1.91A	1.36A	ON	OFF	ON
2.37A	1.69A	OFF	OFF	ON
2.84A	2.03A	ON	ON	OFF
3.31A	2.36A	OFF	ON	OFF
3.76A	2.69A	ON	OFF	OFF
4.50A	3.20A	OFF	OFF	OFF

Figure 26: : DM542T settings pt.1

To achieve this setting, all three switches are set to the OFF position.

The fourth switch is used to determine the idle current. If the switch is on, the current will be full at idle and if off, the current will be half. This is to determine how critical the idle position is. If the current is full, it will be extremely hard to manually turn the motor. With half current, it is

still hard to move but it is much easier. For our application, we decided to use half current to reduce the heat output of the motors when running all day. (Switch 4 OFF)

The last four switches determine the micro-step of the drivers according to the table below. Micro-stepping is where one pulse is converted into fractions of a pulse. This is useful for increasing the resolution of the motor.

Microstep	Steps/rev.(for 1.8°motor)	SW5	SW6	SW7	SW8
1	200	ON	ON	ON	ON
2	400	OFF	ON	ON	ON
4	800	ON	OFF	ON	ON
8	1600	OFF	OFF	ON	ON
16	3200	ON	ON	OFF	ON
32	6400	OFF	ON	OFF	ON
64	12800	ON	OFF	OFF	ON
128	25600	OFF	OFF	OFF	ON
5	1000	ON	ON	ON	OFF
10	2000	OFF	ON	ON	OFF
20	4000	ON	OFF	ON	OFF
25	5000	OFF	OFF	ON	OFF
40	8000	ON	ON	OFF	OFF
50	10000	OFF	ON	OFF	OFF
100	20000	ON	OFF	OFF	OFF
125	25000	OFF	OFF	OFF	OFF

Figure 27: DM542T settings pt.2

For our application, we wanted the highest resolution without losing max speed. Our stepper library has a limit of how fast pulses can be sent and therefore there is a resolution that will start to reduce our speed. In testing, our max speed without a load was determined to be 89 revolutions per second without micro-stepping. To determine the best resolution, the steps per rev were increased on both the driver and in code and the speed of 89rps was tested. It was determined that a micro-step above 16 started to reduce the top speed. Therefore, a micro-step of 16 was selected using switch 5,6 and 8 in the ON position and 7 in the OFF position.

7.1.3 PCB

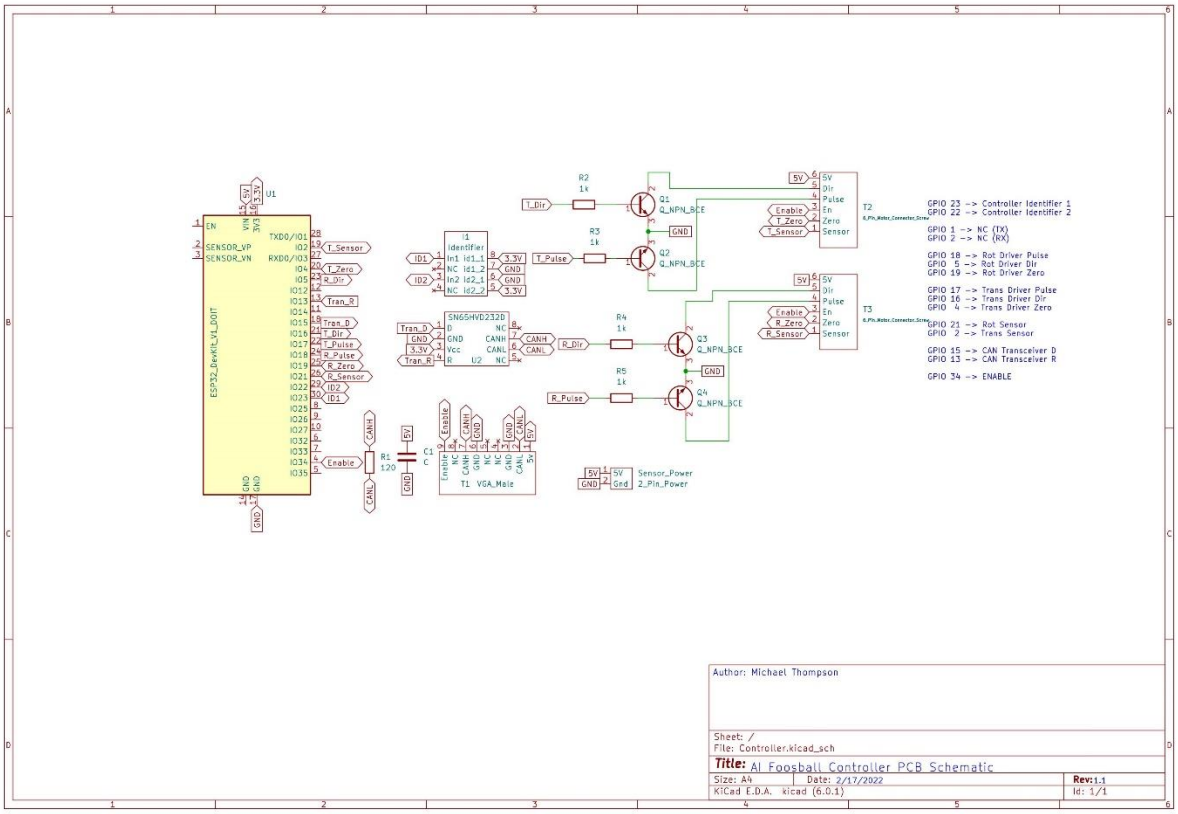


Figure 28: PCB Schematic

There are two big reasons a PCB is needed and multiple reasons that one would be beneficial. First, the chip needed for CAN 2.0 with an ESP32 is a SOIC package. SOIC is surface mount and not though hole, meaning that a bread board would not be a viable alternative. The capacitance of a bread board also results in unreliable digital communication at high speeds. Secondly, the control pins for the drivers must be an open collector signal. This means that the signal needs to pull the pins to ground for a high signal. Most microcontrollers do not have this functionality built in for "high" current applications and therefore it needs to be added using NPN BJT transistors.29

For convenience and ease of use, the PCB has multiple ports and terminals. A terminal for DB-9 allows for easy plug and play of the CAN bus, power, and enable line. This consolidates five individual wires into a single DB-9 cable. Screw terminals for the motor driver connections will be used to allow easy and reliable wire connections. Screw terminals for sensor power will make powering of board sensors (such as the dip switch and break beam sensor) trivial. Two ID pins can be soldered to high or low to allow for easy self-identification in software and a 47uF capacitor is added to filter out noise from the power supply that might affect the micro controller.

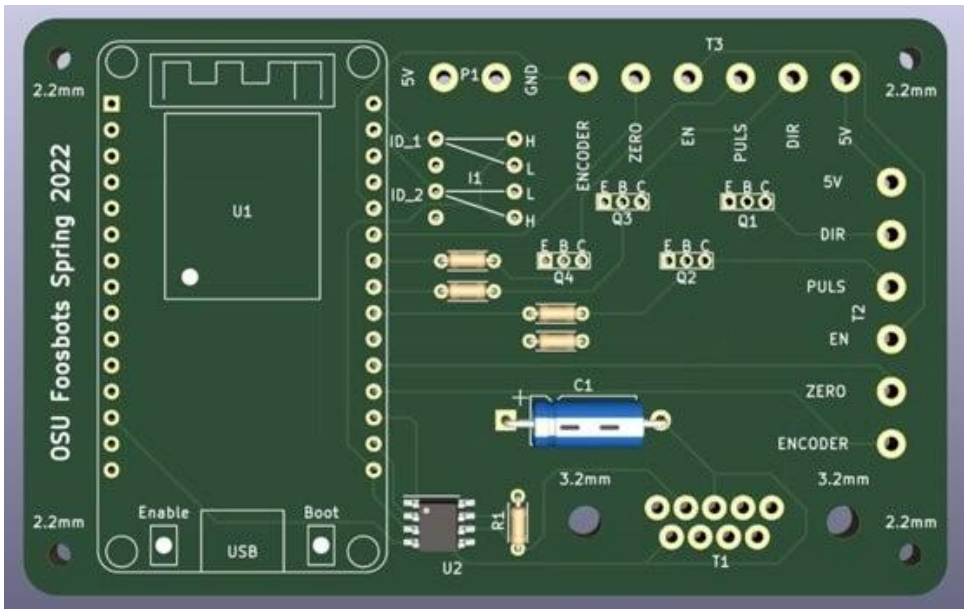


Figure 27: PCB Layout

There were a couple design flaws that were discovered during implementation.

The first flaw was the encoder pin. Most modern encoders do not use a pulse connection, instead they use a communication protocol such as I2C or SPI. Therefore, the single pin allocated to the encoder is not enough.

The second flaw was grounding. All grounds on the board are connected. Initially this seemed to be a good idea, however, that means all noise affects all aspects of the system. Instead, the ground into the board should be directly connected to the ESP32 and then each function that uses ground can use a different ground pin from the ESP32. This uses the ESP32 as a buffer. Additionally, there are not enough ground connections to be used with off board devices.

Thirdly, though minor, the translational encoder pin is attached to GPIO 2. This pin is also the blue "all good LED." To use the LED effectively, GPIO 2 should be left as a non-connection.

As the parts were being installed on the PCB, every connection was tested using a continuity test to determine if any shorts existed. This was also done while wiring the entire system together.

7.2 Controls

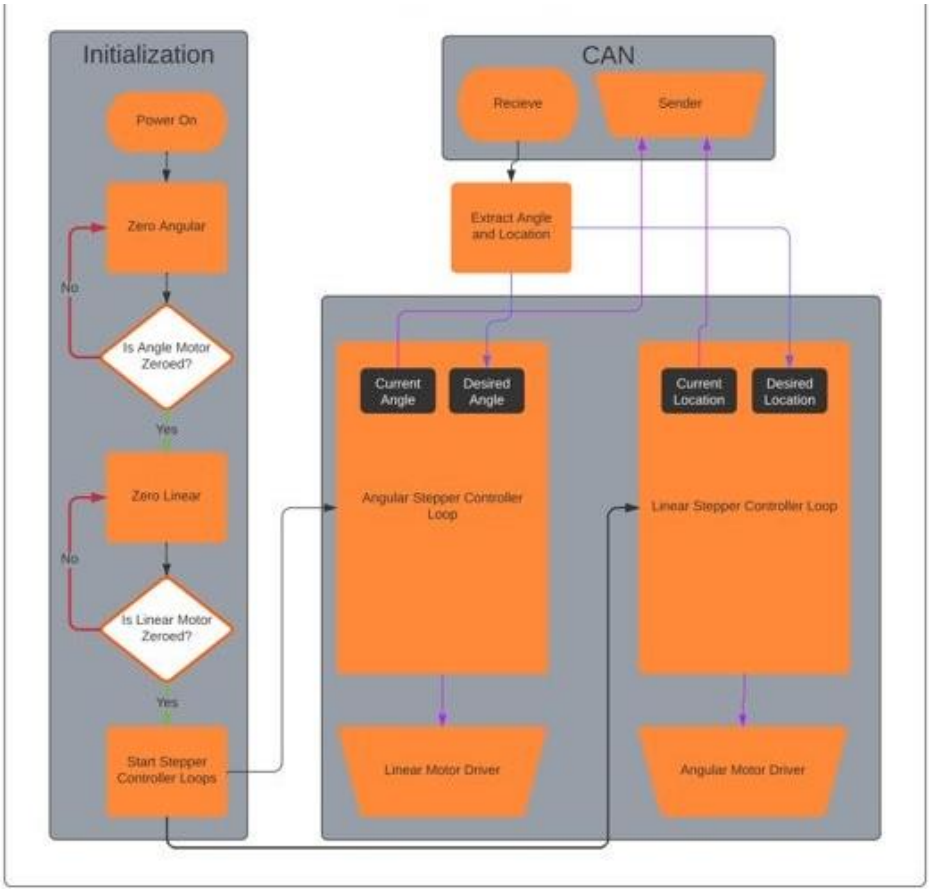


Figure 28: Controller Flow Chart

The base of the controls is the initialization and a feedback control loop. In the initialization, the pins must be assigned, and the motors must zero themselves so that they can accurately move and rotate to desired positions.

To determine the zero position of the rotation, a cam system is added to the gear that is directly attached to the motor. This gear and the gear on the rod have a gear ratio of one. Therefore, when the motor gear is in a specific position, so will the rod. This cam system presses a limit switch when oriented correctly. More details about the cam can be found in the mechanical system section.

To determine the zero position of the linear movement, a limit switch is placed at the end of the track such that it will be pressed when the rod is fully pulled out (i.e. zero position).

An issue arose when running the limit switch wires next to the stepper motor wires. The high frequency and relatively high current of the stepper motors induced a current in the limit switch wires resulting in the ESP reading that the button was pressed when it was not. To solve this

problem, smoothing/debounce was added to the pin. This resulted in the button needing to be pressed for 10ms before the ESP recognizes the button press.

Once the rods are zeroed, the control loop is entered. In the control loop, the current position and angle are compared with the desired position and angle and then pulses are sent to the motors to make the desired and current the same. The stepper motor library "ESP-FlexyStepper" determines current location and current speed by keeping track of how many, and how fast pulses are being sent. An acceleration and max speed can be set, to guarantee smooth movements. With this library, the use of a PID becomes unnecessary. The control loop is shown below.

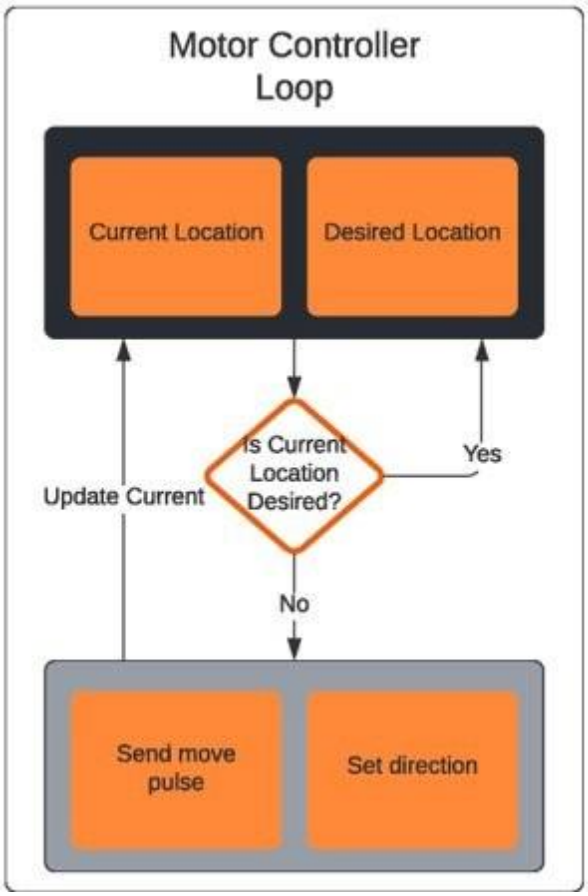


Figure 29: Motion Controller Loop Diagram

Due to the lack of encoders, the actual position of the rods will start to deviate further and further from the assumed position. We accommodate this issue by zeroing the rods as often as possible. The correct way to solve this problem is to add encoders.

7.3 Communication

"Closed Area Network" or CAN, is a hardware level communication protocol, originally designed for communication between many embedded micro controllers in the automotive industry. CAN allows for multiple devices in parallel to communicate to each other over two wires at a max speed of 1Mbps. This allows for cross communication between multiple devices, without a need for a routing device and without the need for excess wires.

CAN functions on the idea that each device talks at the same time unless a more important device talks over it. To achieve this all devices, send at the same time but if a device tries to send a 1 while another device is sending a 0, the device sending 1 stops sending and instead listens. This means that an ID of `0b00` has a higher priority than an ID of `0b10`. All the IDs have been chosen following this priority hierarchy.

Messages from the Brain will always have the highest priority so the first two bits of every Brain message ID will always be 00. The most important message the Brain can send is the stop message. Because stop commands are the most important, a stop command will be represented by 00 as the third and fourth bit. The following four bits indicate which controller the message is from or going to.

The ID will only utilize the final 8-bits of the ID's 11-bits. The message data length is always 8 bytes. Big-endian is used for all data, meaning the bits on the left will be sent or received before the bits on the right.

Messages from the Brain to the motor controllers are the most important as they result in actual movement. The next most important are the status and position messages from the controllers. The least important messages are the goal messages because the controller itself keeps track of the goal count and just sends the current count.

7.3.1 Identification

Bits	0-1	2-3	4	5	6	7
0	(00) message from AI: (01) Zero command	(00) stop\stopped message: (01) NON stop\stopped message	NOT goal rod	NOT 2 rod	NOT 5 rod	NOT 3 rod
1	(11) message from player poles: (10) message from controllers	(11) update goal counter: (10) reset goal counter	goal rod	2 rod	5 rod	3 rod

7.3.2 ID Examples

ID	Description
000XXXX1	Message from AI to controller 3 rod 1

000XXX1X	Message from AI to controller 5 rod 2
000XX1XX	Message from AI to controller 2 rod 3
000X1XXX	Message from AI to controller goal rod 4
010XXXX1	Zero Message to controller 3 rod 1
010XXX1X	Zero Message to controller 5 rod 2
010XX1XX	Zero Message to controller 2 rod 3
010X1XXX	Zero Message to controller goal rod 4
100XXXX1	Message from controller 3 rod to AI
100XXX1X	Message from controller 5 rod to AI
100XX1XX	Message from controller 2 rod to AI
100X1XXX	Message from controller goal rod 4 to AI
110XXXX1	Message from player 3 rod 1 to AI
110XXX1X	Message from player 5 rod 2 to AI
110XX1XX	Message from player 2 rod 3 to AI
110X1XXX	Message from player goal rod 4 to AI
1011XXXX	Message from goal controller to AI to update goal count
0010XXXX	Message from AI to goal controller to reset goal count

7.3.3 Data Definitions

Bytes	0-3	4-8
ID bit 2:3 = 01	Displacement Data	Angular Data
ID bit 2:3 = 11	Player goal count	AI goal count

7.3.4 Message Examples

ID	Displacement Data Length	Rotational Data Length	Description
0X00XXXX	X	X	Stop command from AI
1000XXXX	X	X	Stopped message from controller
1100XXXX	X	X	Stopped message from player
0001XXXX	4 Bytes	4 Bytes	Move command from AI
1001XXXX	4 Bytes	4 Bytes	Location message from controller
1101XXXX	4 Bytes	4 Bytes	Location message from player

7.4 Power

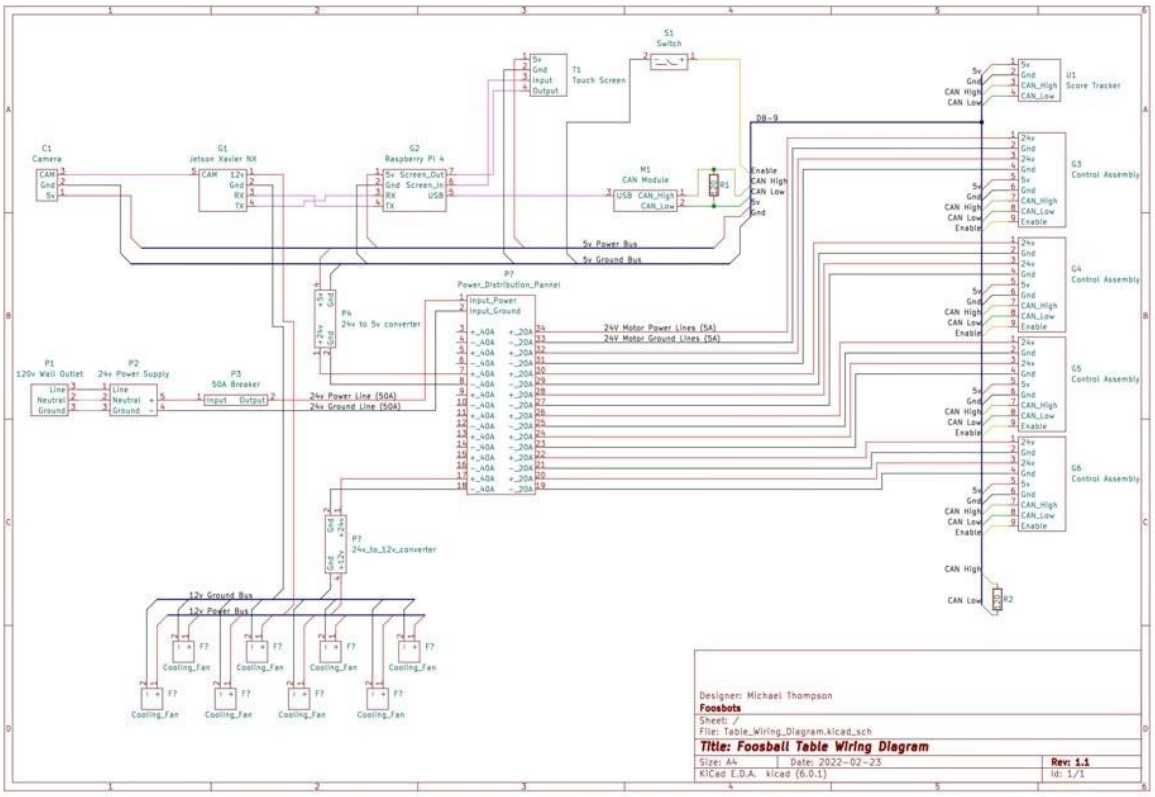


Figure 30: Wiring Diagram

There are three different voltages that are needed in the foosball system. The motors run on 24V, the micro controllers on 5V and the Jetson Xavier and cooling fans on 12V. To satisfy these needs, a 24V, 1200W power supply with two step-down voltage converters (5V and 12V) is used. The total power consumption of the system is about 964W, which is 80.33% of the 1200Ws. With that, we are at the 80% rule and have deemed this power supply sufficient.

To isolate the motors from themselves and the rest of the system a Vex Robotics power distribution panel is utilized. This panel has eight 20/30A channels and eight 40A channels. The motors are 4.20A motors so the eight 20A channels will be sufficient for this purpose. Two 40A channels can be used to power the 5V and 12V converters. The 5V and 12V lines were then connected to individual buses for ease of connection.

The 5V bus serviced the Raspberry Pi 4 and the ESP32s. Each ESP was connected in parallel using a 5V and ground cable that were directly connected to the DB-9 breakouts. One issue that was discovered was the fact that the voltage dropped between each ESP. This was so pronounced that the last ESP had an input voltage of 3.6 volts. This resulted in the last ESP not always being able to control its driver, resulting in no motor movement. To solve this issue, a power loop was introduced around the ESP lines. A power and ground cable on one end and a power cable on the other end. Originally a ground cable was on both ends, but this created a ground loop that messed with the Raspberry Pi's operation.

The 12V bus is actually two buses located on either side of the table. This allows the fans on either side of the table to connect to a close bus and reduces the number of wires that need to be ran. The jetson is powered by the bus that is closest to it.

Device	Power	Count	Total Power
Motor	100.8W	8	806.4W
Driver	3W	8	24W
Touch Screen	5W	1	5W
Pi 4	17.85W	1	17.85W
Jetson	89.3W	1	89.3W
Fans	1.8W	10	18W
PCB	0.715W	5	3.575W
Camera	0.12W	1	0.12W
USB2CAN	0.432W	1	0.432W
Total			964.677W
Margine			80.39%

Figure 31: Power Dispersion

7.5 Goal Tracking

For goal tracking, we use break beam sensors. To accomplish this, we used two sets of sensors to track each side of the field respectively which are mounted inside of the table. The sensors are connected to a single ESP32 microcontroller to send a high signal when a ball is detected. This, then, sends a signal to the CAN bus to properly change the goal count of the game. Each break beam connection was debounced to eliminate the risk of multiple goals being sensed when only one ball was sensed.

8 Software Design

8.1 Overview

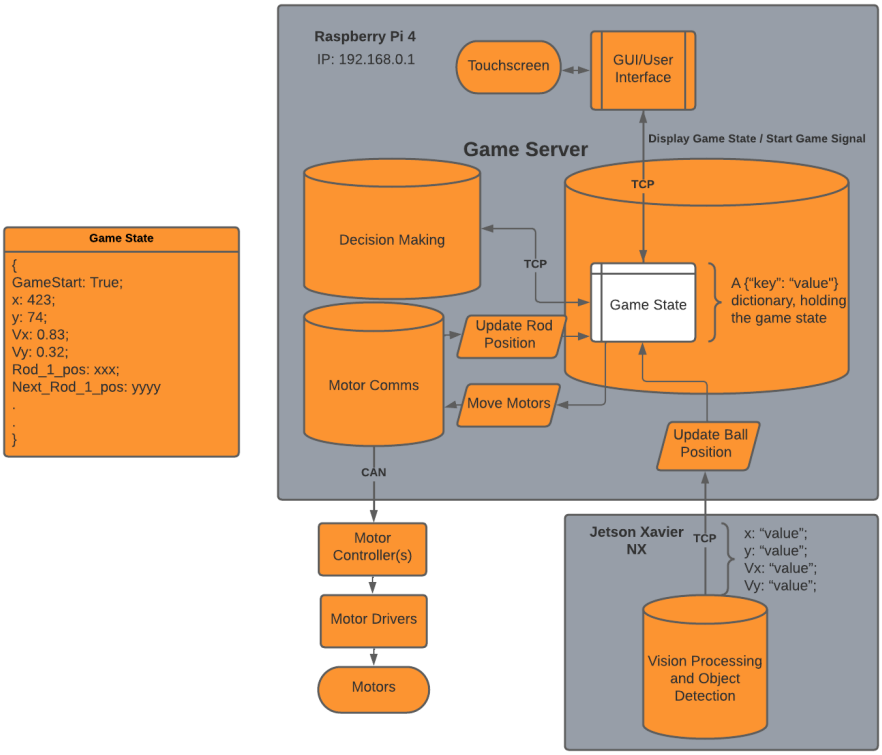


Figure 32: Flow chart of software module interactions

Our goal is to build a modular and extendable system. We achieve this by ensuring a single source of truth by ensuring the state of our system is detached from any dependent systems, very similar to the idea of a state redux. We have selected various communication protocols that will used across our of system (as well as many other systems), while ensuring the specific method chosen is both efficient and reliable.

This design uses two processors, a Jetson Xavier NX (referred to as the Jetson) and a raspberry pi 4 (referred to as the pi). Vision processing is handled on the Jetson while the primary game control and motor communication is handled by the pi. This ensures that each process is able to be completed in a timely manner as well as allowing for processors to be switched at the developer's convenience.

Images are fed into the Jetson, where the necessary filters and calculations are ran, at which point an X, Y coordinate and velocity vector are sent to the pi, updating the ball position. On the pi, various applications are polling the game state, such as the Decision matrix and motor communications that use the data from the game state to make decisions and/or update other portions of the game state.

8.1.1 Inter Process Communication

There are two distinct problems we must approach regarding Inter-Process Communication:How other devices (i.e. the Jetson) communicate data to the pi and how do processes on the pi communicate with each other?

To solve both problems we made use of TCP Sockets. From the figure above, each module communicates with a central server that handles data transfer across modules. Our original intention was to use UART to communicate data from one device to another, but due to the frustrating complexity of the Jetson's custom Ubuntu OS and seemingly lack of documentation we scraped UART in favor of TCP Sockets. We wanted to be able to connect the Jetson and Pi using an Ethernet cable, forming a local network between the two devices. To achieve this, we setup a DHCP[1] Server on the Pi that would assign an IP address to whatever device connects to the Pi's Ethernet port.

This means that we could standardize the way *all* processes communicate through the entire application. The DHCP server should never have to be edited (unless more than 99 clients are needing to be serviced, in which case a proper router is highly encouraged), and as the famous phrase goes "it *should* work out of the box".

8.1.1.1 DHCP Server

For the sake of reference, we used this[2] tutorial to setup the DHCP server, it's a bit more in-depth. In general, two steps are needed to setup a simple DHCP server: assign a static IP address on the interface the corresponds to the Server, then bind that IP address to a DHCP server running on that device. In the process of setting up the DHCP server the gateway and subnet are defined as well. The purpose is to reconfigure an interface to accept DHCP requests rather than sending them (which is what it does normally).

Specifically, we used the *eth0* (Ethernet) interface and assigned *192.168.0.1* as the address for that interface. This means that, when another device is connected to the pi (via ethernet), the device would be able to ping *192.168.0.1* showing a connect between the two devices. I should also note that devices connected to the pi will receive an IP address, for instance the Jetson receives an address of *192.168.0.100* meaning the pi could ping that address as well.

One downside however, is that the pi cannot connect to the internet via its ethernet port however it's wireless interface works perfectly fine, but Endeavor's Access Points don't allow the pi to connect to it without a *wpa_supplicant* work-around (which we did not have access to). To get around this we used our phone's mobile hotspot for quick connection the wider internet.

To use the ethernet port for the internet the DHCP server must first be turned off and the static IP of the *eth0* interface removed. First turn off the DHCP server by running:

```
sudo systemctl disable isc-dhcp-server.service
```

[1] https://docs.microsoft.com/en-us/windows-server/troubleshoot/dynamic-host-configuration-protocol-basics
[2] https://www.technicallywizardry.com/building-your-own-router-raspberry-pi/

Then navigate (in a terminal) to: /etc/network/interface.d (for the uninitiated run `cd /etc/network/interfaces.d`). Open the *eth0* file (sudo nano *eth0*) and flip the commented lines. This file should look something like this:

After edit:

```
# for static IP and DHCP
#allow hotplug eth0
#iface eth0 inet static
#   address 192.168.0.1
#   netmask 255.255.255.0
#   gateway 192.168.0.1

# for general eth0 use
auto eth0
iface eth0 inet manual
```

Then reboot the Pi: `sudo reboot`

This will allow the *eth0* interface to be used normally. To turn the DHCP server back on do the following:

```
sudo systemctl enable isc-dhcp-server.service
```

Return to /etc/network/interface.d. Open the *eth0* file (sudo nano *eth0*) and flip the commented lines. This file should look something like this:

```
# for static IP and DHCP
allow hotplug eth0
iface eth0 inet static
    address 192.168.0.1
    netmask 255.255.255.0
    gateway 192.168.0.1

# for general eth0 use
#auto eth0
#iface eth0 inet manual
```

Then reboot the Pi:

```
sudo reboot
```

One thing to note is that this DHCP server is capable of serving more than one other device. If the pi's ethernet port is connected to a switch, which can allow other devices to connect to the network.

8.1.2 Running Python Scripts at Boot

In order to make the table function when it's powered on, all relevant code needs to be executed at startup. While there are a number of different ways this can be done (one popular method being *crontab*), we made services that would run our python script(s) at boot up.

To setup your script at startup, create a .service file in `/etc/systemd/system/"script name".service`. In this script write:

```
[Unit]
Description=My Script

[Service]
ExecStart=/usr/bin/python3 /path/script.py

[Install]
WantedBy=multi-user.target
```

Notice that the first argument of "ExecStart" points to the location of the compiler and the second argument points to the script. We placed our scripts in the /bin folder using (from the project directory):

`cp ~/"project directory path"/"your script" /bin/"your script"`

This is done to allow the script to run with root privileges at boot, circumventing any potential permission problems.

There are a few commands that are useful to know when working with services:

- `sudo systemctl enable/disable "service name".service`
  - enables/disables the script to run at startup
  - After creating the startup service *enable* needs to ran.
  - If using *disable* the script will not stop until a reboot
- `sudo systemctl start/stop "service name".service`
  - starts/stops the service from running (mainly used for debugging)
  - has no impact on the enable/disable command, meaning the script will start on boot unless *disable* is used.
- `sudo systemctl status "service name".service`
  - shows the current status of the service
  - will display if the script is running or not, and will display error/crash messages.
  - *Very* useful for debugging.

8.2 Vision and Ball Tracking

For vision, we made use of OpenCV to build a blob/color detection algorithm that would reliably calculate the location and velocity vector of the ball. This information is enough for the pi's decision matrix to predict where the ball will be a move the players accordingly.

A challenge with real-time processing of images, and any real-time processing in general, is leveraging the rate of processing with the required amount of data to be useful. The following images outline the relationship between processing time and frame rate, as well as distance delta between frames at various ball speeds.

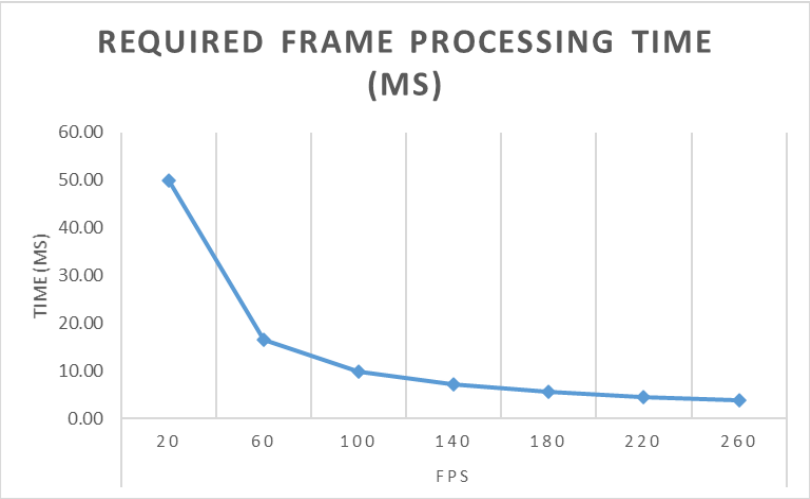


Figure 300: required processing time at various frame rates.

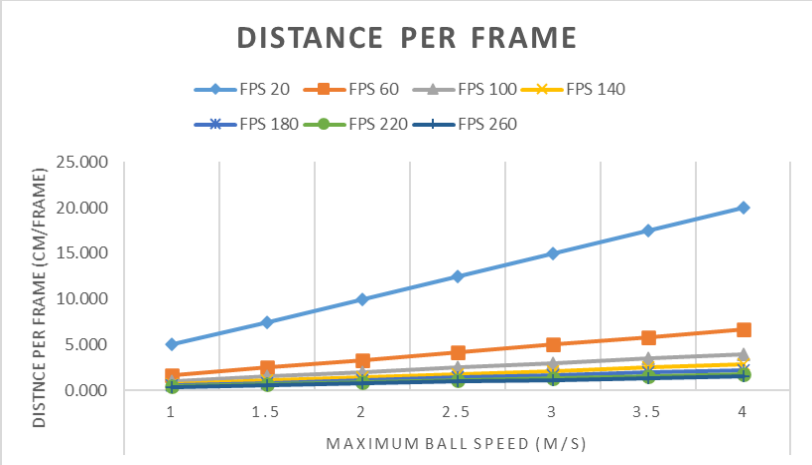


Figure 21: distance between one frame and the next at different ball speeds.

In the early stages of approaching this problem we outlined two parameters that would define the constraints of our vision processing. The graphs show the required processing times, dependent on frame rate, and the distance covered between two frames, at various ball speeds and frames per second.

As stated in our deliverables, our goal is to keep the processing time under 10ms, this meaning processing 100 frames per second. We will still make use the 260fps camera as, the program grabs the most recent image available, so if an image is processed faster than 10ms the decision matrix will be able to make use of the additional frames, and thus improve resolution.

The decision matrix will make use of the velocity vector to make rudimentary predictions regarding where the ball will be and react accordingly. This means that if the ball is moving at 4m/s (the fastest calculated speed of the ball) the decision matrix will be aware of this speed and respond to the ball's future location.

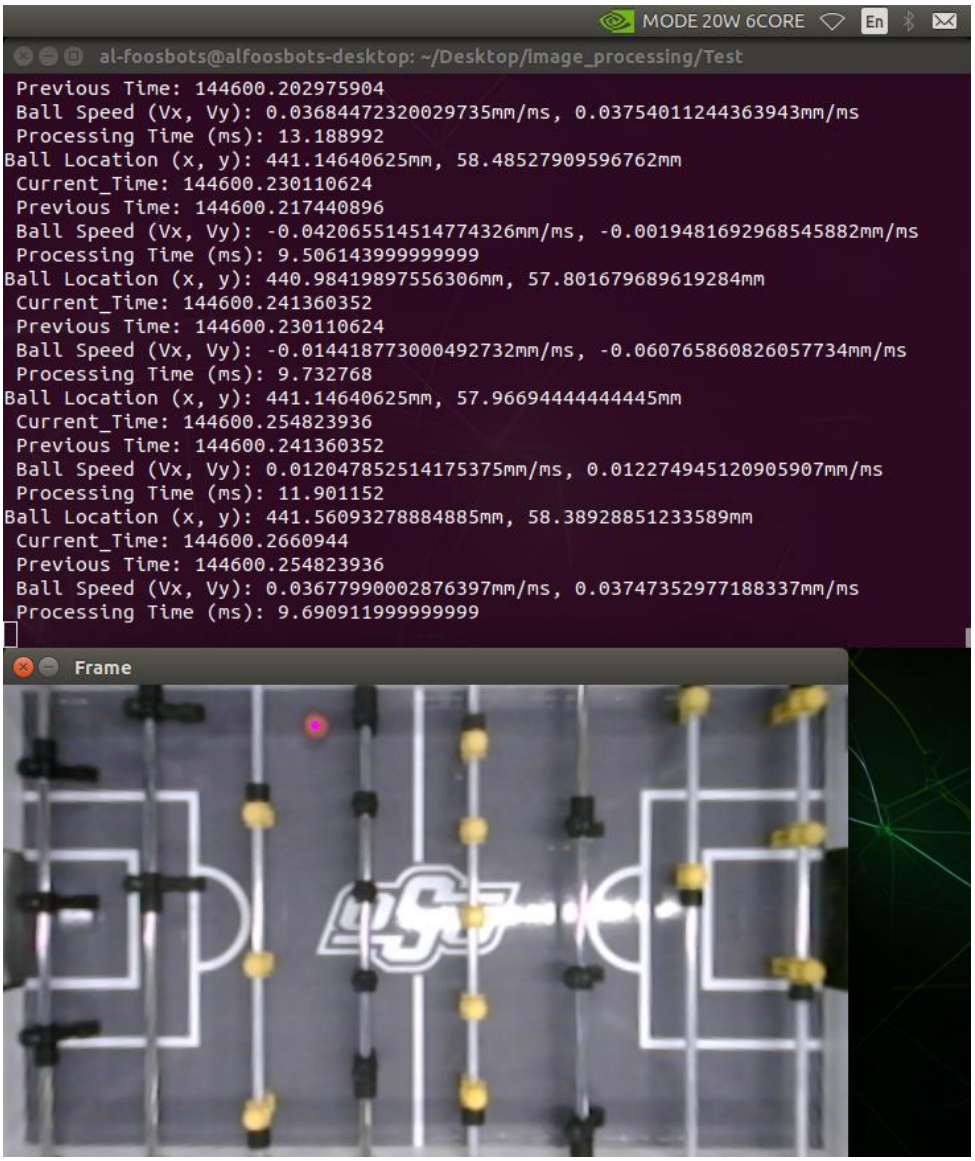


Figure 22: console output of vision algorithm

From the image above, it is clearly shown that we have achieved our 10ms processing constraint. This test was done running all necessary processes that are needed to process each image.

Another challenge tackled is reliability of the vision algorithm in various lighting conditions. To address this, there are two specific things we have done to address this:

1. We're making use of the HSV (Hue, Saturation, Value) spectrum rather than the RGB spectrum. The HSV spectrum is known to be more reliable in various lighting conditions while still allowing for accurate color detection.
2. Making use of multiple thresholds masks the filter out noise, false positives, and overall increase the reliability of the data.

The images below show the algorithm in action, detecting the ball at various locations around the table. We found that, even though the table had a large glare, the algorithm had little difficulty in detecting the ball at that location.

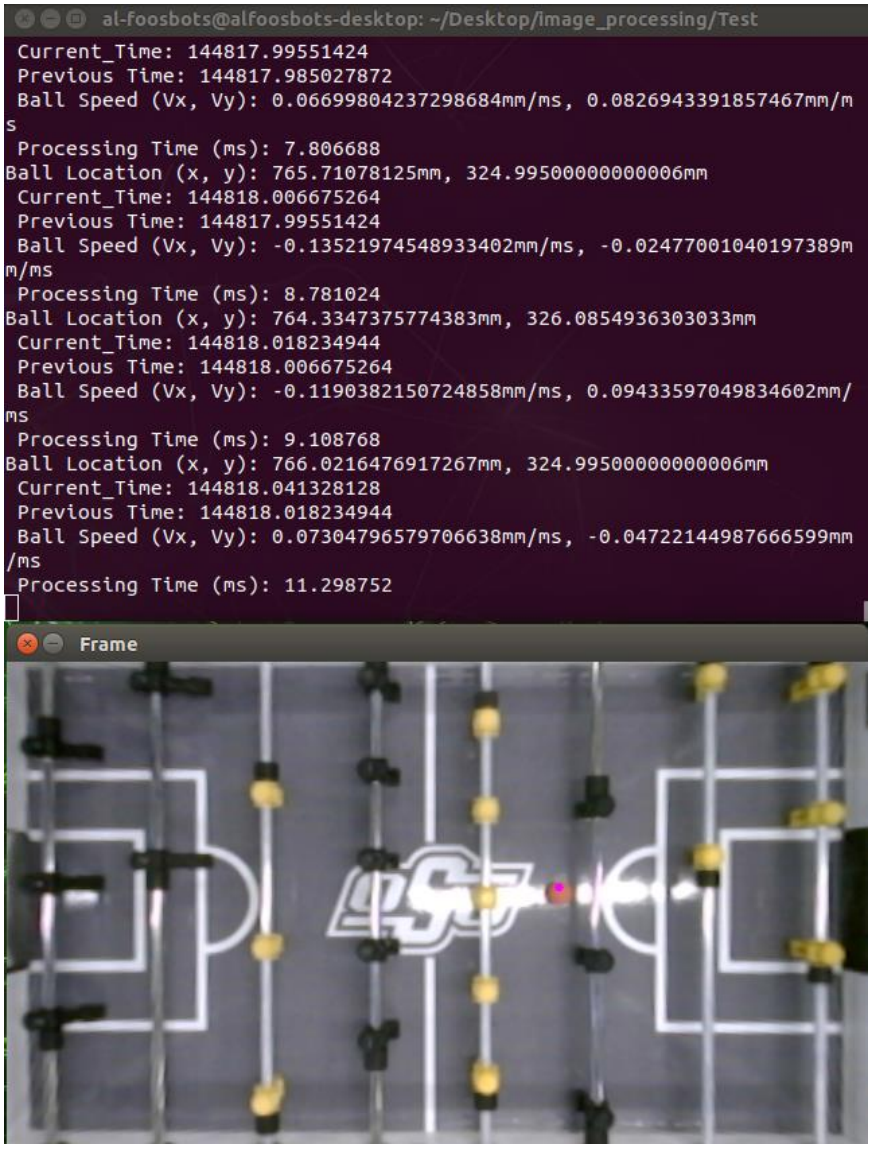


Figure 31: Vision detects the ball with table glare.

8.3 Graphical User Interface

To allow the user to interact with the table we used a touch screen mounted on the frame, as seen in the image below. The touch screen will, at minimum, allow the user to start/stop the game and view the current score of the game.

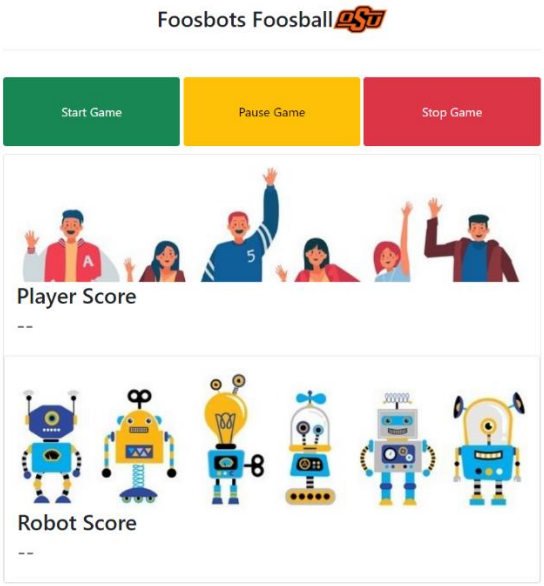


Figure 32: Main screen of the GUI

To design the application, we'll be ElectronJS and industry level cross-platform toolkit of which many popular applications have been built such as VSCode and Slack. We chose this toolkit as its highly flexible and compatible with a variety of stacks, both front-end and back-end. As a basis the application will allow the player to start/stop/pause the game and display the current game's score. Clicking on the OSU logo will bring up a debug modal that displays all the data currently contained within the server.

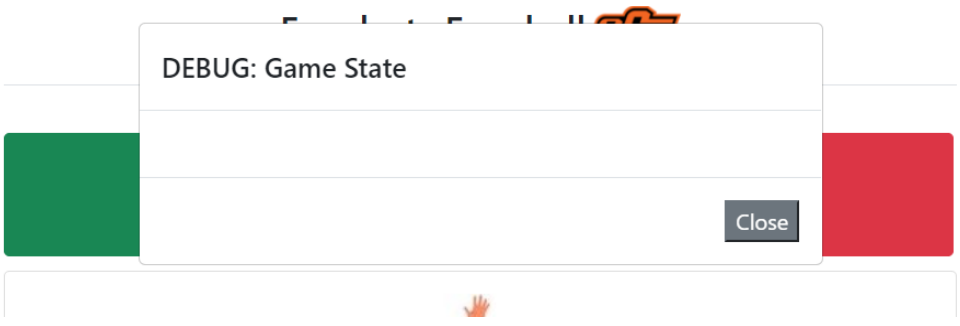


Figure 33: Debug Modal, dumps json data into the modal during runtime.

When the user presses "Start Game" they will be prompted to select a difficult, after which the game will start.

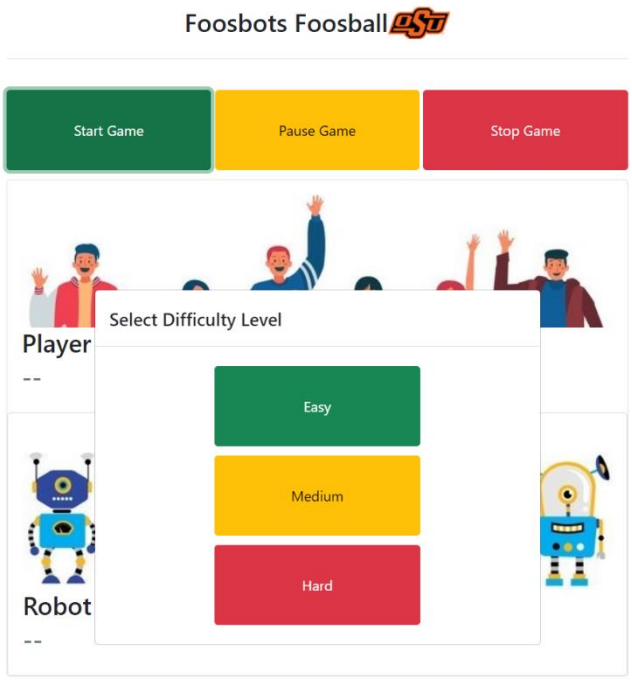


Figure 34: Difficulty modal, appears after selecting start game.

8.4 Server

The server process of the project represents the central hub for all data. Every other process either generates or requests specific pieces of information and the server is responsible for facilitating that interaction. This method of data transfer provides a significant advantage as any new process must only set up communication with the server and can ignore the operations of any other module. The server functions on the basis of socket communication. A socket is in essence a door to a computer process at a specific address. The server first creates its door and then listens for the "knock" of another process. Other processes can create sockets with the same address and "bind" to it, creating a pathway for communication. When our server receives a connection on its socket. It first checks to see if this is a new connection or one that has already been set up. In the case of a new connection, the two processes "shake hands" exchanging their names and addresses. The server then stores that information and adds it to the list of known contacts. If that process sends data again, the server will recognize the connection and move to process its data. Clients can request two types of interactions, writing a message to the server, or reading a message from the server. If the client requests a read, the server looks for data addressed to that connection and sends it to the client process. Clients writing to the server require more processing. Written messages can either be a list of data elements to send back to the client or a list of values to update those stored in the server. When a client requests the server's data, it is collected and stored with the client's address until the client requests again for it to be sent to it. After the completion of any request by a client the server returns and waits for another connection by a client. Currently the server does process requests sequentially, this avoids issues with concurrent data access while in testing still maintains fast speeds.

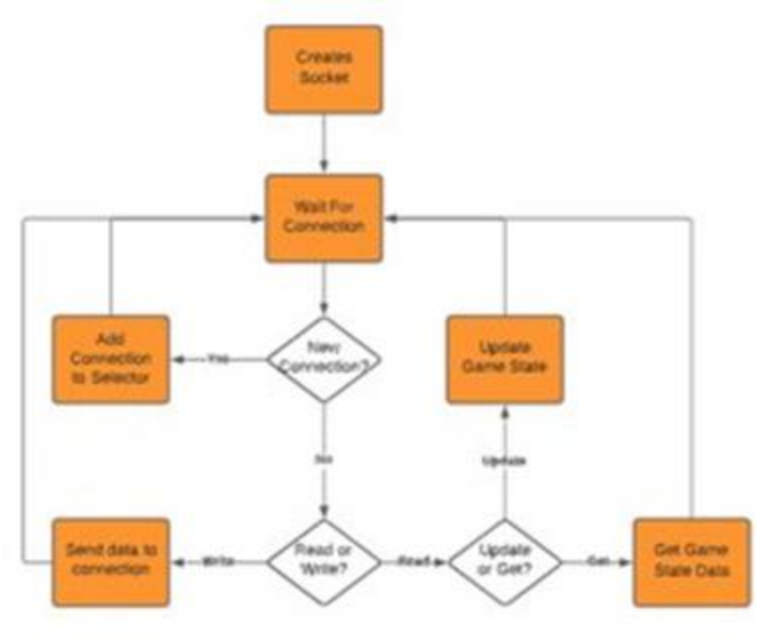


Figure 25: Flowchart of Server Operations

Figure 26: Data in the Server

The data stored in the server takes the form of a python dictionary. A dictionary is a list of key, value pairs. Requests for updating and receiving data also have this format. When data is requested, the server checks if it has a copy of the key requested and updates the value in the sent dictionary, if not the server sets the value to "not found". Updating data works similarly. If the server has a copy of the sent key, it updates its stored value. Otherwise, it adds the key to its dictionary. This setup allows for any information to be added or requested from the server with no additional changes to its function. The only consideration by a user is ensuring that any data needed is also being sent to the server. Formatting messages to the server has also been handled by the creation of a custom Message class. The class allows for the creation of a dictionary for data and handles all serialization needed to send through a socket. No additional work is required for data transfer from processes written in python, new clients written in other languages would have to manually format the data into a JSON object.

8.5 Finite State Machine

One of the more significant clients of the server will be the brain module, responsible for taking in the information gathered by other sensors and making decisions on motor commands that are sent back down to the motor controllers. While many versions of this brain module could be created, our implementation takes the form of a finite state machine. Warning, it is very easy for moving the rods to create dangerous scenarios for the table. For example, if the ball gets caught between a player and the ground or wall while swinging the rods the motors will continue to torque into the stuck position putting our mechanism at risk of breaking or having gears skip. Skipped gears require the table be opened and reset manually, there is no method to reset the system from software. A method to catch this issue has not been implemented in hardware either so these circumstances must be avoided by the software. This concern influenced much of the final implementation. Four state machines will be run, one for each rod on the table. Each state machine

will have seven possible states. The Stop state is triggered by an external signal and halts all output from the state machine. The signal also stops motor actions directly, but additional security is beneficial overall. Leaving the Stop state puts all rods into block position. The Idle state is entered when the ball is significantly far away, more than half the table. Rods in this state will return to a default position centered on the board and avoid attempting to react to changes irrelevant to its position and prevent rods from placing themselves too far out of position. The Open state is entered when the ball is behind a rod. The players are placed in a position parallel with the table allowing shots from rods on the same team to pass underneath. Players will still track the ball's position and attempt to line themselves up with good shots as the ball approaches. Block occurs when the ball is in front of the rod. Players are set vertically and maneuvered to place a player between the ball's current position and the goal. The Prep state can be entered from either Open or Block when the ball approaches the rod under a certain speed. The players are angled in preparation and lined up towards a goal in preparation to take a shot. When the ball reaches the rod the Shoot state is entered, swinging the players quickly forward before returning to the Block state. The final state allows the rod to Recover when the ball is in a dangerous position. Such as when the ball is behind the goal rod against the wall or underneath a rod. This state moves a player to the inner side of the ball and attempts to hit it from the side in order to move the ball into a safer position. This first implementation of a foosball robot is capable of playing foosball and has been able to beat beginner level human opponents. There is significant room for further improvements that will be detailed in a later segment of this report.

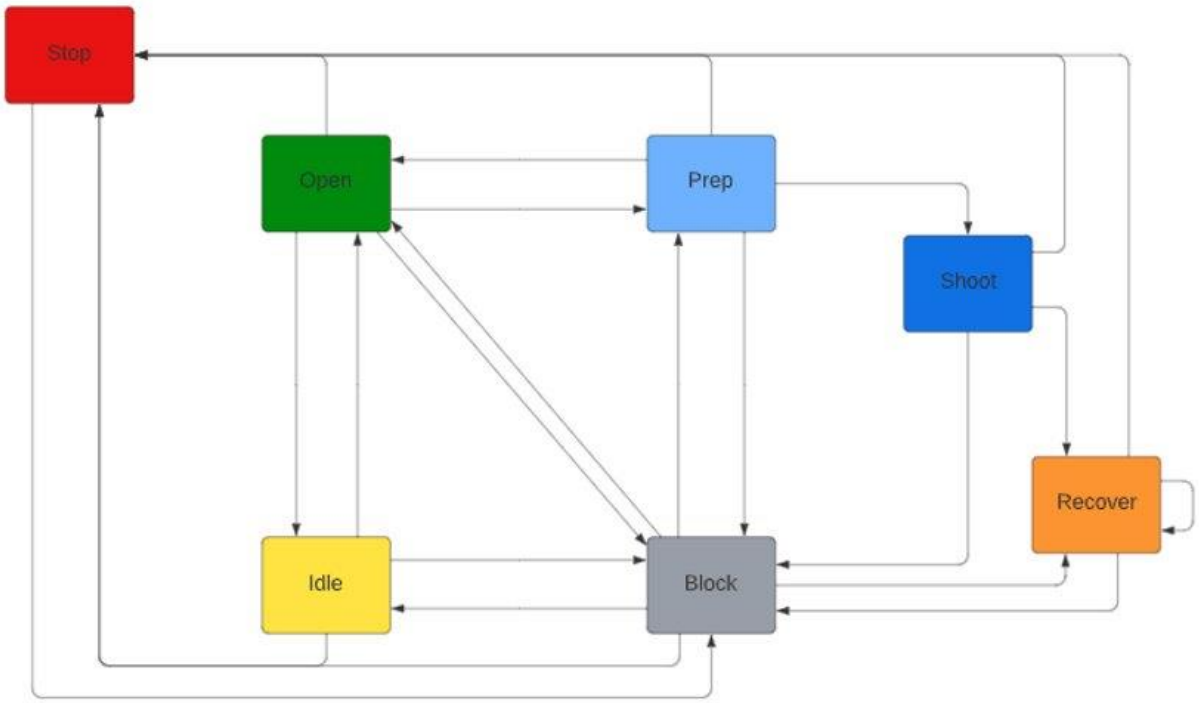


Figure 35: Finite State Machine of Individual Rod

9 Knowledge Acquisition and Engineering Analysis

9.1 Software Knowledge Acquisition

For all of software, google is king. If we had questions, we googled it. Documentation is useful but frequently far too detailed for what we are looking for, thus we looked up people who had previous experience in the problem we were facing.

Putting google aside, as a team (and individual) we carefully dissected the problem(s) we faced, decomposing it into smaller and smaller components. Each portion of the program was carefully analyzed to determine what *that* code needed.

Below is an image showing how we broke down the *behavior* of a section of code. This allowed us to have a clear, 5000 foot overview, of what the code needed to do. Our approach to writing code centered on the idea of writing modular functions and this helped us work together as a group especially.

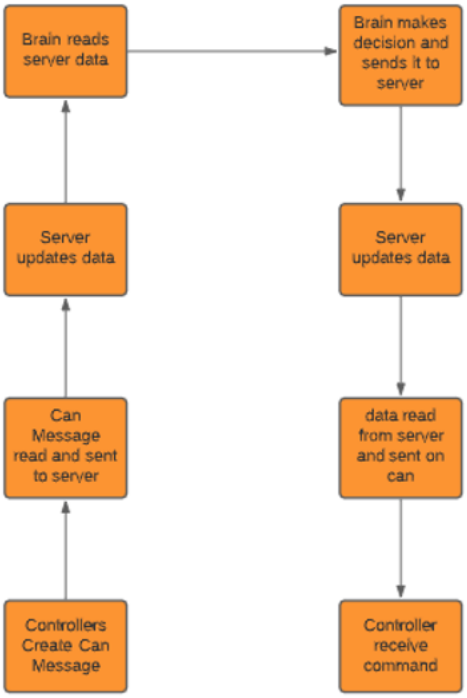


Figure 36:Data Flow Chart

9.2 Electrical Hardware

Our main hardware focus for the electrical side is our motors. To find their max speed, we used a potentiometer to increase the speed of the motor until the motor hits its limit and stops. We further tested motors by making sure they move continuously at their expected speed.

When creating the PCBs, we had to make sure to use the proper trace width with the voltages we are using. We used the following equations to find the needed width according to our parameters. We found that the width needed is smaller than the default width so we opted for the default.

$$Area = \left(\frac{Current}{k \cdot (Temp_{rise})^{.44}}\right)^{\frac{1}{.725}} \quad (2)$$

$$Width = \frac{Area}{Thickness \cdot 1.378} \quad (3)$$

$$Watts = volts \cdot amps \quad (4)$$

Name	Voltage	Resistance	Peak Current	Min Trace Width
Driver Pulse Input	5v	270	18.5mA	0.0032mm
Driver Direction Input	5v	270	18.5mA	0.0032mm
Driver Enable Input	5v	270	18.5mA	0.0032mm
NPN base	5v	1k	5mA	0.0005mm
NPN base	5v	1k	5mA	0.0005mm
NPN base	5v	1k	5mA	0.0005mm
NPN base	5v	1k	5mA	0.0005mm
SN65HVD23			17mA	0.0028mm
ESP32			50mA	0.0125mm
Total			142.5mA	0.0532mm

The default trace width is .25mm which exceeds all minimums. Therefore, the default is used.

Figure 37: PCB Trace Calculations with Thickness: 35um, Temp Rise: 10 C, Ambient Temp: 25 C, and Trace Length: 89mm

9.3 Motion

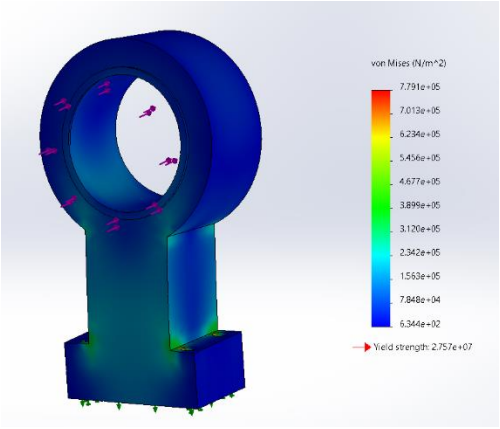


Figure 38: Arm Attachment FEA

According to the FEA (Finite Element Analysis) analysis done on the arm attachment, the shear force that will be expended on it is well below the yield strength. In conclusion the arm attachment should be able withstand the forces acting upon it.

9.3.1 Rotational Torque Calculations

For motor sizing we performed multiple tests to find the forces experienced during the average foosball match. The rotational forces were found by fixing an Arduino powered MPU6050 accelerometer to one of the foosball rods with a custom designed collar. The rotational acceleration values were plotted throughout the game.

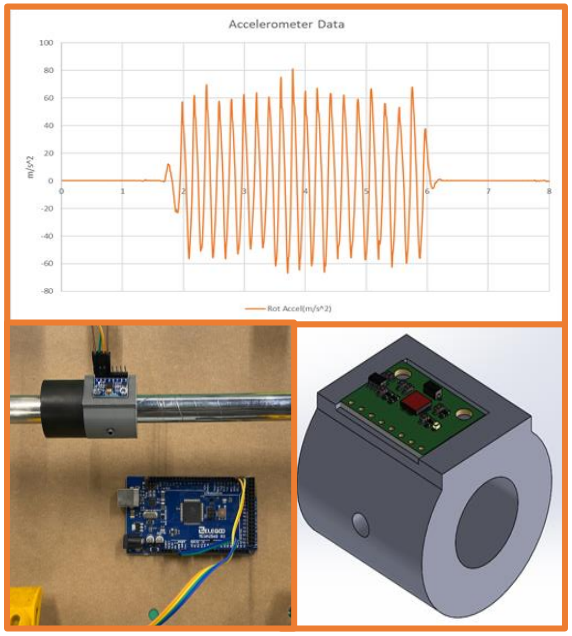


Figure 39: Accelerometer Setup and Output

Accelerometer Data		
Time (s)	Lin Accel (m/s^2)	Rot Accel (m/s^2)
3.74	6.45	-43.76
3.75	-1.19	-33.35
3.76	-7.01	-15.46
3.77	7.79	13.69
3.78	-8.62	40.1
3.79	13.8	79.94
3.8	9.19	69.86
3.81	5.26	57.05
3.82	1.07	53.4
3.83	0.18	44.06
3.84	-9.74	29.92
3.85	-2.84	16.22
3.86	0.92	-1.19
3.87	-2.16	-15.78
3.88	0.19	-31.69
3.89	4.92	-49.66

Figure 40: Accelerometer Data

Using the maximum acceleration (a) that occurred during a game in tandem with the following equations, the worst-case scenario torques that the motors would experience could be found.

Torque due to rod rotation is calculated as,

$$T = I * \alpha \tag{5}$$

Where I is the mass moment of inertia and α is the rotational acceleration.

$$I = \frac{1}{2} * m * r^2_{axis\ of\ rotation} \tag{6}$$

$$\alpha = \frac{a}{r_{Accelerometer}} \tag{7}$$

When using the following values,

$$m = 0.346\ kg$$

$$r_{axis\ of\ rotation} = 0.04\ m$$

$$a = 79.94\ m/s\text{^}2$$

$$r_{Acceleration} = 0.014\ m$$

The maximum torque the motor would experience is 1.58 N*m, which is well below the maximum holding torque our motors are rated at of 3 N*m.

9.3.2 Linear torque Calculations

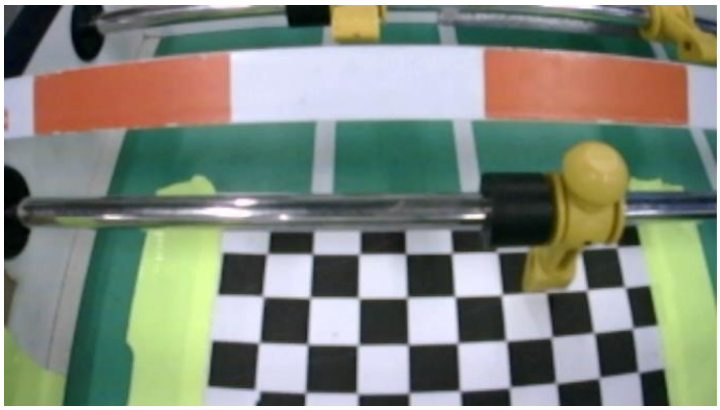


Figure 41: Clipping of video used for analysis

The linear motor torque we calculate is by the data we gathered from video analysis. We solved the velocity using frame rate in the equation below:

$$\Delta V = \frac{\Delta x \times fps}{n_{frames}} \quad (8)$$

Where Δx is the change in position of the players sliding back in forth, fps is the frame rate per second, and n_{frames} was the total number of frames of the video. Once we find velocity, we then solve the change in the time with the equation you see below.

When the following values,

Δx = 59 in

Fps = 260

n_{frames} = 179

$$\Delta t = \frac{n_{frames}}{fps} \quad (9)$$

Now that we have both velocity and time, we can solve for acceleration using the equation:

$$a = \frac{\Delta V}{\Delta t} \quad (10)$$

Where ΔV is the change in velocity and Δt is the change in time. We then solve the total mass using the equation below.

When the following values,

ΔV = 2.47 m/s

Δt = .062587 s

$$m = m_{rod} + m_{gantry} \quad (11)$$

Where m_{rod} is the mass of the rod and m_{gantry} is the mass of the gantry. Once we have mass and acceleration, we use Newton's second law to solve the total force in the equation below.

When the following values,

M_{rod} = .376 kg

M_{gantry} = 1.268 kg

$$F = m \times a \quad (12)$$

Where m is the total mass and a is the acceleration of the rod. Now that we have the force, we can solve linear motion torque using the equation below.

When the following values,

M = 1.644 kg

a = 47.65 m/s^2

$$T = F \times r \quad (13)$$

When the following values,

F = 79.3654 N

r = .0127 m

Where F is the total force of the rod and r is the radius of the rod. We get that torque is equal to 1.58 Nm. The number we got is slow because they were calculated using steel rods for the material. However, the steel rods will be replaced with aluminum rods which will in turn speed up the linear torque.

9.4 Materials

We chose to use aluminum 6061 T-6 for the rods on our AI side because it takes 1.2 Kg of the table. And through numerous calculations we find that the aluminum rods can withstand the forces applied to it. Also, aluminum is both lightweight and flexible with high corrosion resistance. We also chose ONYX material because is the cost benefit for using the material since it is offered to us by the 3-D printing lab. We saved 211.72 dollars using onyx instead of ordering 6010 aluminum gears. Finally, we chose to go we PLA material because of the accessibility offered in the 3D printing lab and it is 15 times stronger than nylon material.

9.4.1 Materials Calculations

The Lewis equation is a well-known method for calculating the maximum load stresses in a gear tooth. It represents a gear tooth as a simple cantilever beam taking the full weight at its tip. Here is the maximum load W we determined for the gear using onyx and the spline using PLA plastic using the equation shown below.

$$W = [\, S \times F \times Y \,] \div D \quad (14)$$

Where S is the Maximum bending tooth stress taken as 1/3 of the tensile strength, F is the face width of the gear, Y is the Lewis factor, and D is the diametral pitch of the gear.

Here we have the fatigue life calculation proving that the aluminum can withstand the forces applied. Using the specs sheet from McMaster-Carr we can gather that the ultimate tensile strength

is 290MPa. Using the corrected fatigue endurance limit and the material strength endurance in torsion in the equations below.

$$S_e = C_{load} \times C_{reliabilty} \times C_{surf} \times C_{size} \times C_{temp} \times S_e' \quad (15)$$

Where C_{load} is the coefficient of the load on the aluminum, $C_{reliablity}$ is the reliability coefficient of aluminum, C_{surf} is the surface coefficient of aluminum, C_{size} is coefficient of size of the aluminum rod, C_{temp} is the coefficient of the temperature of the aluminum, and S_e' is the fatigue endurance limit the aluminum can handle before shearing. Many of the variables such as C_{load}, C_{temp}, and $C_{reliablity}$ was given numbers because coefficient of load in torsion is equal to 1. The coefficient of temperature for any degrees less than 450°F is equal to 1. Then the coefficient of reliability for aluminum is a given number of .897. We then solve the size coefficient using the equation below.

$$C_{size} = 1.189 \times d^{-.097} \quad (16)$$

Where d is the diameter of aluminum in millimeters. Next, we solve the surface coefficient using the equation shown below.

When the following value,

d = 15.7 mm

$$C_{surf} = 4.51 \times S_{ut}^{-.265} \quad (17)$$

Where S_{ut} is ultimate tensile strength of aluminum 6061.Then we solve for the fatigue endurance limit using the equation below.

When the following value,

S_{ut} = 290 MPa

$$S_e' = .5 \times S_{ut} \quad (18)$$

Now that we have all the variables, we can plug it in to equation 2 to receive a corrected endurance limit of 118.84 MPa.

When the following values,

C_{load} = 1

$C_{reliablity}$ =.897

C_{surf} = 1.00378

C_{temp} = 1

C_{size} = .91028

S_e' = 145 MPa

However, to use the S-N curve graph we solved for strength of material in torsion using the following equation.

$$S_m = .9 \times S_{ut} \quad (19)$$

Finally, through the use of the S-N curve graph for aluminum 6061 we find that our fatigue strength is 94.5MPa which is equivalent to 13706.07 psi, and we also find that the number of cycles is 3500000.

9.5 Actuator Table Basic Heat Analysis

9.5.1 Critical Assumptions

There are many unknowns about the heat generation of our components, so for this analysis, several assumptions were made in order to estimate the amount of cooling needed for the upper and lower compartments of the actuator table.

Based on the NEMA 23 Data Sheet, the motors will be drawing a combined 806.4 W from the power supply. The other components are only drawing a combined 135.157 W from the power supply. Based on these power draw numbers, we can assume that cooling for the motors will be most critical.

For this analysis, we will assume constant motor operation at an efficiency of 80%. This is a far worse operating condition than anything our motors will experience during normal operation of the game. With this assumption, we can calculate that the motors will be producing 161.28 W of heat. Therefore $\dot{Q}$ = 161.28 W.

We also know from the NEMA 23 data sheet that the maximum operating temperature of the motors is 122°F or. Since we don't ever want our motors to reach that temperature, we will pick the temperature of air exiting the table to be 92°F or 306.5K. We know the average temperature in Endeavor is 70°F or 294.3K, so that is what we picked for our input temperature. The heat capacity and density of air was taken at the average of the input and output temperatures which we found to be 300.4K. The heat capacity c_p = 1007 J/kg-K and the density ρ = 1.1614 kg/m^3.

Finally, we will assume a heat transfer effectiveness of 20% between the motors and the moving air. This will be shown with ϵ = 0.2.

9.5.2 Calculations

To find the required air flow rate to cool the motors, we used a modified version of the heat capacity equation:

$$\dot{m} = \frac{\dot{Q}}{\varepsilon \, C_{p\,(T_{out} - T_{in})}} \quad (20)$$

where $\dot{m}$ is the flow required flow rate of air through the table. After substituting all the variables into the equation we find a flowrate of $\dot{m}$ = 0.0655 kg/s. Using the density found at the average

temperature in the table and converting to cubic feet per minute we calculate an airflow rate of $\dot{m}$ = 119.5 CFM.

With 140 mm fans which produce an air flow rate of 72.8 CFM, we needed 2 fans on input and 2 fans on output for each compartment of the table for a combined flowrate of 145.6 CFM in each compartment of the table.

After prolonged use and testing, the motors never approach failure temperatures. They will become warm but not too hot to touch, which is a good indication that they are operating well within the heat margins we set up.

10 Concept Evaluation

Electric Drive Type	Max V	Accel Range	Load Range	Precision	Complexity	Cost	Total(least is better)
Lead Screw	6	4	3	2	6	4	25
Spline Gear with Belt	2	2	4	4	1	1	14
Rack and Pinion(straight)	5	6	2	6	3	5	27
Rack and Pinion (helical)	4	5	1	5	2	6	23
Belt (moving)	1	1	5	3	4	2	16
Belt (fixed)	3	3	6	1	5	3	21

Figure 42: Actuator Configuration Decision Matrix

During the Preliminary Design Phase of this project, we generated 3 different actuator concepts which would facilitate the range of motion required by the foosball rods. The first two concepts

are similar, utilizing a gantry for linear motion driven by a motor which drives a Rack-and-Pinion and a Timing Belt respectively. The rotational motion is handled by a second motor located on the gantry. The problem with these concepts is that in order to supply the necessary torque on the rotational motor, the motors are quite heavy. This results in a much higher torque on the linear motor, and a slower response time for the linear motion. This is, of course, undesirable, so we also devised a third concept which utilizes a timing belt and gantry combined with a long spline gear. This allows our rotational motor to remain stationary, while also supplying the desired torque to the rod for kicking. After evaluating all the concepts in a decision matrix, we determined the best concept for the job was Concept 3.

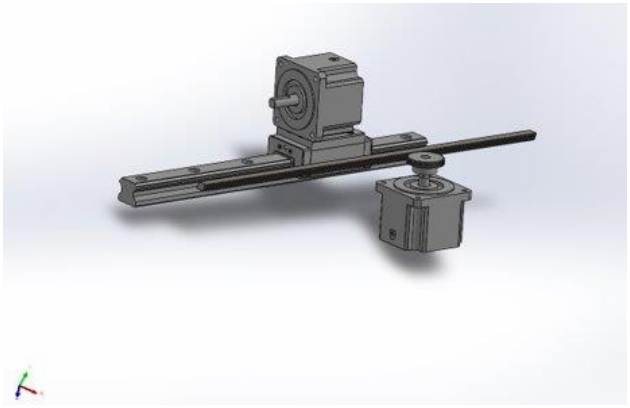


Figure 43:Concept 1- Rack and Pinion

Figure 44: Concept 2- Timing Belt

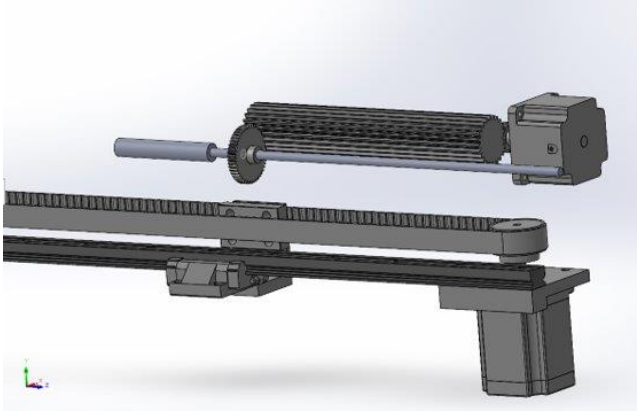


Figure 45: Concept 3-Timing Belt with Spline Gear

11 Integration Testing and Quality Assurance

11.1 Hardware Testing

To test the functionality of the hardware, a singular rod system was assembled to perform stress tests and verify that the components could withstand the applied forces before full assembly. The motors were tested by incrementally increasing the speed and acceleration to the calculated maximum forces that the system could withstand. The final values for the rotational motors came to be 20 rotations per second with an acceleration of 125 rps^2 and the translational motor was set to 2.67 m/s with a max acceleration of 10 meters per second per second.

The final step in mechanical testing following final assembly was extended play stress testing to weed out mechanical failures. Though no major failures arose, the table was flush with minor changes that needed to be made such as the tightening of set screws that connected the rod to the collared ball bearings. It was also found that the heat set inserts used within the rotational motor easily became dislodged, this was corrected by securing them with a two-part epoxy.

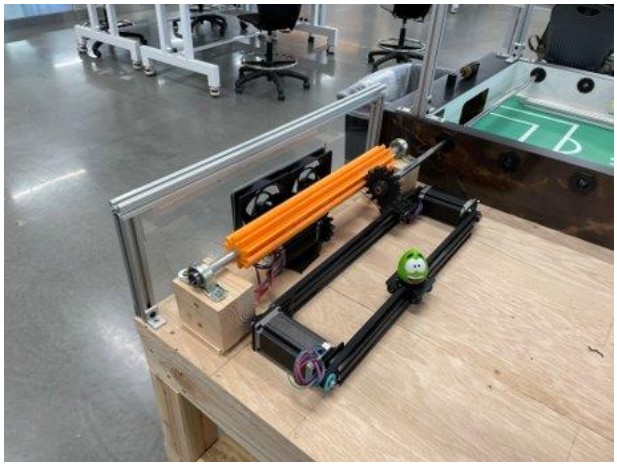


Figure 46: Hardware Performance Test Bed

11.2 Software Testing

Testing occurred as a continuous process throughout the development for features on the table. The first stage of testing consisted of the development of communication infrastructure and ensuring the data encoding and decoding processes were done successfully on both ends of the transmission and that transmission occurred in reasonable time. In our case <1 millisecond.

Once communication was verified, we began testing individual modules for accuracy. This included ensuring that ball locations were accurate values and that the rods moved to the exact positions they were instructed to. This stage culminated with a test lining a player up with the ball location on each rod simultaneously. With accurate control of the table, we began optimizing the strategy of the table taking particular care to avoid situations that endanger the table. This phase required adjusting numerous values such as the angle the rods swing back and forth for a kick or where the block state should position players. With this completed, we had a table that emulated the play of a novice human player.

```
#NETWORK
LOCALHOST = "127.0.0.1"
PI_ADDRESS = "192.168.0.1"
PORT = 5000

#STATE
MOVEMENT_MARGIN = 2
KICK_TIMEOUT = 1
LAST_POSITION = -1
PLAYER_LENGTH = 4
NOISE_THRESHOLD = 3
MIN_VELOCITY_THRESHOLD = 300
OPEN_PREP_RANGE = -30
BLOCK_PREP_RANGE = 100
OPEN_KICK_RANGE = -20
BLOCK_KICK_RANGE = 60
KICK_ANGLE = 60
PREP_ANGLE = -30
BLOCK_ANGLE = 0
OPEN_ANGLE = -90
SPEED_THRESHOLD = 3000
MIN_PLAYER_OFFSET = 40
MAX_PLAYER_OFFSET = 640
IDLE_RANGE = 600
RECOVERY_LINEAR = 80
RECOVERY_ANGLE = -57
```

Figure 47: Values Defining Foosbot Strategy

```
//Clockwise motor movement is positive

DIRECTIONS[4][2] = {{ -1,  1}, //3 rod
                   { -1,  1}, //5 rod
                   { 1, -1}, //2 rod
                   { -1,  1}};//Goal rod

MAX_TRANSLATIONS[4] = {181.23, //3 rod
                       115.265, //5 rod
                       356,    //2 rod
                       228.77}; //Goal rod

DEGREES_PER_REVOLUTION = 360;

STEP_PULSE_ROTATION_CONVERSION = 3200; //pulse per rotation

STEP_PULSE_TRANSLATION_CONVERSION[4] = {41.27, //3 rod
                                        41.27,  //5 rod
                                        41.27,  //2 rod
                                        41.27}; //Goal rod pulse per mm

MAX_SPEED_ROTATION = 20; //rotations per second
MAX_SPEED_TRANSLATION = 2670; //mm per second
MAX_ACCELERATION_ROTATION = 125; // rotations per second per second
MAX_ACCELERATION_TRANSLATION = 10000; // mm per second per second
HOME_SPEED_TRANSLATION = 100;
HOME_SPEED_ROTATION = 1;

COM_DELAY = 10; //in ms
MAX_COM_DELAY = COM_DELAY * 6;

SENSOR_DEBOUNCE = 10;

BAUD_RATE = 1000E3;
```

12 Cost Breakdown

Featured below is the anticipated cost breakdown of the electrical and mechanical components of our design, barring any major complications our project was expected to be nearly 40 percent under budget. However, this cost sheet did not account for several things, the first being components which we were unable to use. There were various items which we ordered, which, due to poor tolerances, did not fit with other components we needed to use. We were forced to reorder several components due to this. Additionally, we did not account for the exact cost of several items which were used in the construction of the actuator table, such as fasteners, door hardware, and paint.

Mechanical	
Vision Framing	$ 639.24
Rotation Motion	$ 663.70
Linear Motion	$ 747.24
Actuation Table	$ 956.52
Electrical	
Processors	$ 2,162.32
PCBs	$ 233.09
Power Supply	$ 330.31
Peripherals	$ 195.45
Total	**$ 5,927.87**
Budget	**$10,000.00**
Variance	**$ 4,072.13**

Figure 48: Anticipated Cost Breakdown

During the Fabrication phase, we encountered several complications which caused our costs to balloon somewhat. Better planning and organization on our part could have prevented some of this. Additionally, there were several connectors, cables, and small electrical components which were not accounted for in our original cost analysis. Our final costs were $7,761.55 and our remaining budget was $2,238.45. This is more than 20% under budget. Below is a breakdown of what we actually spent for the project, and our complete purchase order is included in Appendix C.

Mechanical	
Total Cost	$ 4,014.93
Electrical	
Total Cost	$ 3,841.55
Budget	$10,000.00
Budget left	$ 2,143.52

Figure 49: Actual Cost Breakdown

13 Project Plan

During the Preliminary Design Phase of the project, we generated rough concepts for our actuator design and decided on communication and data standards, as well as the type of vision algorithm we would use. We also constructed various decision matrices to help us determine the best actuation method. Additionally, we made a Gantt chart to help us stay on track and make sure we did not get behind in future project phases.

Task Name	Duration	Start	Finish	Predece
Conceptual Design Phase	**12 days?**	**Thu 1/13/22**	**Fri 1/28/22**	
rough mechanical drive ideas	5 days	Tue 1/18/22	Mon 1/24/22	
get high speed camera working	1 day	Wed 1/19/22	Wed 1/19/22	
rough speed and torque calc	6 days	Wed 1/19/22	Wed 1/26/22	
Decide on Vision Tracking Criteria/AI Types	5 days	Wed 1/19/22	Tue 1/25/22	
Preliminary Motor Sizing	1 day	Thu 1/27/22	Thu 1/27/22	4
decide on Data needed	2 days	Wed 1/26/22	Thu 1/27/22	5
Preliminary Design Powerpoint	4 days	Mon 1/24/22	Thu 1/27/22	
Preliminary Design Presentation	1 day	Fri 1/28/22	Fri 1/28/22	8
Detailed Design Phase	**20 days?**	**Mon 1/31/22**	**Fri 2/25/22**	**1**

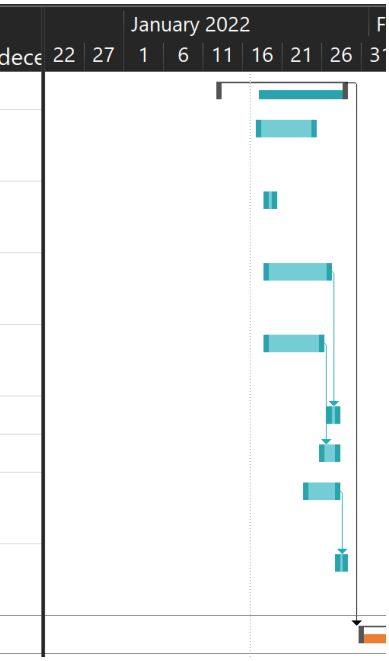
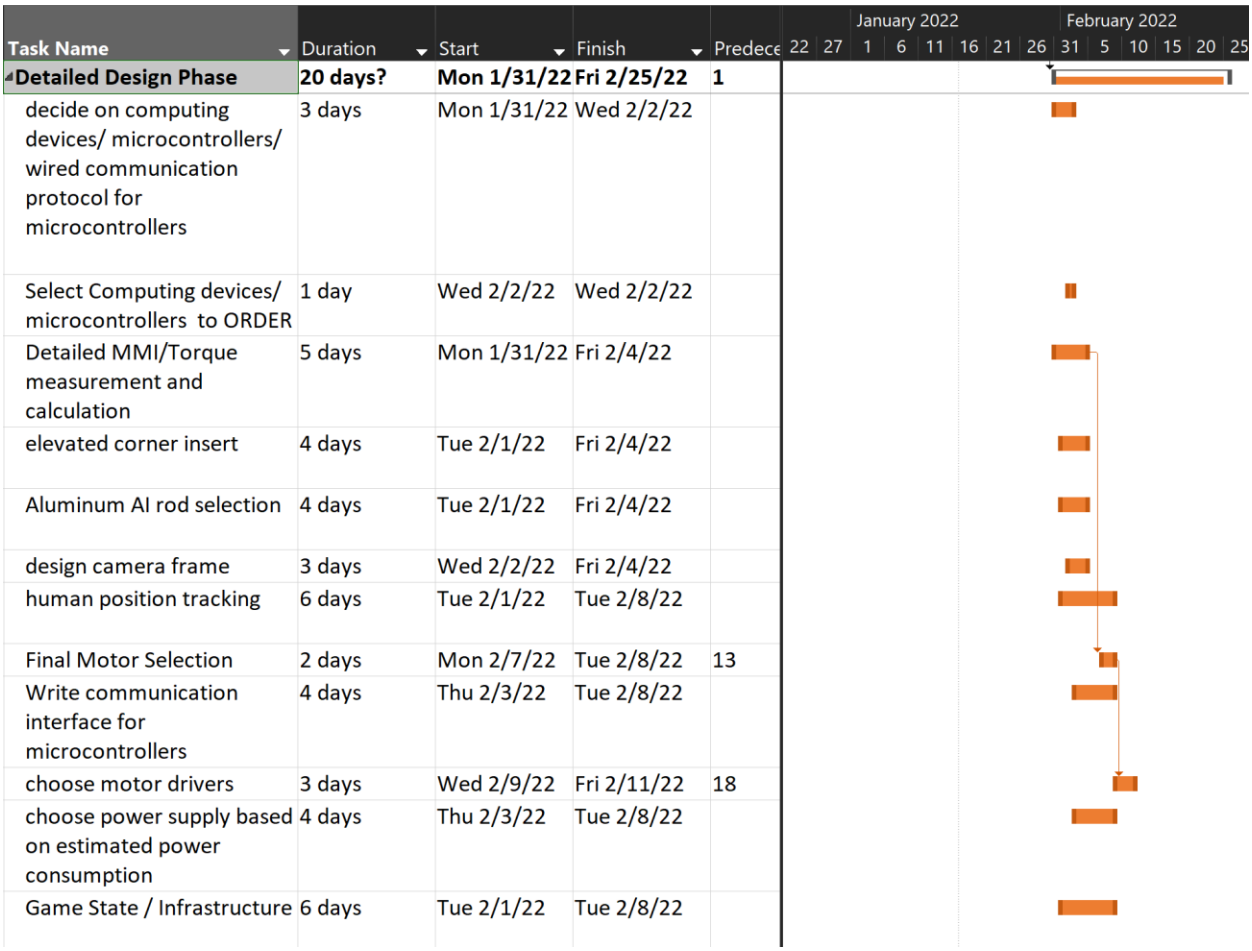


Figure 50: Preliminary Design Phase

After Preliminary Design Review, we worked hard researching and fine tuning our approach. Taking into consideration much of the feedback from the PDR, we settled on our processor and motor selection, as well as designing the many CAD models needed to solve the mechanical challenges we faced.



Task Name	Duration	Start	Finish	Predece
Detailed Design Phase	**20 days?**	**Mon 1/31/22**	**Fri 2/25/22**	**1**
decide on computing devices/ microcontrollers/ wired communication protocol for microcontrollers	3 days	Mon 1/31/22	Wed 2/2/22	
Select Computing devices/ microcontrollers to ORDER	1 day	Wed 2/2/22	Wed 2/2/22	
Detailed MMI/Torque measurement and calculation	5 days	Mon 1/31/22	Fri 2/4/22	
elevated corner insert	4 days	Tue 2/1/22	Fri 2/4/22	
Aluminum AI rod selection	4 days	Tue 2/1/22	Fri 2/4/22	
design camera frame	3 days	Wed 2/2/22	Fri 2/4/22	
human position tracking	6 days	Tue 2/1/22	Tue 2/8/22	
Final Motor Selection	2 days	Mon 2/7/22	Tue 2/8/22	13
Write communication interface for microcontrollers	4 days	Thu 2/3/22	Tue 2/8/22	
choose motor drivers	3 days	Wed 2/9/22	Fri 2/11/22	18
choose power supply based on estimated power consumption	4 days	Thu 2/3/22	Tue 2/8/22	
Game State / Infrastructure	6 days	Tue 2/1/22	Tue 2/8/22	

Figure 51: Detailed Design Phase (Initial)

During the initial phase of the detailed design phase, we calculated power consumption to determine the necessary power supply we'd need and designed the software infrastructure and game state. A multitude of calculations were taken such as torque calculations, and specifications for camera mount frame.

Task Name	Duration	Start	Finish	Predece
Game State / Infrastructure	6 days	Tue 2/1/22	Tue 2/8/22	
Algorithm Research	6 days	Tue 2/1/22	Tue 2/8/22	
Cam Research	6 days	Tue 2/1/22	Tue 2/8/22	
Sensor Research/ Implementation	6 days	Tue 2/1/22	Tue 2/8/22	
Define CAN	6 days	Tue 2/1/22	Tue 2/8/22	
spline gear sizing	8 days	Wed 2/2/22	Fri 2/11/22	
Motor Control	7 days	Thu 2/3/22	Fri 2/11/22	
SOP Submitted to Dr. Connor	5 days	Mon 2/7/22	Fri 2/11/22	
motors, motor drivers and PCB drivers Ordered	9 days	Tue 2/8/22	Fri 2/18/22	
Design PCB	5 days	Mon 2/14/22	Fri 2/18/22	
Detailed actuator CAD Modeling	15 days	Mon 1/31/22	Fri 2/18/22	
Strength Calculations and FEA	4 days	Tue 2/15/22	Fri 2/18/22	
SOP to SRB	5 days	Mon 2/14/22	Fri 2/18/22	29
AI rod covers/gui display stand	5 days	Mon 2/14/22	Fri 2/18/22	
Software Unit Testing	6 days	Mon 2/14/22	Mon 2/21/22	
Full Purchase Request Excel Sheet	9 days	Mon 2/14/22	Thu 2/24/22	
CDR Powerpoint	6 days	Thu 2/17/22	Thu 2/24/22	
CRITICAL DESIGN REVIEW	3 days	Tue 2/22/22	Thu 2/24/22	
CRITICAL DESIGN REPORT	6 days	Fri 2/25/22	Fri 3/4/22	39

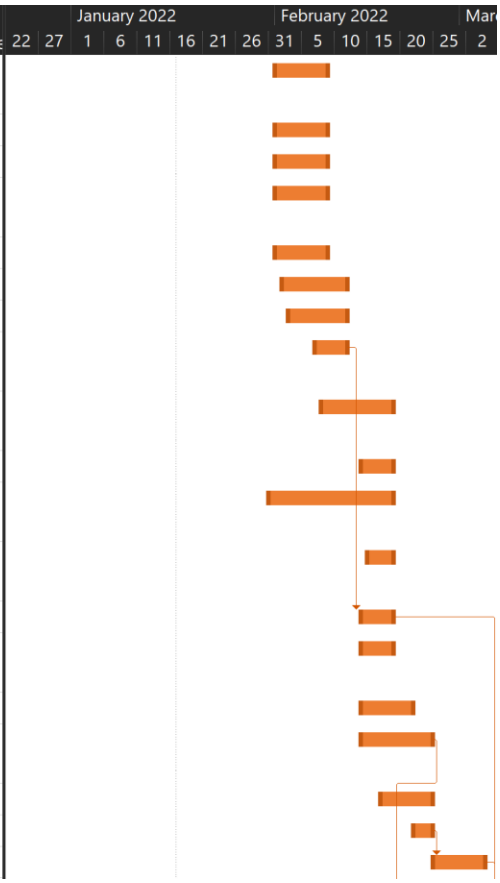


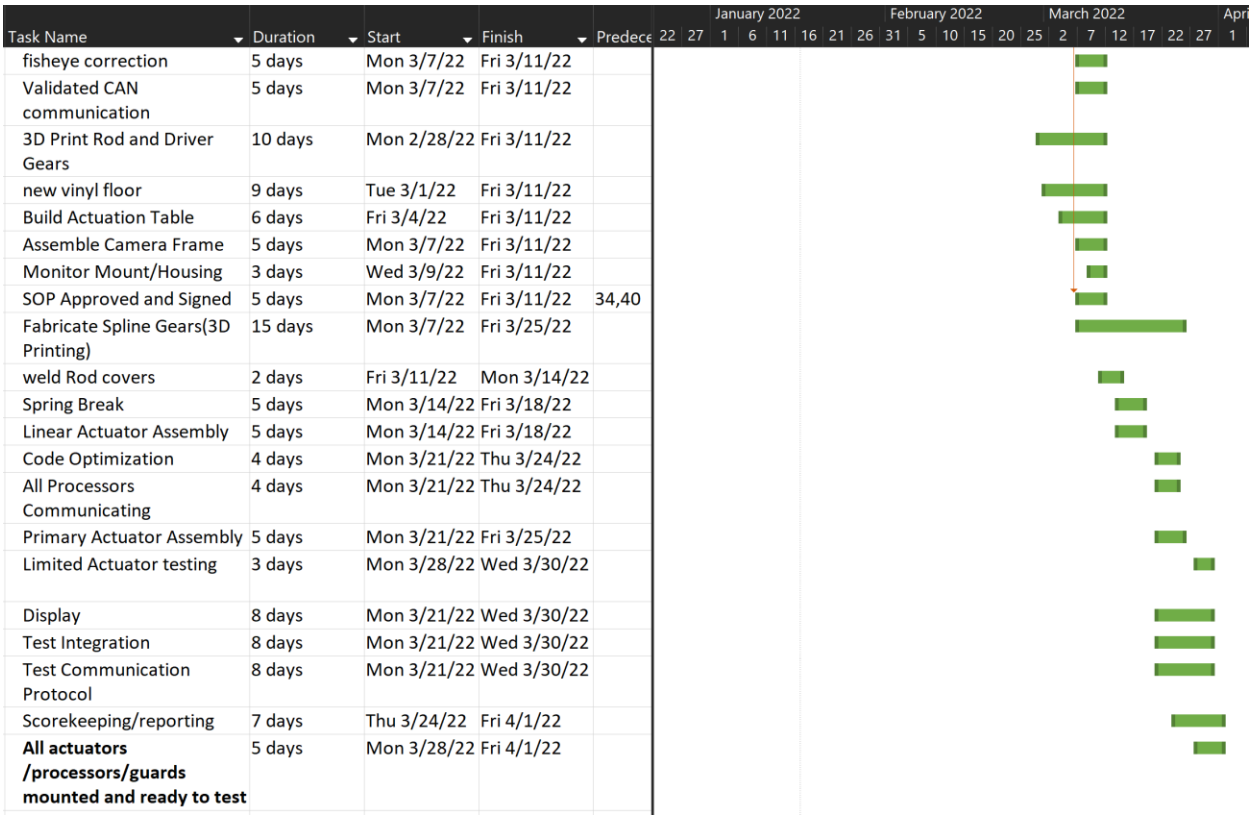
Figure 52: Detailed Design Phase (Final)

For the second half of the detailed design phase, we dedicated our time to research and determining a clear picture for the design of the table and software. Vision, sensor, and communication research took roughly a week and was followed by implementing our various software solutions. FEA analysis and strength calculations were taken, followed by creating a thorough and detailed CAD model (such as seen in this report). This phase came to a culmination with the critical design review.

Task Name	Duration	Start	Finish	Predece
CRITICAL DESIGN REPORT	6 days	Fri 2/25/22	Fri 3/4/22	39
Fabrication Phase	**30 days?**	**Mon 2/21/22**	**Fri 4/1/22**	**37**
machine trainings completed	18 days	Wed 2/9/22	Fri 3/4/22	
Write normalization algorithm to match the frame of reference to consistent reference	8 days	Mon 2/21/22	Wed 3/2/22	
Write predictive algorithm	8 days	Mon 2/21/22	Wed 3/2/22	
Vision Algorithm Done	8 days	Mon 2/21/22	Wed 3/2/22	
Write Finite State Machine (Command Module)	8 days	Mon 2/21/22	Wed 3/2/22	
Fabricate and assemble PCB	8 days	Mon 2/21/22	Wed 3/2/22	
Write library to interface with determined wired protocol and the Command Module	8 days	Mon 2/21/22	Wed 3/2/22	
Design motor controller	3 days	Mon 2/28/22	Wed 3/2/22	
order all necessary components	5 days	Mon 2/28/22	Fri 3/4/22	
Wire devices to power supplies	8 days	Mon 2/28/22	Wed 3/9/22	
Peer Eval #1 Due	1 day	Fri 3/4/22	Fri 3/4/22	
implement FSM	4 days	Mon 3/7/22	Thu 3/10/22	
fisheye correction	5 days	Mon 3/7/22	Fri 3/11/22	

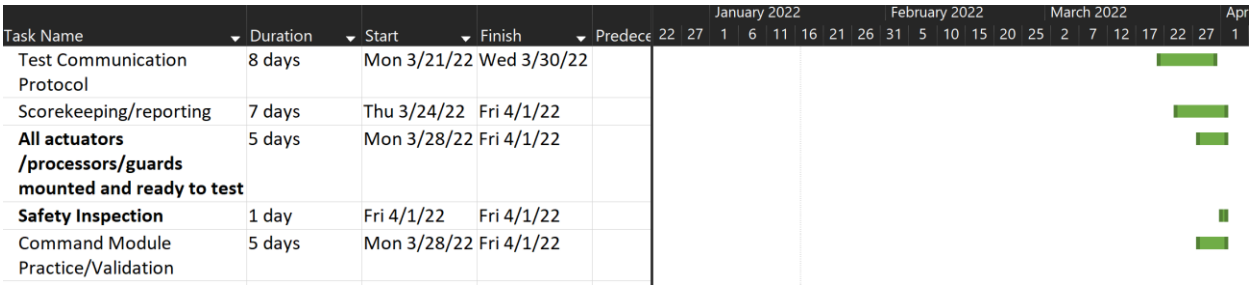
Figure 53: Fabrication Phase (Initial)

The first portion of our fabrication phase was spent finishing our machine trainings and completing our various software modules. This will also be the time where a finalized parts will be presented and ordered. After purchasing many of our components, we could begin fabrication by building our actuator table and 3D printing our trial spline and rotational gears.



Task Name	Duration	Start	Finish	Predece
fisheye correction	5 days	Mon 3/7/22	Fri 3/11/22	
Validated CAN communication	5 days	Mon 3/7/22	Fri 3/11/22	
3D Print Rod and Driver Gears	10 days	Mon 2/28/22	Fri 3/11/22	
new vinyl floor	9 days	Tue 3/1/22	Fri 3/11/22	
Build Actuation Table	6 days	Fri 3/4/22	Fri 3/11/22	
Assemble Camera Frame	5 days	Mon 3/7/22	Fri 3/11/22	
Monitor Mount/Housing	3 days	Wed 3/9/22	Fri 3/11/22	
SOP Approved and Signed	5 days	Mon 3/7/22	Fri 3/11/22	34,40
Fabricate Spline Gears(3D Printing)	15 days	Mon 3/7/22	Fri 3/25/22	
weld Rod covers	2 days	Fri 3/11/22	Mon 3/14/22	
Spring Break	5 days	Mon 3/14/22	Fri 3/18/22	
Linear Actuator Assembly	5 days	Mon 3/14/22	Fri 3/18/22	
Code Optimization	4 days	Mon 3/21/22	Thu 3/24/22	
All Processors Communicating	4 days	Mon 3/21/22	Thu 3/24/22	
Primary Actuator Assembly	5 days	Mon 3/21/22	Fri 3/25/22	
Limited Actuator testing	3 days	Mon 3/28/22	Wed 3/30/22	
Display	8 days	Mon 3/21/22	Wed 3/30/22	
Test Integration	8 days	Mon 3/21/22	Wed 3/30/22	
Test Communication Protocol	8 days	Mon 3/21/22	Wed 3/30/22	
Scorekeeping/reporting	7 days	Thu 3/24/22	Fri 4/1/22	
All actuators /processors/guards mounted and ready to test	5 days	Mon 3/28/22	Fri 4/1/22	

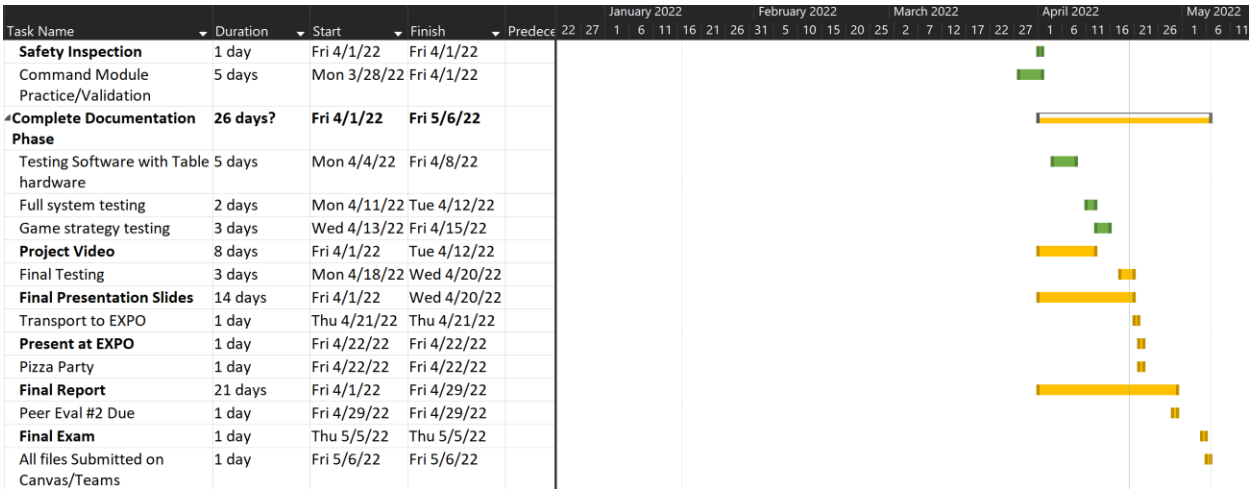
Figure 54: Fabrication Phase (Intermediate)



Task Name	Duration	Start	Finish	Predece
Test Communication Protocol	8 days	Mon 3/21/22	Wed 3/30/22	
Scorekeeping/reporting	7 days	Thu 3/24/22	Fri 4/1/22	
All actuators /processors/guards mounted and ready to test	5 days	Mon 3/28/22	Fri 4/1/22	
Safety Inspection	1 day	Fri 4/1/22	Fri 4/1/22	
Command Module Practice/Validation	5 days	Mon 3/28/22	Fri 4/1/22	

Figure 55: Fabrication Phase (Final)

Final portion of our fabrication phase included full actuator construction and integration testing of the software. Additionally, the camera frame, acrylic paneling and zeroing hardware were also constructed This allowed for the electrical system to be setup and tested—fixing any bugs as necessary. This phase concluded with our safety inspection which was passed with flying colors. We could then begin full system testing.



Task Name	Duration	Start	Finish	Predece
Safety Inspection	1 day	Fri 4/1/22	Fri 4/1/22	
Command Module Practice/Validation	5 days	Mon 3/28/22	Fri 4/1/22	
Complete Documentation Phase	**26 days?**	**Fri 4/1/22**	**Fri 5/6/22**	
Testing Software with Table hardware	5 days	Mon 4/4/22	Fri 4/8/22	
Full system testing	2 days	Mon 4/11/22	Tue 4/12/22	
Game strategy testing	3 days	Wed 4/13/22	Fri 4/15/22	
Project Video	8 days	Fri 4/1/22	Tue 4/12/22	
Final Testing	3 days	Mon 4/18/22	Wed 4/20/22	
Final Presentation Slides	14 days	Fri 4/1/22	Wed 4/20/22	
Transport to EXPO	1 day	Thu 4/21/22	Thu 4/21/22	
Present at EXPO	1 day	Fri 4/22/22	Fri 4/22/22	
Pizza Party	1 day	Fri 4/22/22	Fri 4/22/22	
Final Report	21 days	Fri 4/1/22	Fri 4/29/22	
Peer Eval #2 Due	1 day	Fri 4/29/22	Fri 4/29/22	
Final Exam	1 day	Thu 5/5/22	Thu 5/5/22	
All files Submitted on Canvas/Teams	1 day	Fri 5/6/22	Fri 5/6/22	

Figure 56: Final Testing and Complete Documentation Phase

Our testing and complete documentation phase began after we passed our safety inspection. We did full system hardware and software testing while also completing our project video and preparing for EXPO. Finally, after EXPO we completed our final project report, as well as an end user manual and end user video guide at the request of our advisor Dr. Conner.

14 Risk Management

14.1 Risk Assessment

We designed our risk management plan to be broken down into 3 different categories, cost, productivity, and functionality. The legend as seen below, provides a risk score of 1-5 with the probability of it occurring and explanation of the weighting factor. We have mitigation plans that are similar for a couple of different vulnerabilities such as covid and weather. We have mitigated these by communicating routinely on teams and meeting remotely when we absolutely must but, if possible, we will always meet in person. This has allowed us to move along in this project in a very effective and timely manner. The availability of parts or product lead times, especially with the state of industry now, these can cause massive pain points in our schedule. We have been actively mitigating this by staying in contact with suppliers, we have kept a running Bill of Materials and when we are approved for purchasing, we order the parts immediately. The next set of vulnerabilities, which all relate to availability of Lab spaces or meeting availability of group members. We have avoided any issues with this by holding mandatory weekly meetings and we also keep a running record of availability of all group members in "WhenToMeet". The mitigations we have put in place here allow us to stay punctual with all our deadlines. We have a vulnerability in troubleshooting our hardware and coding, as seen from the chart below it is our single biggest vulnerability because it has the highest chance of hindering our productivity and the functionality of the whole project. We mitigated this by doing all the possible calculations and preplanning possible errors that could occur with possible solutions before they happen. We also have dedicated 2 weeks to nothing but troubleshooting just so this does not become a problem. Manufacturing or the machine work of parts is a possible vulnerability. We have made sure to erase this issue by having multiple people be certified in different areas of work and planned our trainings out very far in advance. As seen below, none of our overall vulnerabilities for any of the three major categories have a score above 2, which great for efficiency in reaching our goals in a timely manner.

LEGEND		
Relative Risk based on current state occurrence potential		
Risk Score	Probability	Explanation of weighting factor
1	0-10%	Extremely sure not to occur
2	11-40%	Almost sure not to occur
3	41-60%	An even chance to occur
4	61-90%	Almost sure to occur
5	91-100%	Extremely sure to occur

Risk Assessment Matrix	Project Impact Matrix				
Vulnerabilities	**Cost**	**Productivity**	**Functionality**	**Risk Score**	**Risk Mitigation Plans**
Covid	2	4	1	**3.00**	Maintaining daily interactions in Microsoft teams and using remote meetings when necessary
Availability of Parts/Products ordered	2	4	3	**3.22**	Ordering parts as soon as possible and maintaining contact with suppliers and availability of supplies
Availability or purchasing ability of a Foosball Table	1	1	1	**1.00**	Foosball table supplied by OSU
Availability of Lab Spaces for group work/Manufacturing	1	2	2	**1.80**	Communication between team members and scheduling lab usage as soon as it is needed.
Availability of Group Members(Non-Covid)	1	4	2	**3.00**	A weekly schedule is released for what each group member/team is required within their given availability as provided by "WhenToMeet" tool
Troubleshooting of Hardware/Coding	1	4	4	**3.67**	Do all calculations and preplan possible errors that could occur with possible solutions before they happen.
Weather	1	3	1	**2.20**	Communication in advance of meeting schedule in advance of known inclement weather.
Machine work/ Manufacturing of Parts	2	2	3	**2.43**	All necessary training was considered beforehand, experts were talked to beforehand, and anything that could be received stock and functional was preferred
Certifications	1	2	2	**1.80**	Team members were already given what certifications they are required individually with available trainings
AVERAGE OF RISK CATEGORIES	**1.33**	**2.89**	**2.11**	**2.46**	

Figure 57: Risk Assessment Matrix

14.2 Design Redundancy

This is part of our mitigation plan and is a backup design just in case our current design fails. We have created a traditional belt and sled design as seen below. The main advantage our current design gives us over this one is that this design is not as efficient and weighs more. We will have to upgrade our current linear motors to be able to account for the increased loads but are prepared to do if the occasion arises.

Figure 58: Mitigated Backup Design

15 Future Development

15.1 Vision and Object Detection

One Computer Engineer/Science Student. (You could make the argument for two, however, I believe it would ultimately be difficult to work with another person on this portion).

The next step for vision is to implement a neural network to be able to accurately detect the ball at all times. Currently, we have a number of dead spots on the table that are caused by the ball being either covered by a rod or player. A neural network that *learns* how to anticipate the ball location would be a major leap in the effectiveness of the project.

The first step to approaching this problem would be to build a data-set, hours of footage compiled and labeled that could be ran through the network. **This is the greatest challenge** to building the neural network, once the data set is gathered (correctly) building the rest of the network would not be that challenging.

For the sake of speed, using the GPU on the Jetson is also something heavily encouraged. Getting the current vision processing to run on the GPU is quite the challenge. Running neural networks on the GPU is far simpler than OpenCV.

15.2 Motor Control

One Electrical Engineering Student with controls and communication knowledge. (An argument could be made for two as there are two separate parts. However, both can be completed by one person within one semester if the individual has drive)

There are two parts that need to be added to the motor control. The first is over torque safety and the second is adding encoders to the motors.

Over torque safety is needed in two situations. The first is the translational motor trying to exceed its bounds. This will only happen if the stepper motor slips enough to not accurately know its own position. The second situation is the rotational motor catching the ball between the player and the table. This is the more pressing concern as this is unhandled by the controller and can break the mechanical system.

The way to handle over torque is to read the current going to each motor. There are a couple of ways of doing this, but the easiest and best way to do this is to get the current information from the PDP or power distribution panel. **It is possible that the PDP does not have this information, in which case another way will need to be implemented.** The PDP has CAN connections which we have confirmed send messages. However, we were unable to find any documentation that had the message definitions. All that is mentioned in the user guide is that the information can be read by a RoboRIO. I recommend calling the manufacturer or digging deeper into the internet to find documentation. If all else fails, the messages can be decoded through inspection. The CAN connections can be directly connected to the existing CAN line as it runs at 1Mbps. Once current is read, it can be used to disable the motors when the current draw exceeds a specified limit.

Adding encoders will require a remake of the PCB as the current PCB only has one pin for an encoder. I recommend adding four pin connections per encoder to allow the use of an encoder that uses SPI. Attaching the encoders to the motors will require some sort of mounting.

15.3 Goal Sensor

The player goal sensor will sometimes not sense a goal. I believe this is due to the ball bouncing over the break beam sensor. The robot goal has a bounce plate unlike the player goal. This problem should be easily solved by adding a bounce plate on the player goal side to prevent the ball bouncing over the sensor.

15.4 Server

The Server should be in its final state, and I would highly caution against any changes to this module. By its nature, any new data can be added to the game state without any change to the underlying infrastructure. The only limitations are that messages are processed sequentially and could slow down if overloaded. During operation at gameplay capacity, messages were still processed in under a millisecond which is far faster than the speed of commands that can be sent to the motor controllers.

The server runs on the Raspberry Pi, which is configured as a DHCP server, allowing it to give IP addresses to connected devices such as the Jetson Xavier. These devices are hardwired with an ethernet cable, but an ethernet switch could be added to the project. This would allow other devices to plug in to the table and communicate with the server. In this setup, the table would be configured to run normally with students plugging into the switch to test their own versions of the FSM or other modules.

15.5 Finite State Machine

Further development on this module would require a computer engineering/science student that is heavily software focused. Improvements could be made to the current FSM in the ways it tracks balls moving in the Y direction or in clearing balls out of risky positions. However, larger scale improvements would come from development in other modules. If motor controllers become able to detect and stop caught balls without skipping on the gears, the FSM would be free to play more aggressively when it kicks. Currently, predictions are made when the ball is not seen behind a rod or player, a better vision algorithm would allow for the ambiguity of these guesses to be less impactful. The current method for calculating intercepts would also improve performance with a more accurate velocity measurement.

The biggest improvement could come from a design for a reinforcement based neural network. **This should not be attempted until a plan is in place for human side tracking, persistent vision, and control implemented safety for caught balls.** It would be hard to train a network on a table when the ball position is lost on the table or to attempt to protect the table from damage. Once these conditions are met, development can begin. The first step will be to create a virtual table to train the network. Reinforcement learning requires thousands of practice games to be played, this should not be attempted on the table and would limit the learning to the skill of the trainer. Additionally, the processors on the table will not support this operation. A useful technology explored for this purpose is OpenAI Gym, specialized to create environments for reinforcement learning. As to how development of the network itself will go, I will leave that to you.

16 Work Breakdown Overview

- Slayter Teal
    - Software Overview, Vision and Perception, Graphical User Interface, and Project Plan
- Jackson Law
    - Testing, Overall Mechanical Design, Motor Sizing, Gear Design
- Jonathan Harris
    - Gantt Chart/Weekly Update, EHS, Global/Ethical concerns, Mechanical Design, Actuator Table design/heat analysis, camera frame/housing design, End user manual
- Alex Rivera
    - Actuation CAD Design, Gear Design, Costs
- Cole Mitchell
    - Software Standards, Server, FSM, and Software Testing
- Hunter Collins
    - Cad Models, Mechanical Design, Risk Assessment Analysis, Mechanical Codes and Standards
- Michael Thompson
    - System design, Controls, Communication, PCB, Power, Goal Tracking
- Kenny Gipson
    - Gear tooth strength calculation, Materials, Fatigue calculations, FEA slide
- Garrison Locke
    - CAD Modelling, Camera Frame/Mount, Rod Covers, Acrylic Guards
- Johnny Enriquez
    - Knowledge Acquisition, Cost Evaluation, Electrical Standards, pi side CAN module

17 Appendix A: Links and References

- DHCP Basics
  - https://docs.microsoft.com/en-us/windows-server/troubleshoot/dynamic-host-configuration-protocol-basics
- Building a DHCP Server into the Pi
  - https://www.technicallywizardry.com/building-your-own-router-raspberry-pi/
- Running scripts as startup
  - https://unix.stackexchange.com/questions/634410/start-python-script-at-startup
- The Camera we're using is the **elp usbfhd08s-l36**. It can be found at this link.
  - http://www.elpcctv.com/elp-full-hd-1080p-free-driver-usb20-high-speed-60-120-260fps-usb-camera-elpusbfhd08sl36-p-129.html
- Fisheye Correction
  - https://medium.com/@kennethjiang/calibrate-fisheye-lens-using-opencv-333b05afa0b0
- Running OpenCV in the GPU
  - https://www.jetsonhacks.com/2019/11/22/opencv-4-cuda-on-jetson-nano/
- Resolving NPM EACCESS Error (this was useful in solving a linux permissions error I was getting)
  - https://docs.npmjs.com/resolving-eacces-permissions-errors-when-installing-packages-globally
- Auto-start NPM (sets the app to run at boot)
  - https://www.npmjs.com/package/auto-launch
- Electron Packager (packages the app, needed for startup-at-boot)
  - https://github.com/electron/electron-packager

18 Appendix B: READMEs

18.1 Running Python from Startup

Follow the instructions here: https://unix.stackexchange.com/questions/634410/start-python-script-at-startup it's fairly self-explainatory.

For our startup scripts we copied the python codes we wanted to run at start into the `/bin` folder. This allows the script to be run with superuser permissions. For reference, the command to copy to the `/bin` directory is: `sudo cp ./"script name" /bin/"script name"`

This would mean the .service file would like like:

```
[Unit]
Description=My Script

[Service]
ExecStart=/usr/bin/python3 /bin/"your script here".py

[Install]
WantedBy=multi-user.target
```

On thing of note however, this will not work for any *graphical* application (i.e. a gui). The script will also crash if you try to open a window (such as to display the camera output). It handles print statements fine, although you won't see them unless you run `sudo systemctl status "your service here".service`.

The alternative is to use a `crontab` to set things at startup, but you'll have to figure that out yourself.

18.2 DHCP README

Here's where things get a little tricky...

18.2.1 Background

To allow communication between the Jetson and the Pi, we created a "closed-loop" network. Meaning that we could connect the Jetson and the Pi with an Ethernet cable and use TCP Sockets/IP to communicate data between the two devices.

Before I get into the *how*, I'll first explain the *why*.

Our original intention was to use UART to serially communicate data between the Jetson and the Pi. Easy right? Well... not quite. The Jetson has a multitude of UART ports across its various GPIO pins, however, I could not figure out how to get any of them to work. No amount of searching on the internet provided me with any answers... at least not ones I could understand. It didn't help that the Jetson comes packed with a custom Ubuntu OS, pre-configured with all sorts of neat things that are utterly useless to our project and served to do nothing more than hinder me.

Thus, I decided to use an Ethernet Cable to communication between the two devices. This, however, it not quite as simple as I make it out to be. In order to get this to work I had to setup a DHCP server on the Pi.

18.2.2 DHCP

If you're not familiar with DHCP, I would highly suggest reading the information on this link.

I converted the `eth0` port of the Pi into a DHCP server. Meaning it'll only respond to DHCP requests on that interface.

Essentially, a DHCP server allows the Pi to respond to a clients request for an IP address. When the Jetson (or any device for that matter) connects to the Pi's ethernet port, the Pi assigns an IP address to the Jetson.

The Pi has a static IP address assigned to its `eth0` interface, meaning that whatever device is connected to it can ping the Pi at `192.168.0.1` and always find the Pi. This is what allowed us to use a TCP Client to send location data from the Jetson to the Pi.

The specific steps I used to make the DHCP server can be found here:
https://www.technicallywizardry.com/building-your-own-router-raspberry-pi/

18.2.3 Connecting to the Internet

The downside to this is that *you cannot use the `eth0` port to connect to the internet*, which can be troublesome if a wireless access point (which I will now lovingly refer to as "WAP"). Although if you have a hotspot on your phone that would also work.

However, if you want to connect to the internet, there are a couple steps you'll have to follow (you're essentially turning off the DHCP server and removing the static IP on the `eth0` interface).

First you'll want to turn off the DHCP server by running:

```
sudo systemctl disable isc-dhcp-server.service
```

Then you want to navigate (in a terminal) to: `/etc/network/interface.d` (for the uninitiated run `cd /etc/network/interfaces.d`).

Open the `eth0` file (`sudo nano eth0`) and flip the commented lines. This file should look something like this:

After edit:

```
# for static IP and DHCP
#allow hotplug eth0
#iface eth0 inet static
#   address 192.168.0.1
```

```
#   netmask 255.255.255.0
#   gateway 192.168.0.1

# for general eth0 use
auto eth0
iface eth0 inet manual
```

Then reboot the Pi: `sudo reboot`.

Tada! That should do it!

To turn it back on you'd do the opposite:

```
sudo systemctl enable isc-dhcp-server.service
```

Flip the commented lines in `/etc/network/interface.d/eth0` to look like this:

```
# for static IP and DHCP
allow hotplug eth0
iface eth0 inet static
    address 192.168.0.1
    netmask 255.255.255.0
    gateway 192.168.0.1

# for general eth0 use
#auto eth0
#iface eth0 inet manual
```

Then restart the pi, `sudo reboot`.

Now the DHCP server is back online.

18.2.4 Troubleshooting

If for some reason it's not working you can run `sudo systemctl status isc-dhcp-server.service` and see if it had problems at boot. Otherwise... google is your best friend.

18.3 Image Processing README

Documentation for the Image Processing portion of the AI-Foosball Spring 2022 Senior Design Project.

If you are reading this, and are not familiar with either *Python* or *UNIX* terminal commands, then I'd **highly** suggest familiarizing yourself with them.

The user is a superuser so you shouldn't have any problems with permissions.

18.3.1 Technologies In Use and Their Purpose

- OpenCV
- Python3
- Sockets

18.3.2 Production vs Development Code

All production code is set as bootup scripts. These scripts are stored in the `/bin` directory and in general should never be directly edited (rather overwritten from the development repo).

To run and test development code, the respective production code has to be stopped. For example if a new ball*tracking algorithm is being written then the *ball_tracking.service* must be stopped before the new code can be ran. To do this you would run the following command:

```
sudo systemctl stop ball_tracking.service
```

To determine what script is running what code you'll have to go to `/etc/systemd/system/` and look for the .service that corresponds to whatever program you are trying to run.

After your development code is ready to be placed into production follow the below commands to either create a new startup script or update an existing one.

18.3.3 Running and Startup Execution

For normal execution, `sudo python3 "your_app_here.py"` while in the project directory is sufficient.

18.3.4 Setting to run at Startup

To run the script at startup we set ball*tracking.py to run as a _service*.

This ONLY works on linux!

The service script looks in the /bin folder for the .py file for execution. The .py file needs to be in a root folder so that it can be ran with root privileges at startup.

If you make changes to the code and what the startup script to reflect it, you'll have to re-copy the .py in the /bin directory via the following command:

```
sudo cp ./ball_tracking.py /bin
```

Then to enable/disable the script from running at startup run:

```
sudo systemctl enable|disable ball_tracking.service
```

That's all there is to it. The link is the tutorial we used to set up the service, should you need to add a new/different service.

If you want to *stop* the script, for instance to test/edit code run:

`sudo systemctl stop ball_tracking.service` and conversely `sudo systemctl start ball_tracking.service` to start the script again.

If, for some reason, the script doesn't appear to be working you can check the status of the script using the command:

```
sudo systemctl status ball_tracking.service
```

18.3.5 What We did and Why We Did it.

Along with the ball*tracking.py (which is the main program), several other programs are included. In `./tools/` the _fisheye_calibration.py* reads images from `./tools/imgs/` to determine the coefficients needed to produce the fisheye correction. The previously mentioned coefficients are stored in the calibration.json. The *calibration_check.py* shows the corrected video stream, you can use this to check how well the coefficients were calculated.

Believe it or not, taking images on the jetson was surprisingly difficult (I blame it on the arm architecture). So I wrote am program in `./tools/imgs/photos.py` that you can run to take images from the camera (Press **space** to capture, **esc** to close).

18.3.5.1 Purpose

The purpose of the jetson is to process image data, extrapolate the balls location and heading, then send the ball data to the pi for use in decision making. To achieve this we made use of OpenCV and Python for general programming, and TCP Sockets for communication.

To detect the ball, we used OpenCV and Python to build a blob detection algorithm to accurately detect the color of the ball. The algorithm would find the center of largest mass of colored pixels and return the camera frame's (x, y) coordinate in pixels. We converted from a pixel (x,y) to a millimeter (x,y) by applying a pixel-to-millimeter ratio based on the view of the camera and the size of the table. This method provided a surprising degree of accuracy that was suitable for the scope of this project.

18.3.5.2 Challenges

In the early stages of design, we outlined two parameters that would define the constraints of our vision processing: processing time and reaction speed. In order for the system to respond appropriately to a fast(ish) moving ball we needed the system to process a corresponding amount of frames per second. The figures below show the required processing times, dependent on frame rate, and the distance covered between two frames, at various ball speeds and frames per second. Ultimately, our deliverable was to process a single frame under 10ms meaning greater than 100 frames processed per second, this would provide us with an adequate understanding of the balls location and direction of movement at a variety of ball speeds.

Another key element we took into consideration was lighting. Changes in lighting can have drastic effects on the performance of any vision algorithm. To mitigate these effects we made use of multiple color masks in the Hue, Saturation, Value (HSV) color spectrum. Rather than the RGB spectrum, the HSV spectrum takes into account how light or dark the color is, allowing for greater tolerances in varying lighting conditions. Multiple threshold masks were used to

account for variances in color and lighting, at the end of the threads-holding operations, done using both masks, their values would be ORed resulting in a cleaner and more reliable location of the ball.

18.3.5.3 Results

We performed numerous tests in various lighting conditions to produce the ensure that the ball would be adequately detected at all times of day and throughout normal play. To ensure performance we made use of Numpy and OpenCV extensively, as in both cases, the python codes are wrapper functions for C code underneath. This ensured that writing the program in python would not result in any drastic performance increases.

18.4 GUI README

A basic GUI for the AI-Foosball Capstone Project.

run the application in developer mode using: `npm run start` in a command terminal.

To bring up the developer tools (while the app is running): `CTRL+SHIFT+I`

To compile, from the command line: `npx electron-packager .` The app will be packaged into `./foosball-gui-linux-armv7l`.

Use `./foosball-gui-linux-armv7l/foosball-gui` to run the compiled app.

18.4.1 Graphical User Interface

The graphical user interface was built using a number of *web based toolkits* chief among those are: JavaScript, ElectronJS, NodeJS, HTML, and CSS. If you aren't familiar with any of these there are a multitude of guides that teach you everything you could possibly need to know about it.

Now before you say, "Why in the world did he build the desktop app using JavaScript?!" I did it because I am familiar with the language and it was really easy to make using Electron.

If you look through `preload.js` (./src/preload.js) on line 61 is where the script polls the server for the data. Looking through this section tells you *exactly* what elements of the JSON object the GUI is looking for to update the player_score and robot_score fields in the app.

18.4.2 How I built It

The app is built using really basic HTML and JavaScript. ElectronJS is only used to run the app in a Desktop app. The caveat is that ElectronJS alters the running scope of backend and frontend processes.

This causes a problem when you want to write code for the frontend that relies on NodeJS modules. The change in scope that Electron introduces prevents Node modules from being

used in the frontend application. This blocks the usage of the `net` module, which allows us to poll the server for the game state.

However, ElectronJS provides a way to "preload" JavaScript into the frontend application. `index.js` is the entrance for the Electron app which creates and loads the window, it also allows for these "preloaded" scripts to be added to the app. This means that we can use Node modules in frontend code, resolving the challenges with process scope.

I'm fairly confident there is a better way to do this but the approach requires a greater depth of knowledge than I possess. After spending hours upon hours reading the electronJS documentation, the approach I used was the one that *easily* fit our purposes.

18.5 Server README

Documentation for the Server portion of AI-Foosball Spring 2022 Senior Design Project.

18.5.1 Purpose

This module serves as a single source of truth for all data in the program. Each other module either generates data that is sent to the server or requests data from the server. Providing a single point of data transfer ensures that each additional module only requires one socket connection for communication with any other part of the system. Providing a layer of abstraction preventing clients from needed any understanding of the operation of other topics. The server functions by maintaining a dictionary object called the GameState which holds the key,value pair of all data sent to the server. Communication occurs by clients sending their own dictionaries formatted as JSON strings that the server then uses to update or add to the GameState. Requests for data see clients sending a dictionary holding the requested values where the server compares keys and returns its stored values. If running a python file, the Message.py file can be used to automatically format python dictionary objects to JSON strings for sending. Clients in other language must handle the JSON formatting manually.

18.5.2 Values Available in GameState

Rod Data

values for specific rods follow this format `robot_(rod descriptor)_rod_(displacement/angle)_(current/command)`

rod descriptors include (goal, 2, 5, 3) denoting which rod is referenced by the number of players

displacement/angle denotes which axis of the rod is referenced

current values are generated by the moter controllers indicating the position the rod thinks it is in

command values are generated by the finite state machine and indicate the position rods should move to

example full value being "robot_goal_rod_displacement_command":143

all values in milimeters

Ball Data

- ball_x: x position of the ball
- ball_y: y position of the ball
- ball_Vx: x component of ball velocity
- ball_Vy: y component of ball velocity
- float values in mm and mm/s

Game Data

- game_flag: boolean value indicating if a game has been started
- pause: boolean value indicating if the game is paused
- stop: boolean value indicating if the rods are stopped
- player_score: the score of the human player
- robot_score: the score of the robot player

CAN DATA

- USB2CAN: success or error message from CAN initialization
- unknown CAN messages are posted with their id as the key and data as the value, both in hexadecimal representation

18.5.3 Files

18.5.3.1 GameState.py

class file for the game state, initialized with an empty dictionary

update_game_data: takes an input dictionary and updates local dictionary. new keys are added and shared keys get updated values

get_game_data:updates values of the input dictionary with the values from identical keys in game state. Keys that can not be located are filled with the string “not found”

get_all_data:returns the entire game state dictionary

18.5.3.2 GameServer.py

This is the actual server file and utilizes the concept of a python selector. Selectors allow for sequential handling of multiple socket connections. A sequential server was created as it eliminated the possibility of race conditions or other data access errors that occur from threading. In addition sequential processing still proved to be fast enough for our purposes. I would highly caution against changing the functioning of the server.

The server exists on port 5000 of the raspberry pi with the current ip address of 192.168.0.1

The server waits in an infinite loop for connections from clients. When a message is sent the server checks if the socket exists in the selector, if not the accept_wrapper function is called

where the socket information is added to the selector. Subsequent read or write requests are then recognized by the selector in the service_connection method.

Service connection first checks if the client requested a read or write. With writes, any data stored in the selector for the socket is sent. Reads start by reading the first four bytes of the message, which must be an integer denoting the length of the rest of the message. Once the rest of the message is read it is converted from a JSON to a dictionary object. This dictionary must contain a key denoted "action" with a value indicating the service requested by the client. Available actions include "POST", "GET", and "DUMP". POST adds the sent dictionary to the game state. GET creates a dictionary of the requested values and formats them to be sent back to the client before saving the data back to the selector. DUMP acts similarly including every value currently in the game state. If anything goes wrong, a dictionary key "error" will be included with an error message.

Clients should send an empty byte string to close the socket connection on the server. Additionally messages from the server are not prefaced with their byte length

18.5.3.3 Message.py

The message class serves to abstract communication to the server for clients written in python. Clients not using this class must send an integer length and JSON formatted dictionary manually to the server.

Initialization of a message creates an empty dictionary and requires an action string. Known actions by the server are POST, GET, and DUMP. The python update method can then be used to add any other key, value pairs to the message dictionary. When ready to send, encode_to_send(True) creates a binary string with length header that is ready to be sent to the server. Using False does not add a length header.

```
message = Message("GET")

message.data.update(server_data)

sock.sendall(message.encode_to_send(True))
```

Decoding can be done by creating a message object and using the method decode_from_receive on the binary data

```
recv_data = sock.recv(1024)

received = Message("RECEIVED")

received.decode_from_receive(recv_data)
```

The dictionary of this message now contains the returned data

18.5.3.4 can_com.py

The purpose of this code is to handle sending messages between the CAN and the server.

initializeSocket will connect to the server if the server is running and will loop trying to connect if the server is not running

initializeCan will create a CAN connection if the USB2CAN adapter is connected to the pi This will keep looping if it does not connect and will update the server on its connection status

receiveCan will receive messages from the CAN bus and will decode them according to our premade CAN messages. It will look for which rod is communicating along with its current position It will also get the goal messages from the CAN bus This ends with posting all of the information to the server

sendCan will request specific information from the server The information received is determined by the local dictionary in the function It will look for the game to be started before it continues. When the game is started, it will send an initial zero command to each rod as well as reset the score in the CAN This then takes the messages and decodes them to determine the translational and rotational desired location of each rod It then sends four messages to the CAN with a 10ms delay, one for each rod

The try block calls the initializeSocket twice, one socket for receiveCan and one for sendCan This then calls initializeCan Last it will create two threads so the receiveCan and sendCan will run constantly and simultaneously.

18.5.3.5 Other Files

Other files in this directory contain various tests for specific functionality including message passing to the server and reading data

18.6 FSM README

Documentation for the Server portion of AI-Foosball Spring 2022 Senior Design Project.

18.6.1 Purpose

This module is responsible for determining commands for each rod based on the collected data from the camera and motor controllers. This data is received via the server and the resulting commands are sent back to the server in the format

```
robot_(rod descriptor)_rod_(displacement/angle)_command
```

rod descriptors include (goal, 2, 5, 3) denoting which rod is referenced by the number of players

displacement/angle denotes which axis of the rod is referenced

8 of these commands are generated for each set of data that is collected. The FSM does not care about how old the data in the server is or if it has already computed commands on that data previously. The FSM computes faster than vision collects data or the CAN bus sends data so this method ensures that the most recent and accurate commands are always available on the server.

18.6.2 Warning!!!

One must take special care in the design of the FSM or replacement for it. It is highly likely for the rods to either trap the ball between a player and the ground or up against a wall. This can not be detected or prevented by the motors and they will attempt to torque through the ball and potentially damage the rod mechanism or cause gears to skip. Skipped gears require being manually reset and increase the liklihood of further damage to the table and lack of control of the rods. For this reason rods should also be zerod whenever they become visually off from their correct position.

18.6.3 States

A breif description of each of the FSM states. Each rod operates a copy of this FSM simultaneously allowing the rods to all move independently.

18.6.3.1 Stop

This state is triggered externally when motor controllers are not sending commands, either from an emergency stop being triggered, communication failure, or the game being paused. In this state the FSM continually output the previous command leaving the rods in their current location.

18.6.3.2 Idle

When the ball is a certain distance away the rods placed in the center of the table. This stops the rods from responding to values far from the rod and places them in a central position to react as the ball approaches.

18.6.3.3 Open

This state is triggered when the ball is behind the rod. The players orient themselves parallel with the table to allow shots from behind to pass through undisturbed. The players still track the ball to place them in an optimal location after the ball passes underneath.

18.6.3.4 Block

This state is entered when the ball is in front of the rod. The 5 and 3 rod attempt to place a player directly in front of the ball to prevent forward progress. The goal and 2 rod place a player between the ball and the center of the goal, priortizing the defense of the goal.

18.6.3.5 Prep

When the ball is close enough to the rod, it swings the players back and attempts to position itself offset from the ball slightly in order to kick the ball at an angle to approach the opposing goal.

18.6.3.6 Kick

When the ball is closer to the rod. The rod waits until it has reached the location for kick before swinging the players forward.

18.6.3.7 Recover

This is a series of four states that attempt to move the ball out of a dangerous positon. Such as underneath the rod after a failed kick or behind the goal rod. These positions are very likely for the ball to trapped and damage the mechanisms of the table. The rod first moves itself to a vertical position and then to the side of the ball closer to the goal. The player than angles back and moves to the position of the ball. This should hit the ball from the side and push it to a better position to kick from.

18.6.4 FSM Constants

18.6.4.1 Network

Variables related to connection with the server - LOCALHOST: ip address of localhost - PI_ADDRESS: ip address of the server on the raspberry pi - PORT: port number of server on the raspberry pi

18.6.4.2 State

variables related to the transition and execution of states, all distances in millimeters - MOVEMENT_MARGIN: how close the rod must be to the desired location to transition in the recover states - KICK_TIMEOUT: number of seconds before the kick state is forcibly exited - LAST_POSITION: tells the FSM to resend the previous command - PLAYER_LENGTH: length of player pad from center point - NOISE_THRESHOLD: distance differences below this value are ignored to combat input noise - MIN_VELOCITY_THRESHOLD: velocities below this number are stationary - OPEN_PREP_RANGE: distance behind a rod where it enters the prep state - BLOCK_PREP_RANGE: distance in front of a rod where it enters the prep state - OPEN_KICK_RANGE: distance behind a rod where it enters the kick state - BLOCK_KICK_RANGE = distance in front of a rod where it enters the kick state - KICK_ANGLE: how far forward a rod swings when it kick - PREP_ANGLE: how far back a rod swings when it is prepped - BLOCK_ANGLE: angle for a blocking rod - OPEN_ANGLE: angle for an open rod - SPEED_THRESHOLD: speeds under this value will allow a kick attempt - MIN_PLAYER_OFFSET: the closest a player can get to the near wall with the bumpers - MAX_PLAYER_OFFSET: the closest a player can get to the far wall with the bumpers - IDLE_RANGE: distance between a ball and rod before it idles - RECOVERY_LINEAR: distance between ball and player when recovering - RECOVERY_ANGLE: angle rod takes when in recovery state

18.6.4.3 Physical Dimensions

dimensions measured on the table, in millimeters - GOAL_ROD: dimensions of the goal rod - TWO_ROD: dimensions of the 2 rod - FIVE_ROD: dimensions of the 5 rod - THREE_ROD: dimensions of the 3 rod - TABLE: dimensions of the table itself

18.6.5 Files

18.6.5.1 brainFSM.py

connect_to_server(): Given that the server exists on a different computer than the FSM, there is no guarantee that the server will exist when the FSM first starts. This method continually tries to connect to the server until the connection is successful, after a success the function returns the created socket.

compute_intercepts(): computes the y value the ball will cross each rod given its current velocity, returns -1 if the intercept is not in the table. This function does work in theory, however given the inconcistency of the velocity values the results of this function are not very reliable and were not used in the final implementation.

ball_speed(): similar to intercepts, the inconcistency of velocity values resulted in this value not being utilized as origionaly intended

main(): the main function starts by defining the dictionary keys for values needed from the server and keys being sent to the server as well as arrays for current states and kick timeouts. On each loop through the program, data is requested from the server and checked to ensure it was all retreived. If server data is not returned, the rest of the program is not run and the data is requested again. It is important that the server data dictionary is not overwritten as that would compromise future requests to the server. Once running, the location of the ball is checked. If the ball location is not found but the last known location is on the table the last ball location is used and the ball is marked as hidden. Additionally, if the new ball location is not further away than the noise threshold, the last ball position is used to avoid shaking the players with small position adjustments. If a goal was scored or the ball is seen off the table the ball location is marked as not found. A gather operation is performed computing the next state and command for each of the rods. A gather operations schedules each function to be executed asynchronously, retruning an array when all have completed. The output of the gather is then used to update the saved states and fill the dictionary to be sent to the server. A check is hten made to see if any rod entered the kick state. If so, a timer is started. Otherwise the timer is set to False. The compiled commands are finally sent to the server.

18.6.5.2 rodFSM.py

this file contains all the functions used to compute the actions of a single rod

compute_next_state(): Given the data from the server returns a value for the next state of a rod. It is important to make sure that the more specific states are checked first as broad conditions will prevent the other states from being reached. Stop and Idle both have straightforward implementations. The recover states have two conditions, one for moving from the previous state and another for remaining in that state. Recovery is entered if the ball is behind the goal rod or the kick timer expires. Kick can be entered once the ball enters a region around the rod or if the rod was prepped and then the ball becomes hidden. The rod assumes it is blocking the ball from the camera and kicks anyway. Prep is entered if the ball is in a wider

range from the rod and teh rod is open if the ball is behind it. If no other condition applies the rod should block.

compute_command(): calls and returns the specific function for each state

compute_rod_linear(): given a rod and desired Y location, this function returns the actuation distance to place a player at that location. This function will always put the same player at the same location and does not take into consideration which player is closer if two players can reach a position. If the desired Y is outside of the actuation range, the max or min actuation is returned. There are two scenarios for rods, if overlap exists between players of not. Overlap exists if the max actuation distance of the rod is further than the spacing between players. If no overlap exists, the distance past the players start location is returned by taking the modulus of desired Y and player spacing. The offset is subtracted from the desired Y as the players start away from the wall due to the bumpers. This method does not work on rods with overlap as it assumes the location of the second player at actuation 0 is the same as the first player at max actuation. We start by adjusting the actuation distance to split each of the overlapped regions in half. We then count how many players are between the desired Y and and the 0 position. For example, the 3 rod with a desired location in the middle of the field should use the second player. Player offset will be 1 as there is 1 player player between the second player and the 0 position. The assumed start postion of players is equal to half the offset multiplied by the player offset, this value is then added to the distance calculated for a no overlap rod with the adjusted actuation.

state_stop(): returns last position

state_idle(): returns vertical rods at half actuation

state_open(): returns horizontal rods with player at ball location

state_prep(): deltaY is the additional distance to the side of the ball needed to kick at an angle towards the goal. A player is sent to the ball location plus deltaY with a backwards angle

state_kick(): still outputs the location from the prep state and keeps the prep angle until it reaches the location for the kick. The angle is then set forward

state_recover_1(): sets the rod upright in its current positon

state_recover_2(): places the rod at a set distance on the inner side of the ball

state_recover_3(): rotates the rod back

state_recover_4(): moves the rod back to the ball location

18.6.5.3 Other Files

These consist of various other test such as isolating control of linear or rotational movement.

Running ESP32 code

Setting up Arduino IDE

This is written using Arduino 1.8.13 as a reference.

Download the Arduino IDE by clicking here.

Once the installer is downloaded, launch it and follow the prompts for instillation. Open application once it is installed.

Install Board

First thing to do is add the **ESP32 Dev Module** to your boards.To do this, click on `Tools->Board->Boards Manager`.

Then search for **esp32** in the *Filter your search* bar. **esp32** by **Espressif Systems** should appear. Select the newest version and click the install button. Then go back to `Tools->Board->ESP32 Arduino` and select the **ESP32 Dev Module**.

The essential settings underTools should be:

• Board: "ESP32 Dev Module"

• Upload Speed: "921600"

All other setting should be correct.

Once the board is plugged in, make sure to select the correct port.You can check the port by using Windows Device Manager.

Install Libraries

The Spring 2022 version of the controller code uses CAN and ESP-FlexyStepper.

Both libraries can be installed through Arduino IDE.To do this, navigate to `Tools->Manage Libraries`. Then search for **ESP-FlexyStepper** by **Paul Kerspe** and install newest version (current version 1.4.5). Do the same for **CAN**.

Because "can" is such a pervasive word, you will have to scroll down quite a bit. Make sure to select the library only called **CAN** by **Sandeep Mistry** as there are many CAN libraries (current version 0.3.1).

18.8 Controller README

Controller.ino

The libraries used in **Controller.ino** are CAN and ESP-FlexyStepper.

Controller.ino works on every rod without modification due to the constants being selected by the board ID which comes from two pins soldered to ground or power (`board_ID = digitalRead(ID_2)*2 + digitalRead(ID_1)`).

Each rod has a specific soldered ID which is commented in **Controller_Constants.h**.

Every arduino sketch always includes a `setup()` and `loop()`. All other functions are used inside those two functions.

This specific sketch relies heavily on global constants and the following global variables:

- state
- board_ID
- receive_time
- send_time
- emergency_stop
- translation_measured
- rotation_measured
- translation_desired
- rotation_desired

The global constants are found in **Controller_Constants.h**.

Note: the majority of the bulk for this sketch is in serial print statements. These statements can be turned off by setting the three constants in different ways: `SERIAL_ON`, `SERIAL_MESSAGES` and

`SERIAL_STATES`. If `SERIAL_ON = false` then no serial messages will be printed. If `SERIAL_ON = true` only a few essential messages will be printed. If `SERIAL_ON = true` and `SERIAL_MESSAGES = true` all CAN messages received and sent will be printed. If `SERIAL_ON = true` and `SERIAL_STATES = true` then all state transitions will be printed. If all are true, everything will be printed.

`setup()`

In this setup, we set up the following:

1. Begin Serial communication at 115200bps
2. Set mode to input on both ID pins and the ENABLE pin
3. Connect RX and TX pins to the CAN object
4. Set mode of the "all good" led to output
5. Connect the pulse and direction pins to both stepper objects
6. Set the board ID based on digitally reading the ID pins as binary digits
7. Set the "steps per" values and the acceleration and deceleration of both stepper objects
8. Start can with mask based on board ID and with specific baud rate with `start_CAN()`
9. Set the start time and message time variables to the current time
10. Do the first state evaluation by calling `evaluateState()` (Note: this does not need to be done because it is the first thing done in the `loop()`)

Outside of `setup()`, two `Sensor_Debounce` objects are bound to each zeroing pin. This was to keep the objects global.

For some reason, the pointer could not be declared null and then assigned in `setup()`.

Note: the stepper objects are ran on core 0 while everything else is ran on core 1.

`loop()`

A sketches `loop()` function is an infinite loop. In this loop, four functions are called.

1. `evaluateState()`

2. `CANReceiver()`

3. `setControl()`

4. `CANSender()`

However, `CANSender()` is only called at intervals based on the **COM Delay** variable.

`evaluateState()`

In short, there are five major states: **Zero**, **Starting**, **Disabling**, **Running** and **Disabled**. However, there are many conditions in which **Disabled** can be triggered and released. Because of this, **Disabled** is split into the following five sub-states: **Short CANWait**, **Long CANWait**, **Emergency Stop**, **Stop Switch** and **Large CAN Delay**.

ZERO

The zero state is the initial state of the system and is only entered on three conditions.

The first is that the emergency stop switch has been released. This is because this switch bypasses the controller and disables the drivers. Because of this, the internal step count will get off and therefore needs to be reset.

The second is if a *zero* message is received during `CANReceiver()`.

The third is if the stepper objects ever go 1mm negative.

The state does three major things. The first is call `emergencyStop()` on both steppers. The second is disabling the stepper services if they are running. Finally, `zero()` is called.

The only way to leave the state is if `zero()` returns true. Once true is returned, the speed of each stepper is reset, each stepper is started as a service again and due to zeroing taking a while, the `message_time` variable is set to the current time. The next state is always **Short CANWait** to prevent the motors from moving if there are no incoming messages.

RUNNING

The **running** state is the primary state. This state is the only state to set the *all good led* to on.To leave the running state, an emergency stop command can be received, the emergency stop switch is pressed, or the received message time exceeds the timeout. If any off these occur, the next state is set as **disabling**.

EMERGENCY_STOP

Emergency stop state is completely dependant on the global variable `emergency_stop`. This variable is set as true if the emergency stop message is received from the CAN bus in `CANReceiver()`. If any other message is received, `emergency_stop` is set to false. The state can only be left if `emergency_stop` is false. The next state is **Short CAN Wait** to prevent the motors from moving if there are no incoming messages.

Note: In the current system, the emergency stop command and state are not utilized. Instead a CAN timeout is used to stop the motors.

STOP_SWITCH

As long as the pin on the enable line is grounded, the stop switch will be pressed. As long as the switch is pressed, the **stop switch** state will never be left.

Once the stop switched is released, the next state is set to **zero**.

SHORT_CAN_WAIT

This state is the default disabled state. As long as no CAN message intended for the particular rod gets received, then this state will be active. If the stop switch is pressed or the emergency stop command is received, the next state will be **stop switch** or **emergency stop** respectively. If `millis() - message_time` is less than the timeout, the next state will be **starting**.

An issue arouse where the CAN bus object would stop it's connection with the CAN line. This issue occurred only when the emergency switch was pressed or the zero message was received. The root of this issue was never found.To fix it temporarily, the two states: **large CAN delay** and **long CAN wait** were added.

If the message time exceeds 10x the **MAX_COM_DELAY**, then the next state will be **large CAN delay**.

LARGE_CAN_DELAY

This is an in-between state that restarts the CAN bus and then sets the next state as **long CAN wait**. This state is needed because constantly trying to restart the CAN module results in communication issues.

LONG_CAN_WAIT

This state is identical to **short CAN wait**, but without the opportunity to go to **large CAN delay**.

The following two states are each evaluated independently from the others. This means that if the state is ever set to **starting** or **disabling** above, the following two states will process and set the next state without having to call `evaluateState()` again. Therefore, outside of `evaluateState()` the state should never be **starting** or **disabling**.

STARTING

This state starts both stepper objects as services if they are not already started. Then the next state is set as **running**.

DISABLING

This state calls `emergencyStop()` on both steppers and then determines which disabled state to go to.

`zero()`

The zeroing function actually consists of two functions, `zeroRotation()` and `zeroTranslation()`. This is because while both are zeroing stepper motors, they each have different needs.

In `zero()`, `zeroTranslation()` is called first. If it fails, false is returned. If it succeeds, `zeroRotation()` is called. If it fails false is returned. If both succeed, true is returned.

Note: the steppers as a service are stopped in the **zero** state to allow for *ESP-Flexystepper's* `processMovement()` to work.

`zeroRotation()`

To zero rotation, first the current location is set at home and stopped. Then the speed is set to the constant `HOME_SPEED_ROTATION` and the target set to three rotations in the direction from the direction array.

Due to the zero button having a large range that it can be depressed, the motor must first rotate until the button is released and then fully rotate until the button is depressed again. This is done in to while loops.

The first uses `!rotation_stepper.processMovement() && rotational_zero.sensorActive()` and the second uses `!rotation_stepper.processMovement() && !rotational_zero.sensorActive()`.

To sense the button press, the `Sensor_Debounce` object associated with the zero pin is petted every millisecond. If the button is pressed then 10 pets will occur and then `sensorActive()` will return true or false if the button is released. If the emergency switch is ever pressed, false is returned.

Note: the main need for the button debounce is that the button wires run next to the stepper motor wires which induce a current which will trigger a pressed if not debounced.

`zeroTranslation()`

The translation zeroing works the same way as the rotation with three major exceptions. First, if the button is pressed, it will not try to release the button. The second is that once the button is pressed, the motor moves one millimeter away to relieve pressure on the rod's rubber stopper. The last one being, the target distance is set at the max displacement for the rod.

`setControl()`

This function first checks if the state is the **running** state. If it is not the **running** state then nothing happens. But if it is the **running** `translation_desired` is truncated to be between zero and the max translation . If the value of the current position is over 1mm negative, then the next state is set as the **zero** state. Otherwise, the direction from the direction array is multiplied by the desired positions and then set as the target for rotation and translation steppers.

CAN communication

CAN (Closed Area Network) is a wire communication protocol originally designed for communication between many embedded micro controllers in the automotive industry. CAN allows for multiple devices in parallel to communicate to each other over two wires at a max speed of 1Mbps. This allows for cross communication between multiple devices without a need for a routing device and without the need for excess wires.

CAN functions on the idea that each device talks at the same time unless a more important device talks over it.To achieve this all devices send at the same time but if a device tries to send a **1** while another device is sending a **0**, the device sending **1** stops sending and instead listens. This means that a **ID** of `0b00` has a higher priority than an **ID** of `0b10`. With this in mind, all the **IDs** have been chosen following this priority hierarchy.

Messages from the *AI* will always have the highest priority so the first bit of every *AI* message *ID* will always be **0**. The second bit will indicate if the message is a **stop** command or not. Because **stop** commands are the most important, a stop command will be represented by a **0** as the second bit. The following four bits indicate which *controller* the message is from or going to.

The ID will only utilize the final 8-bits of the IDs 11-bits. The message data length is always 8 bytes. All bit data is using **Big-endian** meaning the bits on the left will be sent or received before the bits on the right.

ID: 8 bits

Bits	**0-1**	**2-3**	**4**	**5**	**6**	**7**
0	(00) message from *AI* : (01) Zero	(00) stop\stopped message : (01)	**NOT**	**NOT** 2 rod	**NOT** 5 rod	**NOT** 3 rod
1	command (11) message from player poles : (10) message from controllers	**NON** stop\stopped message (11) update goal counter : (10) reset goal counter	goal rod goal rod	2 rod 5 rod 3 rod		

NOTE: Due to the PDP using high priority messages such as `0xA`, `0xC` and `0x2`, the stop command must be sent to all rods and not individual rods. Therefore, the only valid stop command is `0xF`

DATA: 8 bytes

Bytes	**0-3**	**4-8**

ID bit 2:3 = 01 Displacement Data Angular Data

ID bit 2:3 = 11 Player goal count AI goal count

ID	**Description**
`0b000XXXX1`	Message from *AI* to *controller 3 rod* **1**
`0b000XXX1X`	Message from *AI* to *controller 5 rod* **2**

0b000XX1XX	Message from *AI* to *controller 2 rod* **3**
0b000X1XXX	Message from *AI* to *controller goal rod* **4**
0b010XXXX1	Zero Message to *controller 3 rod* **1**

ID	Description
0b010XXX1X	Zero Message to *controller 5 rod* **2**
0b010XX1XX	Zero Message to *controller 2 rod* **3**
0b010X1XXX	Zero Message to *controller goal rod* **4**
0b100XXXX1	Message from *controller 3 rod* to *AI*
0b100XX1X	Message from *controller 5 rod* to *AI*
0b100XX1XX	Message from *controller 2 rod* to *AI*
0b100X1XXX	Message from *controller goal rod* **4** to *AI*
0b110XXXX1	Message from *player 3 rod* **1** to *AI*
0b110XX1X	Message from *player 5 rod* **2** to *AI*
0b110XX1XX	Message from *player 2 rod* **3** to *AI*
0b110X1XXX	Message from *player goal rod* **4** to *AI*

0b1011XXXX Message from *goal controller* to *AI* to update goal count

0b0010XXXX Message from *AI* to *goal controller* to reset goal count

ID	Displacement Data Length	Rotational Data Length	Description

0b0X00XXXX	**X**	**X**	**Stop** command from *AI*
0b1000XXXX	**X**	**X**	**Stopped** message from *controller*
0b1100XXXX	**X**	**X**	**Stopped** message from *player*
0b0001XXXX	**4 Bytes**	**4 Bytes**	**Move** command from *AI*
0b1001XXXX	**4 Bytes**	**4 Bytes**	**Location** message from *controller*
0b1101XXXX	**4 Bytes**	**4 Bytes**	**Location** message from *player*

`CANSender()`

This function takes the current location of the stepper motors (rotations or mm), converts rotations into degrees . Then the value is written into the float part of a byte-float union. Then the bytes from the byte-float union are sent with a running or not running ID depending on state. Everything is sent **Big-Endian**.

`CANReceiver()`

This function checks to see if a message is in the buffer and then processes the message. If there is no new message, nothing happens.

The first thing done is the `receive_time` variable is set to the current time. Then the packet ID is checked to see if the message is an emergency stop, zero or movement command. If the message is either of the first two, `evaluateState()` is called to enact the message command after the pertinent variables are changed.

If the message is a movement command, the message is read into a bytes of a byte-float union using **Big-Endian**. Then the float is read and assigned to the two **desired** variables to be used in `setControl()`.

Controller_Constants.h

This header file contains all constants for **Controller.ino**. These constants are all global to allow for easy changes to the header file to change **Controller.ino**.

All constants are explicitly named or labeled. Each constant was determined through calculations or extensive testing.

The constants should only be changed if the mechanical components of the system change, the motors are replaced or the PCB is replaced.

Sensor_Debounce.h

This header file contains an object that can be used to do debounce on buttons or other sensors connected to a digital pin.

This object does not use interrupts (adding interrupts could be a part of future development). Instead, it uses pin petting.

Constructor

The constructor takes in four values:

1. Pin number
2. Pet Count
   - This is the amount of consecutive pets that would result in the sensor being active.
3. Pin Mode
   - Same as in `pinMode()`
4. Pressed value
   - The value that is considered pressed. Either **HIGH** or **LOW**

`readSensor()`

This function is used to read if the sensor is active and if it has been read. This is used in functionality such as pressing a button and waiting for it to be released before it can be pressed again.

`sensorActive()`

This function returns **true** if the sensor is active and **false** if the sensor is inactive.

`sensorMonitor()`

This is the petting function. Whenever possible call this function. Whenever this function is called the **Pet Count** amount of times with the pin reading a consistent value, the internal `pressed` variable will be set as true or false, depending on the value on the pin.

18.9 Goal Detection README

Goal_Detection.ino

This script is separated into two tasks: CAN communication and goal sensing. These tasks are ran on two separate cores so that there is no delay in the goal sensor or in communications.

The goal sensor core is an infinite loop that monitors both goal sensors using the `Sensor_Debounce` object and then checks if either sensor is active. If a sensor is active, the goal count corresponding to that sensor is increased by one.

The communication core is also an infinite loop that checks if a CAN message has been received and sends the current score every 50ms. The current score has to be converted into bytes using long a long-byte union. The only message that is looked for is the **Goal Reset** message. If this message is received, both goal counts are set to zero.

Sensor_Debounce.h

This header file contains an object that can be used to do debounce on buttons or other sensors connected to a digital pin.

This object does not use interrupts (adding interrupts could be a part of future development). Instead, it uses pin petting.

Constructor

The constructor takes in four values:

1. Pin number

2. Pet Count

◦ This is the amount of consecutive pets that would result in the sensor being active.

3. Pin Mode

◦ Same as in `pinMode()`

4. Pressed value

◦ The value that is considered pressed. Either **HIGH** or **LOW**

`readSensor()`

This function is used to read if the sensor is active and if it has been read. This is used in functionality such as pressing a button and waiting for it to be released before it can be pressed again.

`sensorActive()`

This function returns **true** if the sensor is active and **false** if the sensor is inactive.

`sensorMonitor()`

This is the petting function. Whenever possible call this function. Whenever this function is called the **Pet Count** amount of times with the pin reading a consistent value, the internal `pressed` variable will be set as true or false, depending on the value on the pin.

19 Appendix C: Complete Purchase Order

Ordering Sheet

All orders must be tax exempt!

Budget: 10000 < Enter amount given by sponsor

Spent: $ 7,856.48

Instructions

1) Identify all parts, provide a detailed description, and their respective quantity required. For example, you may need to indicate size and color in addition to the part name.

2) Identify a vendor (Note. Shipping may be cheaper if you group items to purchase.) 0

*****Notice: Post a link for the item you wish to purchase. Proceed all the way through the purchase process to view the estimated delivery date. If the projected delivery date is too long, you may not receive your parts in time to complete your project.**
The links you provide for each item will be used for purchase and it is assumed by the buyer that you have confirmed delivery date is acceptable!

3) Include Grainger into your search for potential vendors (OSU gets free shipping and more options are available for placing orders).

4) Enter items into spreadsheet below. Ask your Mentor to approve items by initialing in Column J. Notify your purchasing agent when you are ready to purchase items and have secured faculty mentor approval.

Balance (Remaining Funds): $2,143.52

Date Requested	Supplier Name	Part #	Link to part	Comments/Additional Instructions	Quantity	Cost/Unit, Excluding Shipping	Cost for all Units	Shipping charges	Team Purchaser Approval	Instructor /Mentor Approval	Order placed by	Supplier Order Number	ETA	Shipping Company	Tracking Number	Date Received
4-Feb	Amazon	Xunba DC 24V 50A 1200w Power Adapter Driver Transformer 110V AC DC 24V Power Supply for Led Strip lamp CNC CCTV (SS1200-24)	https://www.amazon.com/gp/product/B07JW2XT85/ref=ppx_yo_dt_b_search_asin_title?ie=UTF8&psc=1	0			$ 107.98	$ -	st	jpv	Conner					x
4-Feb	Amazon	NVIDIA Jetson Xavier NX Developer Kit (812674024318)	https://www.amazon.com/NVIDIA-Jetson-Xavier-Developer-Kit/dp/B086874Q5R/		1	$ 1,979.42	$1,979.42	$ -	st	jpc	Conner		Feb. 9 - Feb. 11			x
4-Feb	CanaKit	PI4-8GB-EXT128EW-C8-BLK-1	https://www.amazon.com/CanaKit-Raspberry-Pi-Extreme-Kit/dp/B08B6F8QRV/ref=sr_1_5?crid=3RKB5CW9E27F6&keywords=pi+4+8gb+extreme+kit+-+128gb&qid=1643991483&sprefix=pi+4+8gb+extreme+kit+-+128gb%2Caps%2C501&sr=8-5	The Pi4 8GB Extreme Kit - 128GB (has the best lead times)	1	$ 169.95	$ 169.95	$ 12.95	st	jpc	Conner		Feb. 11, 2022 - Feb. 14, 2022			x
14-Feb	Newark	SN65HVD232DR	https://www.newark.com/texas-instruments/sn65hvd232dr/can-transceiver-1mbps-soic-8/dp/17AH9318?gclid=Cj0KCQiAmKiQBhClARIsAKtSj-lv8Bkxt5W0NvaFpoDaunleCO481iHVNXpfk_R56Algd1fjVmGM3A8aAr16EALw_wcB&mckv=s_dc%7Cpcrid%7C487913213211%7Cplid%7C%7Ckword%7Csn65hvd232dr%7Cmatch%7Cp%7Cslid%7C%7Cproduct%7C%7Cpgrid%7C116333079515%7Cptaid%7Ckwd-307273936525%7C&CMP=KNC-GUSA-GEN-SKU-Semiconductors-Semiconductors-ICs&cmp=KNC-GUSA-GEN-SKU-Semiconductors-Semiconductors-ICs		15	$ 3.89	$ 58.35	$ 9.99	st	jpc	conner		Ordered on Feb 14 Shipping 2 to 5 days			
4-Feb	DigiKey	182-809ME-ND	https://www.digikey.com/en/products/detail/norcomp-inc./182-009-113R181/858344?utm_adgroup=Connectors%2C%20Interconnects&utm_source=google&utm_medium=cpc&utm_campaign=Shopping_Supplier_NorComp_0037_Co-Op&utm_term=&utm_content=Connectors%2C%20Interconnects&gclid=Cj0KCQiAuvOPBhDXARIsAKzLQ8HsrYEzSZa1b4jF6HqKyF3W57cl3yrZHVsjlfda3cNGQToZMTAU4yAaAiwfEALw_wcB		10	$ 2.39	$ 23.90	$ 6.99	st	jpc	Conner					x

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
4-Feb	Amazon	Full HD 1080P VGA Splitter Cable (VGA Y Cable) for Screen Duplication - Does NOT Show Separate Displays (No Screen Extension)	https://www.amazon.com/Cable-Matters-Splitter-Screen-Duplication/dp/B005H3I38G/ref=asc_df_B005H3I38G/?tag=hyprod-20&linkCode=df0&hvadid=309833041189&hvpos=&hvnetw=g&hvrand=2268592760927478405&hvpone=&hvptwo=&hvqmt=&hvdev=m&hvdvcmdl=&hvlocint=&hvlocphy=1024339&hvtargid=pla-594027510474&th=1		6	$ 7.49	$ 44.94	$ -	st	jpc	Conner					x
4-Feb	Amazon	Teyleten Robot ESP32S ESP32 ESP-WROOM-32 Development Board 2.4GHz Dual-Core WiFi +Bluetooth 2 Function Microcontroller for Arduino (ESP32 30P, 3PCS)	32-Development-Microcontroller/dp/B08246MCL5/ref=sr_1_1_sspa?crid=1BUSODPERPLM9&keywords=esp32+development+board&qid=1643992685&s=electronics&sprefix=esp32%2Celectronics%2C91&sr=1-1-spons&psc=1&spla=ZW5jcnlwdGVkUXVhbGlmaWVyPUFSMjQ1T05MSjg0WEgmZW5jcnlwdGVkSWQ9QTA1OTM4NjUzQ0kyVEpOUzE2N1ZlUmVuY3J5cHRlZEFkSWQ9QTEwMDcwMDkyMURHQURRWjcwM1ZUJndpZGdldE5hbWU9c3BfYXRmJmFjdGlvbj1jbGlja1JlZGlyZWN0JmRvTm90TG9nQ2xpY2s9dHJ1ZQ==	It's a pack of 3 (we need six) this is reflected in the quantity. We want to use the 30pin version.	2	$ 7.30	$ 14.60	$ -	st	jpc	Conner					x
4-Feb	Amazon	ANMBEST VGA Solderless Quick Connector 16mm Thinner 3+9 DB15 3Row Serial Plug to 15-pin Port Terminal Adapter Breakout Board Compatible with VGA 3+7/3+6/3+6/3+4/3+2 (Male+Female)	https://www.amazon.com/ANMBEST-Solderless-Connector-Terminal-Compatible/dp/B08HSX43NJ/ref=mp_s_a_1_3?crid=O5JZM1SS5ONO&keywords=vga%2Bconnector&qid=1643993052&sprefix=vga%2Bconn%2Caps%2C178&sr=8-3&th=1	Size: Metel Female	1	$ 13.99	$ 13.99	$ -	st	jpc	Conner					x
5-Feb	Amazon	USB to CAN Converter Module for Raspberry Pi4/Pi3B+/Pi3/Pi Zero(W)/Jetson Nano/Tinker Board and Any Single Board Computer Support Windows Linux and Mac OS	https://www.amazon.com/Innomaker-Converter-Module-Raspberry-Zero/dp/B07P9JGXXB/ref=asc_df_B07P9JGXXB/?tag=&linkCode=df0&hvadid=416652333997&hvpos=&hvnetw=g&hvrand=15110021240692036544&hvpone=&hvptwo=&hvqmt=&hvdev=c&hvdvcmdl=&hvlocint=&hvlocphy=9026533&hvtargid=pla-884130091857&ref=&adgrpid=100759324064&th=1	Size: USB2CAN	1	$ 32.99	$ 32.99	$ -	st	jpc	Conner					x
13-Feb	Amazon	HMTECH Raspberry Pi Screen 10.1 Inch Touchscreen Monitor 1024x600 Portable HDMI Monitor 16:9 IPS Screen Display for Raspberry Pi 4/3/2/Zero/B/B+ Win10/8/7, Free-Driver	https://www.amazon.com/HMTECH-Raspberry-Touchscreen-1024x600-Portable/dp/B0987468N2/ref=sr_1_10?keywords=touchscreen+monitor+raspberry+pi&qid=1644635390&s=electronics&sprefix=Touchscreen+Monitor+ras%2Celectronics%2C94&sr=1-10		1	$ 109.99	$ 109.99	$ -	st	jpc	Conner		Feb. 15			x
10-Feb	Amazon	SanDisk 64GB Extreme PRO SDXC UHS-I Card - C10, U3, V30, 4K UHD, SD Card - SDSDXXY-064G-GN4IN	https://www.amazon.com/SanDisk-128GB-Extreme-UHS-I-SDSDXXY-128G-GN4IN/dp/B07H9J1YXN/ref=sr_1_4?crid=1RQOFE3FESKXR&keywords=sd%2Bcards&qid=1644527588&sprefix=sd%2Bcards%2Caps%2C96&sr=8-4&th=1	Size: 64GB	3	$ 17.49	$ 52.47	$ -	st	jpc	Conner					x

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
10-Feb	Amazon	SDSDXXY-064G-GN4IN	-SDSDXXY-128G-GN4IN/dp/B07H9J1YXN/ref=sr_1_4?crid=1RQOFE3FESKXR&keywords=sd%2Bcards&qid=1644527588&sprefix=sd%2Bcards%2Caps%2C96&sr=8-4&th=1	Size: 64GB	3	$ 17.49	$ 52.47	$ -	st	jpc	Conner					x
17-Feb	VexRobotics	Power Distribution Panel	https://www.vexrobotics.com/217-4244.html		1	$ 204.99	$ 204.99	$ 16.46	st	jpc	Cpnner					x
17-Feb	DigiKey	1725698 TERM BLK 6P SIDE ENT 2.54MM PCB	&gclid=CjwKCAiAgbiQBhAHEiwAuQ6BkupR5DZsyixK7tIQHJ-_eoiCc-IfvqSBwmf5nFyIND5j0Mlwa5kz6RoCDq8QAvD_BwE		10	$ 5.04	$ 50.38	$ 6.99	st	jpc	conner					x
17-Feb	DigiKey	1758364 TERM BLOCK PLUG 2POS STR 5.08MM	https://www.digikey.com/en/products/detail/phoenix-contact/1758364/1926180?utm_adgroup=Terminal%20Blocks%20-%20Headers%2C%20Plugs%20and%20Sockets&utm_source=google&utm_medium=cpc&utm_campaign=Shopping_Product_Connectors%2C%20Interconnects_NEW&utm_term=&utm_content=Terminal%20Blocks%20-%20Headers%2C%20Plugs%20and%20Sockets&gclid=CjwKCAiAgbiQBhAHEiwAuQ6BkrOlu178kR5tAlodb59HIJbFXLluEV43CEn_Eki7AhNnw6R_x7LDGhoCO-UQAvD_BwE		10	$ 2.31	$ 23.10		st	jpc	conner					x
17-Feb	JLCPCB	Contoller PCB	https://jlcpcb.com/	gbr files are located in Electrical->files->controls->Controller_PCB_gbr.zip , all defaults for settings will work	10	$ 0.50	$ 5.00	$ 19.80	st	jpc	cpnner					x
22-Feb	Amazon	30cm DB9 Y Splitter Cable DB9 9 Pin 1 Male to 2 Female Rs232 Serial Splitter Adapter Straight-Through Cable YOUCHENG for Connect Various Serial Interface Devices	https://www.amazon.com/Splitter-Adapter-Straight-Through-YOUCHENG-Interface/dp/B08P8G9R6D/ref=pd_bxgy_img_1/132-2036138-4303850?pd_rd_w=rOVMM&pf_rd_p=6b3eefea-7b16-43e9-bc45-2e332cbf99da&pf_rd_r=GYR87GY03SCZENHZRJPH&pd_rd_r=301d5e07-22a5-4222-86d2-97bf926ca010&pd_rd_wg=7bLDH&pd_rd_i=B08P8G9R6D&psc=1		8	$ 7.99	$ 63.92		st	jpc	cpnner		Feb 28th			x
2/28/2022	Amazon	Regulation Size 1.365" Table Soccer Foosball Table Replacement Balls: Official Tournament Table Soccer Balls	https://www.amazon.com/Regulation-Foosballs-Replacement-Officia-Tournament/dp/B08Q7VK37M/ref=sr_1_3?crid=3880W0X3NW0SU&keywords=foosballs%2Bgreen&qid=1646057059&sprefix=foosballs%2Bgreen%2Caps%2C86&sr=8-3&th=1	Color: Green!, we're getting two for plenty of backups. Multi pack was in stock. Will contain 3 green balls	2	$ 6.99	$ 13.98		st	jpc	Conner	Ordered multipack which contains 3 green balls.				
2/28/2022	Amazon	Welugnal 50 Amp Automotive Marine Circuit Breaker with Manual Reset Surface-Mount for Trolling Boat Motor Battery Thermal 12V-48V DC Waterproof	https://www.amazon.com/Welugnal-Automotive-Surface-Mount-Trolling-Waterproof/dp/B07J5FBLKM/ref=asc_df_B07J5FBLKM/?tag=hyprod-20&linkCode=df0&hvadid=309778489815&hvpos=&hvnetw=g&hvrand=14775880494838494530&hvpone=&hvptwo=&hvqmt=&hvdev=m&hvdvcmdl=&hvlocint=&hvlocphy=9026533&hvtargid=pla-637601781081&psc=1	Size: 50 Amp	1	$ 13.99	$ 13.99		st	jpc	Conner		Mar. 2, 2022			
2/28/2022	Mcmaster-Carr	47065T101	https://www.mcmaster.com/47065T101/	T-Slot Framing 1inch hollow 4 slot rail silver, **3ft**	5	$ 23.92	$ 119.60		st	jpc	LES		Mar. 7, 2022			
2/282022	Mcmaster-Carr	47065T101	https://www.mcmaster.com/47065T101-47065T121/	T-Slot Framing 1inch hollow 4 slot rail silver, **4ft**	6	$ 28.10	$ 168.60		st	jpc	LES		Mar. 7, 2022	FedEx	566557696047	7-Mar
2/28/2022	Mcmaster-Carr	47065T239	https://www.mcmaster.com/47065T239/	1inch extended corner bracket for single rails	12	$ 7.07	$ 84.84		st	jpc	LES		Mar. 7, 2022			
2/28/2022	Mcmaster-Carr	47065T259	https://www.mcmaster.com/47065T259/	Silver Surface Bracket, 4" Long for 1" High Single Rail	4	$ 8.16	$ 32.64	$ 98.85	st	jpc	LES	5348425	Mar. 7, 2022			
2/28/2022	Mcmaster-Carr	47065T236	https://www.mcmaster.com/47065T236/	Silver Corner Bracket, 1" Long for 1" High Rail T-Slotted Framing	20	$ 7.01	$ 140.20		st	jpc	LES		Mar. 7, 2022			

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
2/28/2022	Mcmaster-Carr	47065T259		for 1" High Single Rail	4	$ 8.16	$ 32.64		st	jpc	LES		Mar. 7, 2022			
2/28/2022	Mcmaster-Carr	47065T236	https://www.mcmaster.com/47065T236/	Silver Corner Bracket, 1" Long for 1" High Rail T-Slotted Framing	20	$ 7.01	$ 140.20		st	jpc	LES		Mar. 7, 2022			
2/28/2022	Mcmaster-Carr	47065T267	https://www.mcmaster.com/47065T267/	Silver Corner Surface Bracket for 1" High Single Rail	8	$ 10.26	$ 82.08		st	jpc	LES		Mar. 7, 2022	FedEx	56655769 6036	7-Mar
2/28/2022	Mcmaster-Carr	8701K53	https://www.mcmaster.com/8701K53/	Slippery UHMW Polyethylene Rod: raw spline material: **2ft**	6	$ 38.80	$ 232.80		st	jpc	LES		Mar. 7, 2022	FedEx	56655769 6025	7-Mar
2/28/2022	Amazon	23HS45-4204S	https://www.amazon.com/Torque-Stepper-Motor-425oz-Router/dp/B00PNEPW4C/ref=pd_lpo_1?pd_rd_i=B00PNEPW4C&psc=1	23HS45-4204S : HIGH TORQUE NEMA 23	4	$ 39.99	$ 159.96		st	jpc	Conner		Mar. 2, 2022			3-Mar
2/28/2022	Amazon	DM542T	https://www.amazon.com/STEPPERONLINE-1-0-4-2A-20-50VDC-Micro-step-Resolutions/dp/B06Y5VPSFN/ref=asc_df_B06Y5VPSFN/?tag=hyprod-20&linkCode=df0&hvadid=242022044358&hvpos=&hvnetw=g&hvrand=12363182901297871867&hvpone=&hvptwo=&hvqmt=&hvdev=c&hvdvcmdl=&hvlocint=&hvlocphy=9026533&hvtargid=pla-588890261232&psc=1	NEMA 23 DRIVERS	4	$ 28.99	$ 115.96		st	jpc	Conner		Mar. 2, 2022			3-Mar
2/28/2022	Mcmaster-Carr	9056K68	https://www.mcmaster.com/9056K68-9056K681/	New Aluminium rods **6ft**	5	$ 42.50	$ 212.50		st	jpc	LES	5348425	Mar. 7, 2022	FedEx	56655770 6567	7-Mar
2/28/2022	Mcmaster-Carr	5913K62	https://www.mcmaster.com/5913K62/	Low-Profile Mounted Sealed Steel Ball Bearing	8	$ 10.95	$ 87.60		st	jpc	LES		Mar. 7, 2022	FedEx	56655769 6036	7-Mar
2/28/2022	Pololu	2258	https://www.pololu.com/product/2258	Steel L-Bracket for NEMA 23 Stepper Motors	4	$ 4.95	$ 19.80	$ 12.45	st	jpc	Cpnner					
2/28/2022	Openbuilds	SKU: 2680-Bundle	https://openbuildspartstore.com/v-slot-nema-23-linear-actuator-belt-driven/	V-Slot® NEMA 23 Linear Actuator (Belt Driven) Bundle **500mm**	4	$ 121.99	$ 487.96	$ 15.48	st	jpc	Conner					
2/28/2022	Mcmaster-Carr	7959K25	https://www.mcmaster.com/7959K25/	L Series Timing Belt, Trade No 367L050 **5ft**	4	$ 20.85	$ 83.40		st	jpc	LES		Mar. 7, 2022	FedEx	56655769 6036	7-Mar
2/28/2022	Mcmaster-Carr	6527K96	https://www.mcmaster.com/6527K96/	Low-Carbon Steel Rectangular Tube 0.083" Wall Thickness 2-1/2" x 2-1/2" Outside Size **3ft**	5	$ 35.11	$ 175.55		st	jpc	LES	5348425	Mar. 7, 2022	FedEx	56655769 6058	7-Mar
2/28/2022	Mcmaster-Carr	47065T915	https://www.mcmaster.com/47065T915/	Friction Mount Monitor Mount	1	$ 85.06	$ 85.06		st	jpc	LES		Mar. 7, 2022			
2/28/2022	Mcmaster-Carr	47065T139	https://www.mcmaster.com/47065T139/	End-Feed Single Nut with Button Head 1/4"-20 Thread Size	2	$ 2.40	$ 4.80		st	jpc	LES		Mar. 7, 2022			
2/28/2022	mcmaster-Carr	6495K747	https://www.mcmaster.com/6495K747/	Timing Belt Pulley XL Series, Black-Oxide Steel	4	$ 20.48	$ 81.92		st	jpc	LES		Mar. 7, 2022	FedEx	56655769 6036	7-Mar
2/28/2022	Lowes	Item #130743 Model #720971-10	https://www.lowes.com/pd/Top-Choice-2-in-x-4-in-x-10-ft-Douglas-Fir-Lumber-Common-1-5-in-x-3-5-in-x-10-ft-Actual/1000463565	2-in x 4-in x 10-ft Top Choice Douglas Fir Lumber	4	$ 11.72	$ 46.88		st	jpc						3/1/2022
2/28/2022	lowes	130744	https://www.lowes.com/pd/Top-Choice-2-in-x-4-in-x-12-ft-Douglas-Fir-Lumber-Common-1-5-in-x-3-5-in-x-12-ft-Actual/1000463569	2-in x 4-in x 12-ft Douglas Fir Lumber	4	$ 13.95	$ 55.80		st							3/1/2022
2/28/2022	Lowes	Item #786164	https://www.lowes.com/pd/SoyStrong-NAF-3-4-in-HPVA-Oak-Plywood-Application-as-4-x-8/1000096488	3/4-in x 4-ft x 8-ft Oak Sanded Plywood	1	$ 75.98	$ 75.98		st	jpc						3/1/2022
2/28/2022	Lowes	Item #195323	https://www.lowes.com/pd/Severe-Weather-15-32-CAT-PS1-09-Square-Structural-Plywood-Pressure-Treated-Southern-Yellow-Pine-Application-as-4-x-8/4564604	1/2-in x 4-ft x 8-ft Poplar Sanded Plywood	2	$ 53.88	$ 107.76		st	jpc						3/1/2022
2/28/2022	Lowes	item#580316 model#48307	https://www.lowes.com/pd/Power-Pro-10-x-2-1-2-in-Ceramic-Deck-Screws-240-Count/1000363881	Power Pro #10 x 2-1/2-in Bronze Epoxy Flat Exterior Wood Screws	1	$ 32.98	$ 32.98		st	jpc						3/1/2022
2/28/2022	Lowes	item#580298 model#48316	https://www.lowes.com/pd/Power-Pro-8-x-1-5-8-in-Ceramic-Deck-Screws-175-Count/50406010	Power Pro #8 x 1-5/8-in Bronze Epoxy Trim Exterior Wood Screws	1	$ 10.98	$ 10.98		st	jpc						3/1/2022
2/28/2022	Mcmaster-Carr	5071A51	https://www.mcmaster.com/5071A51/	Latch-Style Toggle Clamp	4	$ 14.99	$ 59.96		st	jpc	LES		Mar. 7, 2022			
2/28/2022	Mcmaster-Carr	1798A21	https://www.mcmaster.com/1798A21/	hinges	2	$ 9.07	$ 18.14		st	jpc	LES		Mar. 7, 2022			
2/28/2022	Mcmaster-Carr	5186A6	https://www.mcmaster.com/5186A6/	door handle	1	$ 36.05	$ 36.05		st	jpc	LES		Mar. 7, 2022			
				Lid Support								5348425				

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
2/28/2022	Mcmaster-Carr	5186A6	https://www.mcmaster.com/5186A6/	door handle	1	$ 36.05	$ 36.05		st	jpc	LES		Mar. 7, 2022			
2/28/2022	Mcmaster-Carr	1227A43	https://www.mcmaster.com/1227A43/	Lid Support Soft Opening, Left Side Mount	1	$ 21.90	$ 21.90		st	jpc	LES	5348425	Mar. 7, 2022			
2/28/2022	Mcmaster-Carr	1227A42	https://www.mcmaster.com/1227A42/	Lid Support Soft Opening, Right Side Mount	1	$ 21.90	$ 21.90		st	jpc	LES		Mar. 7, 2022	FedEx	56655769 6036	7-Mar
2/28/2022	Lowes	Item #233959	https://www.lowes.com/pd/Severe-Weather-15-32-CAT-PS1-09-Square-Structural-Plywood-Pressure-Treated-Southern-Yellow-Pine-Application-as-4-x-8/4564604	Style Selections Matte Cabinet Latch	4	$ 1.78	$ 7.12		st	jpc						3/1/2022
2/28/2022	Amazon	ACFAN00136A	https://www.amazon.com/ARCTIC-P14-Value-Pack-Pressure-optimised/dp/B07HC9ZWDN/ref=sr_1_3?crid=2GXHHPFWSKM64&keywords=140mm+case+fan&qid=1645564733&s=industrial&sprefix=140mm+case+fan%2Cindustrial%2C84&sr=1-3	ARCTIC P14 (5 Pack) - 140 mm Case Fan, Value Pack, Pressure-optimised, Very quiet motor, Computer, Fan Speed: 1700 RPM - Black	2	$ 32.00	$ 64.00		st	jpc	Conner		Mar. 2, 2022			3-Mar
2/25/2022	Amazon	Item added by Conner	https://www.amazon.com/gp/product/B01EF9OI0O/ref=ox_sc_act_title_2?smid=A28RI4FXFS5SV8&psc=1	JBtek All Black Sleeved PWM Fan Splitter Cable 1 to 2 Converter, 2 Pack	3	$ 5.99	$ 17.97		st	jpc	Conner		Mar. 2, 2022			3-Mar
3/1/2022	mcmaster-Carr	4768K5	https://www.mcmaster.com/4768K5/	set screw bearing	8	$ 48.60	$ 388.80		st		LES	5348425	Mar. 7, 2022	FedEx	56655769 6036	7-Mar
3/1/2022	DigiKey	EG5617-ND	https://www.digikey.com/en/products/detail/e-switch/RA11131121/2720267	RA11131121	1	0.67	$ 0.67		st	jpc	Conner		I have one like this in sotkc at NCL. Will bring them to ENDV 340			
3/1/2022	Amazon	Kohree DC/DC Converter Regulator 24V Step Down to 12V 20A 240W Golf Cart Low Voltage Reducer Buck Transformer Waterproof	https://www.amazon.com/Kohree-Converter-Regulator-Transformer-Waterproof/dp/B09GXXQZ6P/ref=sr_1_1_sspa?keywords=24V+to+12V+Converter&qid=1646180492&sr=8-1-spons&psc=1&spLa=ZW5jcnlwdGVkUXVhbGlmaWVyPUFOREJJVUJFQU8zMFgmZW5jcnlwdGVkSWQ9QTA0NjM1NTMyWExxTzBCNlpCRzgzJmVuY3J5cHRlZElkSWQ9QTA3NDMwOTVEWkdGU1lyWlQzQjYmd2lkZ2V0TmFtZT1zcF9hdGYmYWN0aW9uPWNsaWNrUmVkaXJlY3QmZG9Ob3RMb2dDbGljaz10cnVl		1	$ 19.99	$ 19.99		st	jpc	conner					
3/1/2022	Amazon	DC Voltage Regulator Buck Converter DC 12V 24V Step-Down to 5V 20A 100W Reducer stabilized Power Supply Converter Waterproof Module Transformer	https://www.amazon.com/Regulator-Converter-12V-Waterproof-Transformer/dp/B08CHMJM9J/ref=sr_1_12_sspa?crid=366U5H4OMKEBT&keywords=12v+to+5v+dc+step+down&qid=1646180834&sprefix=12v+to+5v+dc+step+down%2Caps%2C88&sr=8-12-spons&psc=1&spLa=ZW5jcnlwdGVkUXVhbGlmaWVyPUExMVAxTTJGODg4NkVOJmVuY3J5cHRlZElkPUEwNjc1NDQ3VDFQUDdaTVVZTzFPJmVuY3J5cHRlZEFkSWQ9QTA3NDAzNDYyMjAxOFw4N1k5TUh5IndpZGdldE5hbWU9c3BfbXRmJmFjdGlvbj1jbGlja1JlZGlyZWN0JmRvTm90TG9nQ2xpY2s9dHJ1ZQ==		1	$ 18.55	$ 18.55		st							
3/4/2022	Mouser	SN65HVD232DG4	https://www.mouser.com/ProductDetail/595-SN65HVD232DG4		15	2.55	$ 38.25		st	jpc	Conner		7-Mar		1Z77594 5030415 1168	
3/8/2022	Mcmaster-Carr	7779K52	https://www.mcmaster.com/7779K52/	Miniature Snap-Acting Switch	8	$ 3.79	$ 30.32	$ 13.40	st		LES		9-Mar	FedEx	56655786 6429	
3/8/2022	Mcmaster-Carr	9246K596	https://www.mcmaster.com/9246K596/	Multipurpose 6061 Aluminum	2	$ 70.81	$ 141.62		st		LES		9-Mar	FedEx	56655786 6429	
3/8/2022	AndyMark	20 Amp Snap Action Breaker	https://www.andymark.com/products/20-amp-snap-action-breaker	PDP 20A Fuses	10	$ 6.00	$ 60.00	$ 9.50	st	jpc	Cpnner					
3/8/2022	AndyMark	40 Amp Snap Action Breaker	https://www.andymark.com/products/40-amp-snap-action-breaker	PDP 40A Fuses	10	$ 7.00	$ 70.00		st	jpc	conner					

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
3/8/2022	AndyMark	40 Amp Snap Action Breaker	n-breaker	PDP 40A Fuses	10	$ 7.00	$ 70.00		st	jpc	conner					
3/9/2022	Amazon	Foosball table grommet	https://www.amazon.com/Piece-Tornado-Foosball-Bushing-Nut-Kit/dp/B00SBSQK8Y/ref=asc_df_B00SBSQK8Y/?tag=hyprod-20&linkCode=df0&hvadid=241885613712&hvpos=&hvnetw=g&hvrand=16004473932961696318&hvpone=&hvptwo=&hvqmt=&hvdev=c&hvdvcmdl=&hvlocint=&hvlocphy=1024339&hvtargid=pla-492257789180&psc=1	3 Piece Tornado Foosball Bushing-Nut Kit	4	$ 10.95	$ 43.80		st	jpc	conner		15-Mar			
3/9/2022	Amazon	Foosball table replacment men	https://www.amazon.com/Tornado-Foosball-Men-Counter-Balanced/dp/B00576JQFM	Tornado 4 Foosball Men Counter Balanced	3	$ 19.89	$ 59.67	$ 21.18	st	jpc	Conner		15-Mar			
3/9/2022	amazon	riccioofy	https://www.amazon.com/Mushroom-Emergency-Button-Switch-Station/dp/B083D1R3YB/ref=asc_df_B083D1R3YB/?tag=hyprod-20&linkCode=df0&hvadid=416668650463&hvpos=&hvnetw=g&hvrand=4541704561996410547&hvpone=&hvptwo=&hvqmt=&hvdev=c&hvdvcmdl=&hvlocint=&hvlocphy=1024339&hvtargid=pla-889658444303&psc=1&tag=&ref=&adgrpid=90730998421&hvpone=&hvptwo=&hvadid=416668650463&hvpos=&hvnetw=g&hvrand=4541704561996410547&hvqmt=&hvdev=c&hvdvcmdl=&hvlocint=&hvlocphy=1024339&hvtargid=pla-889658444303	Red Sign Mushroom Emergency Stop Push Button Switch Station 1 NO 1 NC 10A 660V	1	$ 9.99	$ 9.99		st	jpc	Cpnner		mar 16 to mar 18			
3/10/2022	Mcmaster-Carr	97171A160	https://www.mcmaster.com/97171A160/	Brass Heat-Set Inserts for Plastic	1	$ 16.06	$ 16.06	$ 18.32	st		LES	5749110	14-Mar	FedEx	541474187240	
	Mcmaster-Carr	92160A123	https://www.mcmaster.com/92160A123/	Installation Tip for 8-32/M4 x 0.7 mm Thread Size Heat-Set Inserts	1	$ 18.69	$ 18.69		st		LES			FedEx	566558031337	
3/11/2022	Mcmaster-Carr	5432T17	https://www.mcmaster.com/5432T17/	Wire connector	1	$ 7.36	$ 7.36	$ 9.32	st	jpc	LES	5873197	16-Mar	FedEx	566558135405	
3/11/2022	Showmecables	SKU: 3609	?ExFnnMXTiJkqppQY8BQHjMse09RRke2xoCpmIQAvD_BwE	Female HDMI to female HDMI panel mount	1	$ 8.96	$ 8.96		st	jpc	Conner					
3/11/2022	l-com	ECF504-3AA-S	https://www.l-com.com/usb-30-adapter-coupler-panel-m	Female USB to Female USB panel n	2	$ 14.85	$ 29.70		st	jpc	Conner					
3/11/2022	Mcmaster-Carr	6641T16	https://www.mcmaster.com/6641T16/	RCA Coxial cord	1	$ 6.17	$ 6.17		st	jpc	LES	5873197	16-Mar	FedEx	566558135405	
3/11/2022	DigiKey	SC1907-ND	oogle&utm_medium=cpc&utm_campaign=Shopping_Product_Connectors%2C%20Interconnects_NEW&utm_term=&utm_content=D-Sub%2C%20D-Shaped%20Connectors%20-%20Adapters&gclid=CjwKCAiA4KaRBhBdEiwAZi1zzu e9aTTgulCd5nAUmIFn9YVH5a1wgfWBrKXvcs9m4QMk6k 9Ggj4-3hoCpOcQAvD_BwE3hoCpOcQAvD_BwE	Female DB9 Panel Mount	1	$ 14.46	$ 14.46		st	jpc	Conner					
3/14/2022	Amazon	Ancable Replaceme	https://www.amazon.com/Ancable-Replacement-5-5x2-5mm-Connector-Shrinkle/dp/B07X4XLFQS/ref=sr_1_4?keywords=2.5mm%2B5.5mm%2Bbarrel%2Bplug&qid=164727 4930&sr=8-4&th=1	2.5x5.5 connector for jetson Changed to 10 pack as it was only $1 more	1	$ 7.99	$ 7.99		st	jpc	Friday, Mar. 25 Estimated delivery					
3/14/2022	Amazon	Terminal Blocks Gr	https://www.amazon.com/Terminal-Circuit-Positions-Insulated-Electrical/dp/B08P46WVJX/ref=sr_1_3?crid=3SHS7DTH0AQ3F&keywords=Electrical+Bus+12+position&qid=1647275789&sprefix=electrical+bus+12+position%2Caps%2C80&sr=8-3	12 pos for 12v and 10 or 8 pos for !	1	$ 12.99	$ 12.99		st	jpc	Friday, Mar. 25 Estimated delivery					

A	B	C	D	E	F	G	H	I	J	K	L	M	N
3/14/2022	Amazon	5pcs USB C Type-C	https://www.amazon.com/Type-C-Cable-10inch-22AWG-Pigtail/dp/B09C7SLHFP/ref=asc_df_B09C7SLHFP/?tag=hyprod-20&linkCode=df0&hvadid=546914260534&hvpos=&hvnetw=g&hvrand=14412584906472440498&hvpone=&hvptwo=&hvqmt=&hvdev=c&hvdvcmdl=&hvlocint=&hvlocphy=1024339&hvtargid=pla-1461994339116&psc=1		1	$ 12.98	$ 12.98		st	jpc	Friday, Mar. 25 Estimated delivery		
3/15/2022	Lowes	Item #824	https://www.lowes.com/pd/Hillman-1-4-in-20-x-3-in-Phillips-Slotted-Combination-Drive-Machine-Screws-25-Count/3044202	Hillman 1/4-in-20 x 3-in Phillips/Slotted Combination-Drive Machine Screws (25-Count)	1	$ 5.98	$ 5.98		st	jpc			
3/15/2022	Mcmaster-Carr	4634T39	https://www.mcmaster.com/4634T39-4634T19/	Multipurpose 6061 Aluminum 15 mm Diameter	5	$ 26.40	$ 132.00	$ 26.66	st		LES	6317119	24-Mar
							$ -						
3/21/2022	Open Builds	Nylon Insert Hex Lc	https://openbuildspartstore.com/nylon-insert-hex-locknut-m5-10-pack/		10	$ 0.99	$ 9.90		st	jpc			
3/21/2022	Open Builds	Low Profile Screws	https://openbuildspartstore.com/low-profile-screws-m5-1	**Size: 8mm**	10	$ 0.99	$ 9.90		st	jpc			
3/21/2022	Open Builds	Low Profile Screws	https://openbuildspartstore.com/low-profile-screws-m5-1	**Size: 27mm**	10	$ 1.39	$ 13.90		st	jpc			
3/21/2022	Open Builds	Double L Bracket	https://openbuildspartstore.com/l-bracket/	Size: Double	18	$ 1.49	$ 26.82		st	jpc			
3/21/2022	Open Builds	Double Tee Nut	https://openbuildspartstore.com/double-tee-nut/		18	$ 0.69	$ 12.42		st	jpc			
3/23/2022	Amazon	8 ft HDMI	https://www.amazon.com/Cable-Support-1080p-Ethernet-Return%EF%BC%88ARC%EF%BC%89/dp/B07RYSNHGP/ref=sr_1_3?keywords=8ft+hdmi+cable&qid=1648059875&sr=8-3	picked up the two pack which was	1	$ 12.99	$ 12.99		st	jpc	Friday, Mar. 25		
3/23/2022	Amazon	6 ft HDMI	https://www.amazon.com/Cable-Capshi-Speed-18Gbps-Ethernet/dp/B07FFS7RH1/ref=sr_1_2_sspa?keywords=6ft+hdmi+cable&qid=1648059710&sr=8-2-spons&psc=1&spLa=ZW5jcnlwdGVkUXVhbGlmaWVyPUE5N0hFUDc2SFhCVksmZW5jcnlwdGVkSWQ9QTA5NDA4MTQySjBNMUE0REhONVpDJmVuY3J5cHRlZEFkSWQ9QTA0NDczOTUzNVdYWlk2Q09aMF9CJndpZGdldE5hbWU9c3BfYXRmJmFjdGlvbj1jbGlja1JlZGlyZWN0JmRvTm90TG9nQ2xpY2s9dHJ1ZQ==	Did not order as I ordered a 2 pack of the 8 ft HDMI for only 12.99						jpc			
3/23/2022	Amazon	8 ft USB M-F	https://www.amazon.com/Extension-Transfer-Extender-Printer-Keyboard/dp/B097R5166D/ref=sr_1_6?crid=1TW9H8EXMZQXC&keywords=male+to+female+usb+cable+8ft&qid=1648059891&sprefix=male+to+femal+usb+cable+8ft%2Caps%2C119&sr=8-6		1	$ 15.90	$ 15.90		st	jpc	Friday, Mar. 25		
3/23/2022	Amazon	6 ft USB M-M	https://www.amazon.com/AINOPE-Rupture-Double-Compatible-Enclosures/dp/B07YSNFNKD/ref=sr_1_3?crid=3HJ5CDJQZTFPB&keywords=Male+To+Male+Usb+Cable+6ft&qid=1648059714&sprefix=male+to+male+usb+cable+6ft%2Caps%2C159&sr=8-3		1	$ 11.99	$ 11.99		st	jpc	Friday, Mar. 25		
3/23/2022	amazon	Elite Core Panel-Mount Connector | 50-Pack Screw and Nut Kit for D-Series Panel	https://www.amazon.com/Elite-Core-Panel-Mount-Connector-50-Pack/dp/B01LANHHQU/ref=pd_day0fbt_img_2/141-1147947-2867823?pd_rd_w=FBTQg&pf_rd_p=bcb8482a-3db5-4b0b-9f15-b86e24acdb00&pf_rd_r=03WZS511N2Z34SMW6PBY&pd_rd_r=f3657a95-2565-477f-863f-4c7af331471c&pd_rd_wg=b78xg&pd_rd_i=B01LANHHQU&psc=1		2	$ 13.99	$ 27.98		st	jpc	Friday, Mar. 25		
3/23/2022	amazon	USB 3.0 Connector Dual USB Port Female Socket 86 Panel Mount Square Female to Female \	https://www.amazon.com/Connector-Converter-Connectors-Optoelectronic-Industrial/dp/B087MZZV8F/ref=sr_1_6?keywords=usb%2Bpanel%2Bmount&qid=1648062674&sr=8-6&th=1		1	$ 13.99	$ 13.99		st	jpc	Friday, Mar. 25		
			https://www.amazon.com/Adafruit-Accessories-Break-Be										

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
6	3/23/2022	amazon	Female Socket 86 Panel Mount Square Female to Female \	https://www.amazon.com/Connector-Converter-Connectors-Optoelectronic-Industrial/dp/B087MZZV8F/ref=sr_1_6?keywords=usb%2Bpanel%2Bmount&qid=1648062674&sr=8-6&th=1		1	$ 13.99	$ 13.99		st	jpc	Friday, Mar. 25			
7	3/24/2022	Amazon	Adafruit Accessorie	https://www.amazon.com/Adafruit-Accessories-Break-Beam-Sensor/dp/B01BU6YBWU		2	$ 10.15	$ 20.30			jpc	Monday, Mar. 28			
8	3/24/2022	Amazon	StarTech.com 0.5m	https://www.amazon.com/StarTech-com-Black-Straight-Through-Serial/dp/B00A6GIVM2/ref=sr_1_1_sspa?crid=3D9RGAKMVO7K0&keywords=db9%2Bcable%2B2ft&qid=1648161390&sprefix=db9%2Bcable%2B2ft%2Caps%2C85&sr=8-1-spons&spLa=ZW5jcnlwdGVkUXVhbGlmaWVyPUFUNEU0SkFQOEdUQTlmZW5jcnlwdGVkSWQ9QTAzMzU4NDMyTVRUMUwxMElQSzZGJmVuY3J5cHRlZEFkSWQ9QTA1NDI2MDExTkFWVVc2TFdBNTc1JndpZGdldE5hbWU9c3BfYXRmJmFjdGlvbj1jbGlja1JlZGlyZWN0JmRvTm90TG9nQ2xpY2s9dHJ1ZQ&th=1		1	$ 4.95	$ 4.95		st	jpc	Monday, Mar. 28			
9	3/8/2022	Mcmaster-Carr	7779K52	https://www.mcmaster.com/7779K52/	Miniature Snap-Acting Switch	8	$ 3.79	$ 30.32	$ 9.38	st	LES	30-Mar	Order #661292!	FedEx	566558755631
0	3/28/2002	Lowes	Phifer 3-ft x 7-ft Ch	https://www.lowes.com/pd/Phifer-3-ft-x-7-ft-Charcoal-Fiberglass-Replacement-Screen-Kit/1000179075		1	$ 9.78	$ 9.78							
1	3/29/2022	Amazon	Regulation Size 1.365" Table Soccer Foosball Table Replacement Balls Official Tournament Table Soccer Balls	https://www.amazon.com/Regulation-Foosballs-Replacement-Officia-Tournament/dp/B08Q823SZ1/ref=sr_1_14?keywords=red%2Bfoosball&qid=1648588176&sr=8-14&th=1	3 more red foosballs so we have enough for a full game	1	$ 7.99	$ 7.99		st	jpc	Thursday, April 7			
2	3/29/2022	Lowes	Boshe Cross-Line L	https://www.lowes.com/pd/Bosch-30-ft-Red-Beam-Self-Leveling-Cross-line-Cross-Laser-Level-		1	$ 59.98	$ 59.98		st					
3	3/29/2022	Lowes	Metalic Sharpie			1	$ 4.48	$ 4.48							
4	4/1/2022	Mcmaster-Carr	4514K2	https://www.mcmaster.com/4514K2-4514K213/	Harsh-Environment DC Circuit Brea	2	$ 67.31	$ 134.62		st	LES		Order #6897062		
5	4/4/2022				Re order			$ -							
6	4/5/2022			https://www.amazon.com/Connector-Adapter-Terminal-Signal-Module/dp/B071DS5GTW/ref=		1	$ 8.69			st	jpc	April 6			
7	4/7/2022	Lowe's	23837	https://www.lowes.com/pd/Silver-Line-Plastics-5-ft-L-Schedule-40-Pipe/5002110677	3in-5ft PVC cellcore			$ 24.19				7-Apr			
8	4/7/2022	Lowe's	58401	https://www.lowes.com/pd/Hillman-Steel-Screw-Eye-Hook/3036096	steel screw eye hook			$ 1.28				7-Apr			
9	4/7/2022	Lowe's	332304	https://www.lowes.com/pd/Hillman-Zinc-S-Hooks/3239991	s hook			$ 1.28				7-Apr			
10	4/7/2022	Lowe's	3735983	https://www.lowes.com/pd/ReliaBilt-ReliaBilt-4-7-8-in-Stainless-Steel-Screen-Door-Pull-Handle-1-Pack/5002824245	handle			$ 4.48				7-Apr			
11	4/7/2022	Lowe's	1329453	https://www.lowes.com/pd/Blue-Hawk-10-ft-Weldless-Nickel-Plated-Steel-Chain/1000834064	door chain			$ 9.38				7-Apr			
12	4/7/2022	Lowe's	23407	https://www.lowes.com/pd/Charlotte-Pipe-3-in-x-3-in-dia-PVC-Schedule-40-Spigot-Test-Fitting/3357828	rod cover end cap			$ 5.12				7-Apr			
13	4/7/2022	Lowe's	23467	https://www.lowes.com/pd/Oatey-8-fl-oz-PVC-Cement/4750803	pvc cement			$ 7.48				7-Apr			
14	4/7/2022	Lowe's	326271	https://www.lowes.com/pd/Oatey-PVC-Flange/50315121	pvc flange			$ 41.72				7-Apr			
15	4/12/2022	Amazon	ANMBEST 4PCS DB	https://www.amazon.com/ANMBEST-Breakout-Connector-	4 Female DB-9 breakouts	1	$ 19.99	$ 19.99		mt		14-Apr-22			
16	4/12/2022	Amazon	Fermerry 20AWG S	https://www.amazon.com/Fermerry-Stranded-Electrical-Si	20 AWG wire	2	$ 10.99	$ 21.98		mt		April 14, 2022			
17															
18															

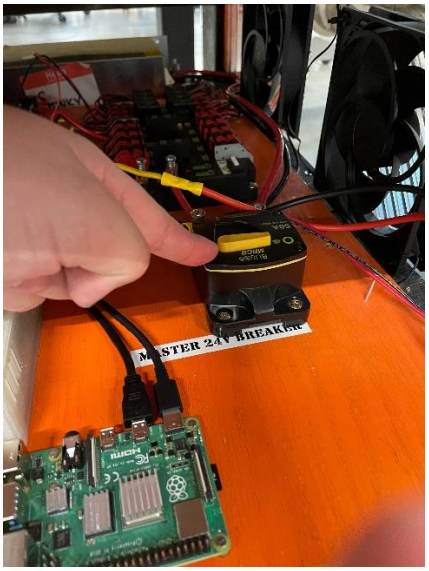
20 Appendix D: End User Manual

20.1 General Operation Instructions

To TURN ON the Foosbot Table and play a game:

1. Ensure the Power Cable is plugged into a wall outlet or extension cord. (120V AC)
2. Open the acrylic door on the back of the table.

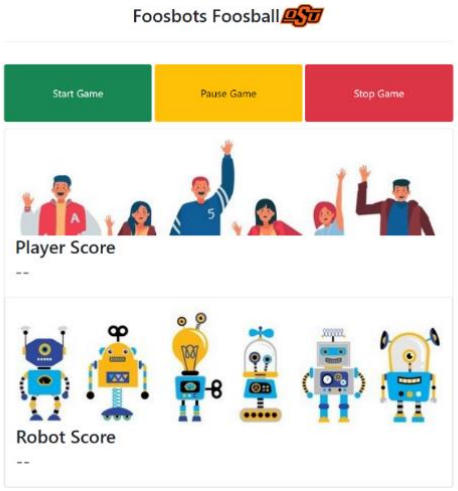
3. FLIP THE POWER SWITCH on the breaker inside the table.

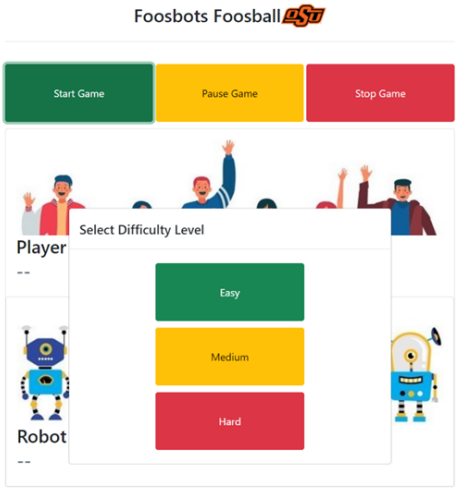


4. Close the door.

5. Wait for the bootup sequence to complete. (This takes 1-2 minutes and the rods will move and then zero during this time)

6. Once the game UI is showing on the touch screen, Press START GAME and select a difficulty level.

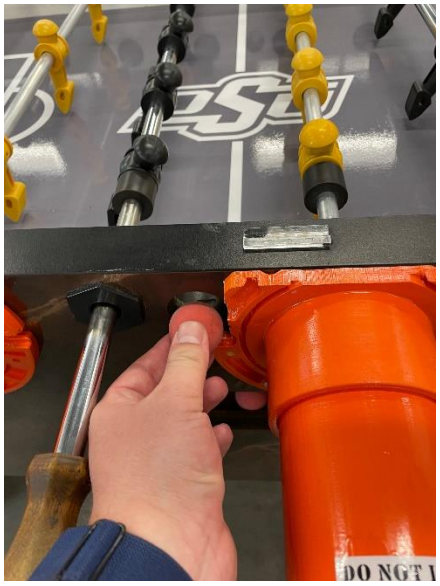




7. Once the Motor controllers are all showing BLUE AND RED lights, the game is ready to start.

8. Push the ball return mechanism in all the way, and then pull back to make sure it completely turns over.

9. Take a ball from the return slot and put it on the table via the hole between the 5-Rods.



10. The table will begin playing foosball. Good Luck

During play, the ball may get stuck in a dead spot. If this happens:

1. Press the PAUSE GAME button on the touch screen.
2. Remove the ball from the table.
3. Press RESUME GAME
4. Return the ball to the table via the hole between the 5-Rods.

If the Motors lose track of their position (i.e. the players are pointing up and kicking the air):

1. Press the PAUSE GAME button on the touch screen.
2. Remove the ball from the table
3. Press the RESUME GAME button on the touch screen.
4. Put the ball into either goal. This will cause the rods to re-zero themselves and they should begin working as normal once again.

To END the game, press the STOP GAME button.

To TURN OFF the table:

1. Open the acrylic door on the back of the table.
2. Flip the breaker switch to the OFF position.
3. Close the door.

20.2 Disassembly and Transportation Instructions

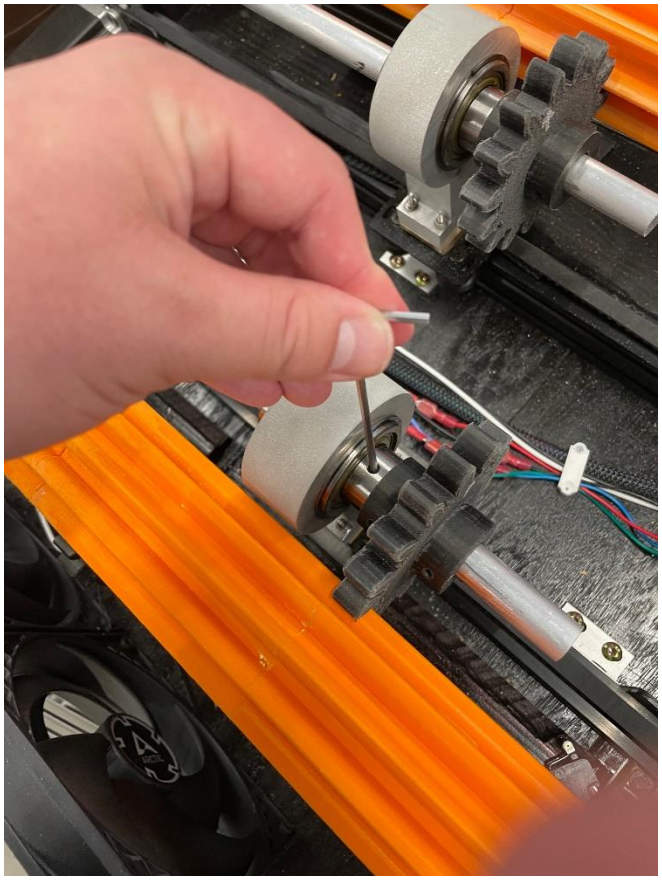
To DETATCH the TABLES:

1. Remove the Acrylic TOP COVER PANEL from the Actuator table. Gently pull on the edge to slide it out of the slot in the T-Channel Frame. This should be done with TWO people.

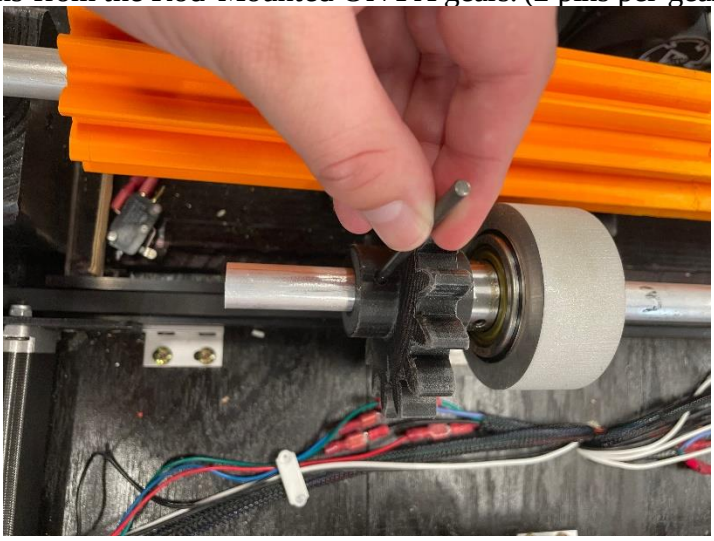
2. Gently remove the Acrylic BACK COVER PANEL from the T-Channel Frame.
3. Looking at the table from the front, on the RIGHT SIDE of the table several cables go from the foosball to the actuator table, and connect to the lower acrylic panel via panel mount connectors. DISCONNECT the from these connectors.

4. UNLATCH the red-handled latches which connect the actuator table legs to the foosball table legs.

5. LOOSEN the set screws which connect the aluminum rods to the gantry arm attachment. (2 set screws per bearing)



6. REMOVE the pins from the Rod-Mounted ONYX gears. (2 pins per gear)



7. Slide the ONYX Rod gears off the aluminum rods.
8. Now the Tables can be gently separated without issue.