

Autonomous Robot Design for Field Navigation, Obstacle Avoidance, and Ball Launching

ECE 118 Mechatronics Final Project

Group 19

Alexi del Rosario
Clayton Heath
Luca Guiga

University of California, Santa Cruz
June 2026

1 Introduction

This report reviews the design and development for an autonomous, ping-pong ball launching robot built for a robot soccer-style competition. The objective was to have the robot had to start in the starting zone, navigate to the initial shooting zone (ISZ), avoid obstacles, locate the opponent using an IR beacon detector, and fire ping-pong balls at the opponent within a two-minute match. The design also had to stay within the size and cost constraints of the project. Meeting these requirements involved a lot of integration between mechanical design, sensors, motor control, and software design. This report covers the main design requirements, the hardware and software design, challenges faced, testing results, and major takeaways from the project.

2 Design Requirements

2.1 Functional Requirements

The goal of this project was to design and build an autonomous robot capable of navigating to the end of the player's field, locating the opposing robot, and launching ping-pong balls at said opponent. Points are awarded for successful shots. 100 points are given when the ball goes through the circular goal on top of the opponent, and 10 points are awarded if the ball strikes the body of the opponent. If the ball bounces on the field before hitting the opponent, the points are not awarded.

Figure 1 shows the layout and measurements of the player's field. The robot starts in the red starting zone, and may be in any rotational orientation. From there, it must navigate across the field to the Individual Shooting Zone (ISZ), shown in blue. The dotted lines show the three possible obstacle placement regions. The left and right borders of the field are also labeled because they will be referenced later when discussing the robot's state machines and software architecture.

The minimum requirement is to have at least two balls hit the opposing robot for two out of three rounds. The total gameplay may not exceed 2 minutes, and the total cost of materials must remain within a \$150.00 budget.

2.2 Physical Constraints

Each player's field is a white 4 x 8 area with a 2 inch black tape boundary. Additional black tape lines run across the center of the field to indicate the different field zones. The robot is required to keep at least half of it's body within the black tape borders. It must detect obstacles and avoid contact within 5 seconds. In Figure 2, it shows the entire field with both opposing fields. Example obstacles are colored orange.

The robot was required to fit within an 11 x 11 x 11" cube of compliance. The obstacles were 10 x 3 x 10" (Figure 3). The obstacles have a track wire 3" above the ground, emitting a 24-26 KHz signal. The beacons, positioned 8" above the ground, may be set to 1.5 and/or 2.5 KHz.

Since the distance between obstacle position lines is 14" and the obstacles are 3" wide, the clearance between obstacles may be approximately 11".

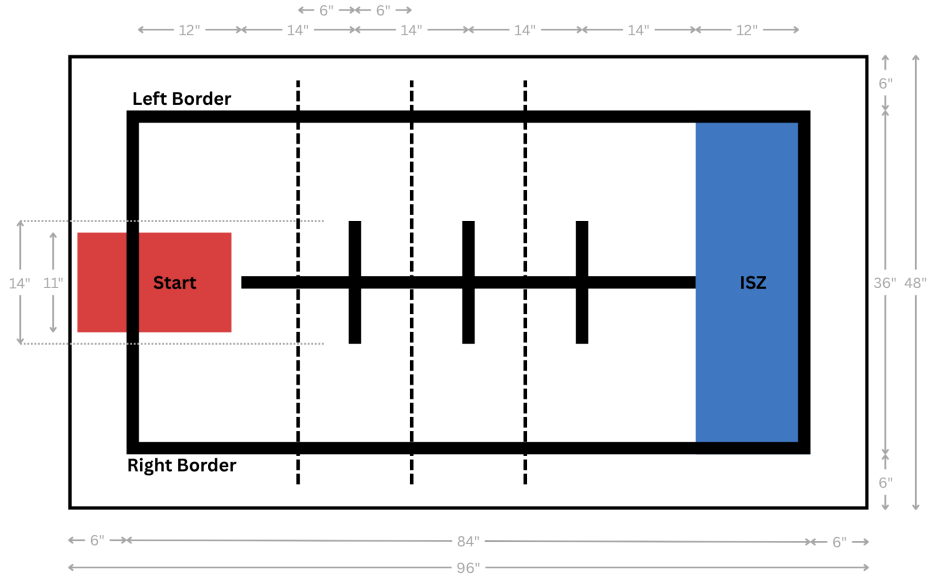


Figure 1: Field layout

Both robots must position a 2 KHz IR beacon 11" above the ground.

2.3 Software Requirements

The robot software was programmed in C using MPLAB X IDE and built using the provided ES_Framework. The software architecture is event driven, where event checkers continuously monitor changes in sensor states and post events to services and the hierarchical state machine (HSM).

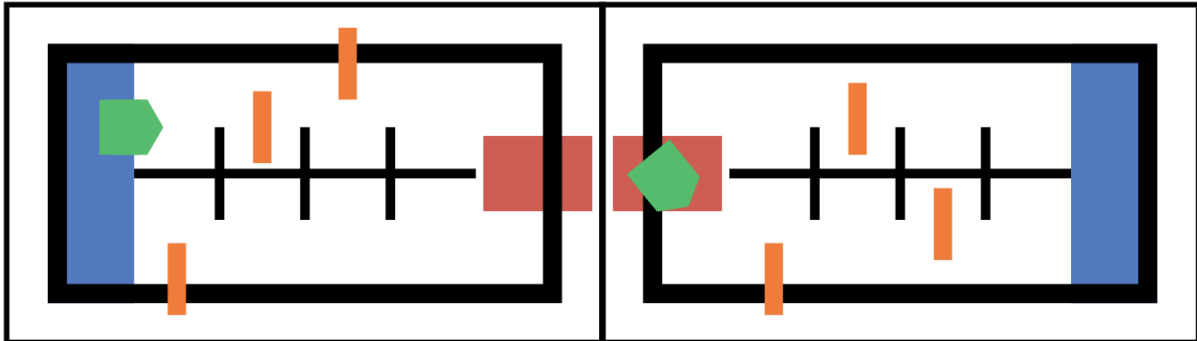


Figure 2: Left: Opponent droid in the ISZ, Right: Player in starting position

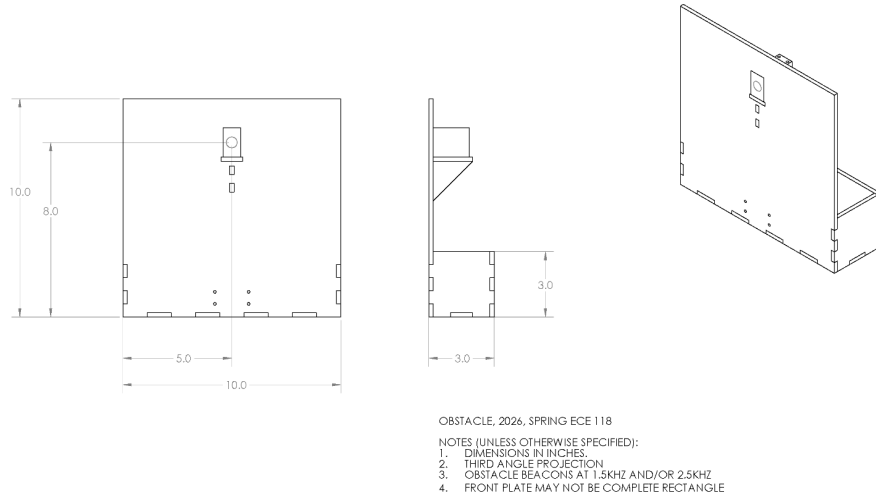


Figure 3: Obstacle Design

3 Initial Design Concepts

3.1 Design 1:

Figure 4 shows the initial design concept, which includes a slanted base, a large ball basin, caster wheels on the front, and large rear-mounted wheels. The design also leaves room for circuits, including the Uno32 I/O stack, motor driver boards, the battery, and soldered circuits. The ping pong ball cannon is mounted at the front center and uses a double-flywheel, gear-driven design. The goal was to step up the provided 12 V DC motors from their maximum no-load speed of 150 rpm to 2000 rpm, allowing the robot to shoot a ping pong ball a maximum distance of 16 feet. The slanted base was included to prevent balls from getting jammed or collecting on one side of the basin. Internally, the loading mechanism uses a revolving loader with four slots for balls to fall into and one opening where the balls can drop into the shooter.

The sensor layout for this design consisted of tape, trackwire, and ping (ultrasonic) sensors. The ping sensors were mounted behind the wheels underneath the base platform. The tape sensors and trackwire sensors were located on the front of the platform, with the idea that the tape sensors would keep the robot aligned with the tape while traveling along the line, and the trackwire sensors would allow the robot to detect obstacles in front of it and also while moving around them.

3.2 Design 2:

Figure 5 shows a modular upgrade to the structure used to mount the electronics. This design includes a dual-layer base with screw holes for custom mounting brackets to secure circuits in place. At the top, a beacon is mounted on a small DC motor, which operates a wire mesh unjamming system that rotates to disrupt balls from settling and getting stuck.

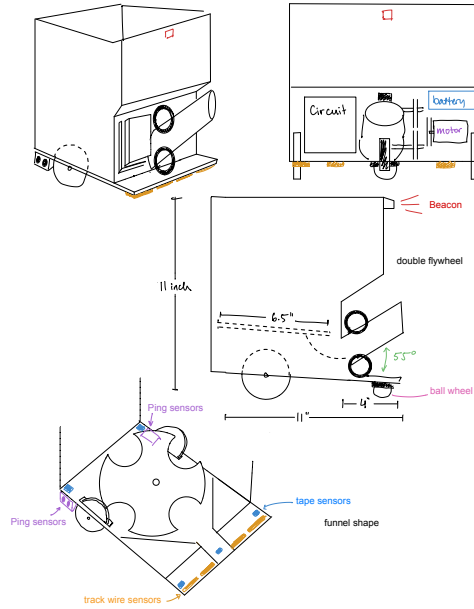


Figure 4: Design 1

At the bottom of the basin, a tube connects to the same gear-driven ball cannon from Design 1. A servo was included in the cannon design to control ball loading, allowing the robot to shoot incrementally instead of firing bursts. The drive motors, drive wheels, and caster wheels were kept in the same positions.

The sensor suite included tape and trackwire sensors. The I/O board only has eight analog sensor inputs, which limited the number of sensors that could be used to detect distance over a range. Because this design maximized space, a larger sensor suite was desired to provide more visibility and allow the robot to react more effectively. To address this, a tank circuit input board was designed to take multiple inputs and reduce them to a single analog input on the stack. This design would use four pins connected to 12 tank circuits arranged across the four quadrants of the top platform. The tape sensors were located on the bottom platform and faced downward to track the tape lines.

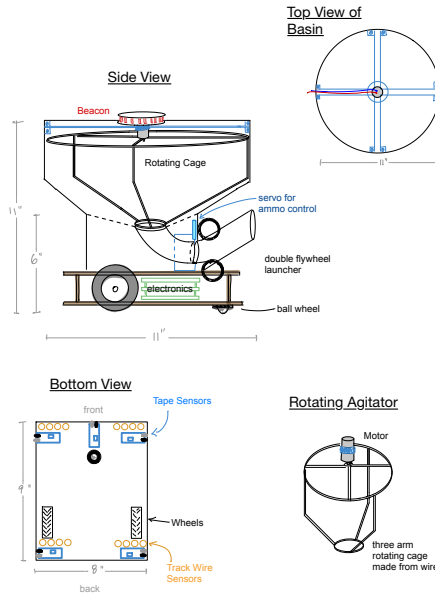


Figure 5: Design 2

4 Final Design

After the initial design iterations and trying different approaches, we finalized a design that best met the project requirements and addressed issues discovered along the way. The following sections break down the final mechanical, electrical, and software systems that were integrated to create the final robot.

4.1 Mechanical Design

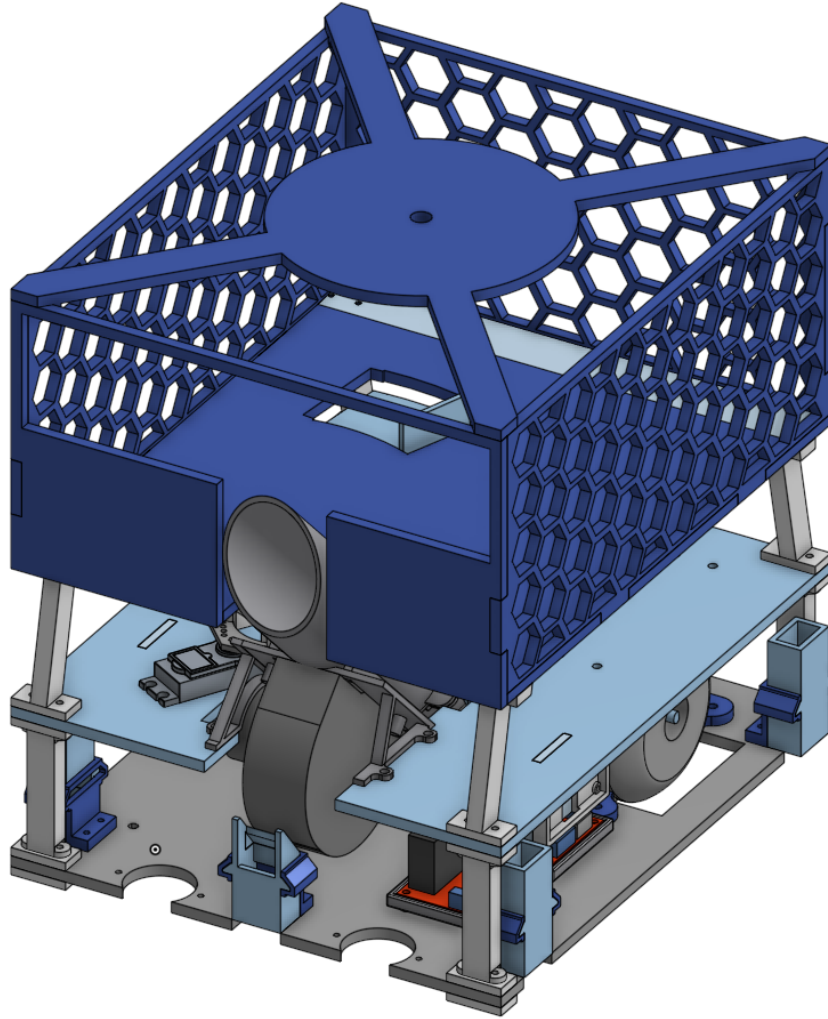


Figure 6: Final design: full assembly

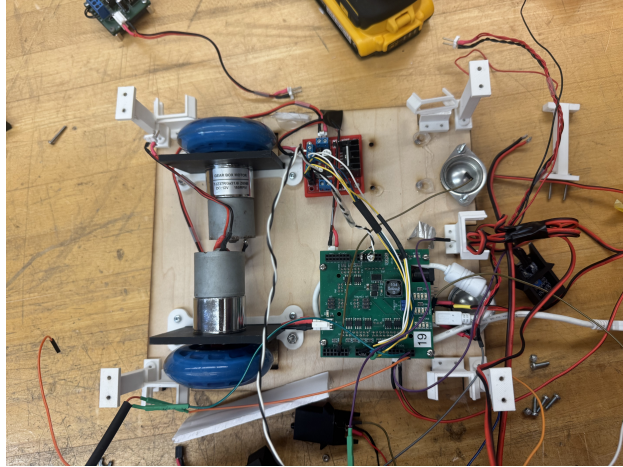


Figure 8: Base Layer with all components

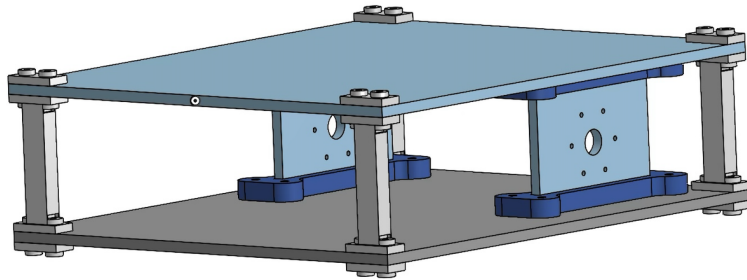


Figure 7: Updated Robot Base

Figure 7 shows the updated flat, dual-layer platform concept with redesigned wheel mounts that work across both platforms. The mounts were screw-based and were initially laser cut, but they were later changed to 3D-printed parts after the laser cutter stopped working. The four posts in the corners served as mounting brackets that held the two platforms together using screws and hex nuts.

4.1.1 Layer 1: Base

The wheels were mounted toward the middle rear of the platform. We used rubber wheels that were 3 inches in diameter, and the mounts held them in place well under load. The rubber wheels were mounted with a 3D-printed insert that fit the gap between the motor drive shaft and the wheel. The tolerance was kept as close to zero as possible so that the pressure from pressing the insert onto the motor shaft would keep it in place and prevent slipping. Figure 8 shows the mounting locations for the wheels, as well as the brackets holding them. The four corners have risers and are secured with screws. The top center

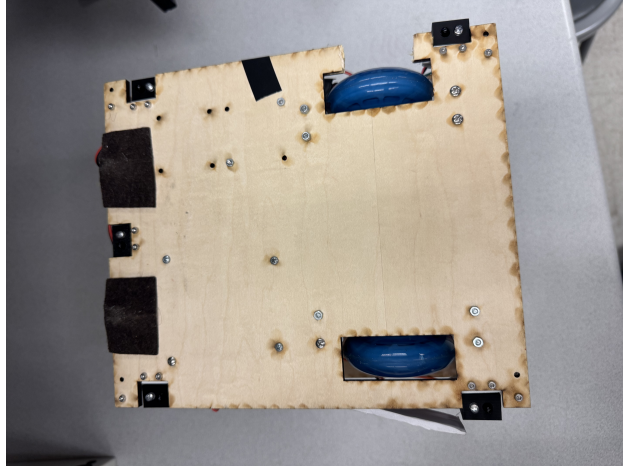


Figure 9: The bottom of the robot. The black rectangles are tape sensors, the two wheels are blue and the felt pads at the front are covering the caster wheels.

shows the H-bridge screwed into a red mount. The holes directly to the right were intended for a second H-bridge, but this was later changed to a DS3658 driver because it allowed us to draw more current by putting all the outputs in parallel. This was done so the flywheel, which powered the cannon, could maintain its speed as consistently as possible. This mount was secured with hot glue, but it repeatedly came undone due to vibrations created by the robot.

Mounted near the bottom center is the Uno32 stack, and surrounding the perimeter are five other white mounting brackets that hold the tape sensors. The tape sensor housings were designed to slip in and out of their brackets in anticipation that they might break easily and require maintenance or replacement. Aside from the single motor driver board, every component was secured in place with screws and lock nuts.

4.1.2 Layer 1: Underneath

Figure 9 shows the wheel mounts and caster wheels. The top wheel has a piece of wood that was cut away so the wheel would not drag against it and cause the robot to move erratically. The black felt-like material covering the caster wheels was used to slow down the robot during the initial scan of the beacon on the opponent's field, allowing for a more accurate survey of the beacon's location. Later, while trying to complete the final minimum specifications (min spec) checkoff, the state machine was edited to make the robot shift forward and backward quickly to jostle any stuck balls free and prevent jamming.

4.1.3 Layer 2: Shooting

On the middle layer, we mounted the main shooting mechanism, the LDO board at the bottom center Figure 19, and the beacon detector circuit. Figure 10 shows the location of each component. The red rectangle on the far right is the housing for the beacon Figure 18.

Shooting was a difficult challenge. Initially, the plan was to use a large gear train to step up the rpm of a DC motor and shoot the ball. However, Bels provided flywheel motors with

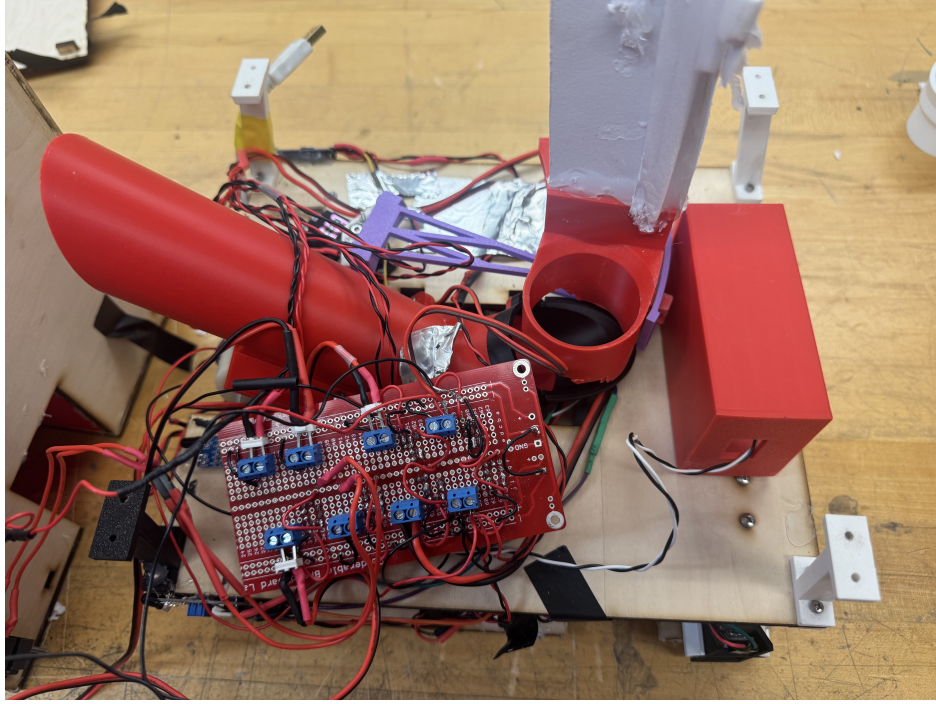


Figure 10: Robot middle level with circuits and loading mechanism

foam around the wheel, which launched the ball well over the required 16 feet at 5 V. The second design change was a fixed barrel at 45° . The idea was that the barrel did not need to move horizontally because the robot could be programmed to rotate, and if the angle was fixed, we would not need to worry about shooting over obstacles. The loader design was originally the revolving drum from Figure 4, but given the space constraints and the continued desire to maximize ball load capacity, a simpler and smaller sliding system was used. The pivot point was a servo mounted next to the flywheel. The design slid left and right by 23.75° .

The ball diameter was roughly 40 mm, or about 1.57 inches. We set the diameter of the tubes that the balls fell through to 1.65 inches, giving about 0.0375 inches of clearance on each side of the ball. This allowed the balls to move freely through the tubes even if there were slight size inconsistencies.

The slider was designed to fit through a cutout in the ball basin, as shown in Figure 6. The wall was meant to disrupt and displace the balls to prevent jamming while the shooting state operated. When the slider was over the barrel leading to the flywheel, as shown in Figure 12, the flat part on the left side of the wall prevented any other balls from falling through into the second layer. Figure ?? also shows this, as well as the platform where the ball would drop and rest. In Figure ??, the slider is open, meaning the hole is visible through the cutout in the basin and a ball can fall through. The slider was controlled by a two-second timer, moving for one second in each direction, which allowed the robot to shoot one ball every two seconds. Shot power was also regulated with PWM and varied to accommodate for a potentially closer target.

The first iteration of this system worked relatively consistently. Around 9 out of 10 reload

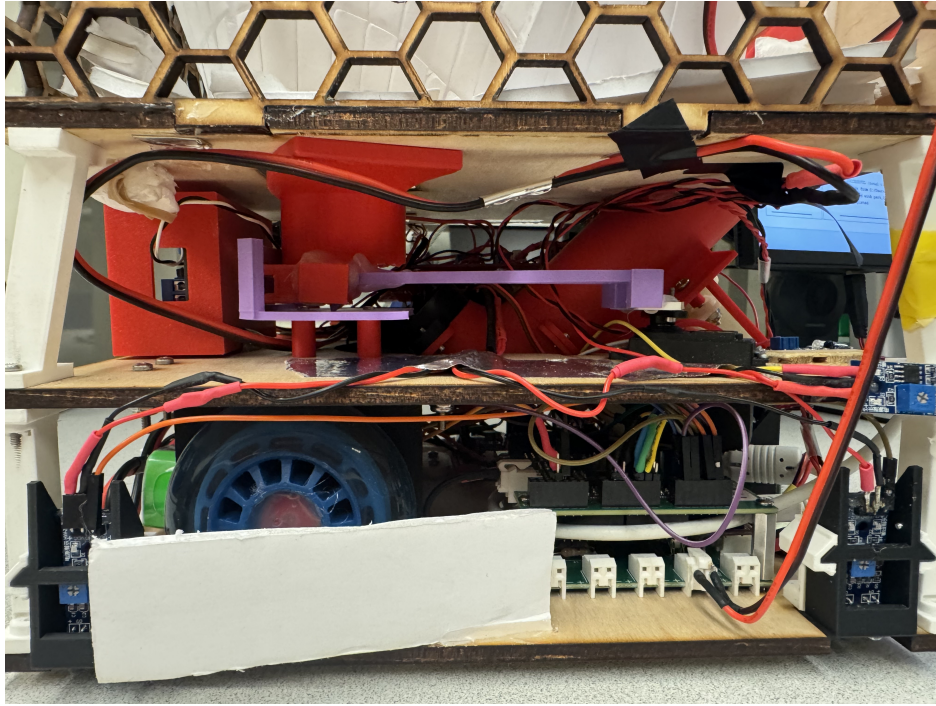


Figure 11: Side view of the load slider

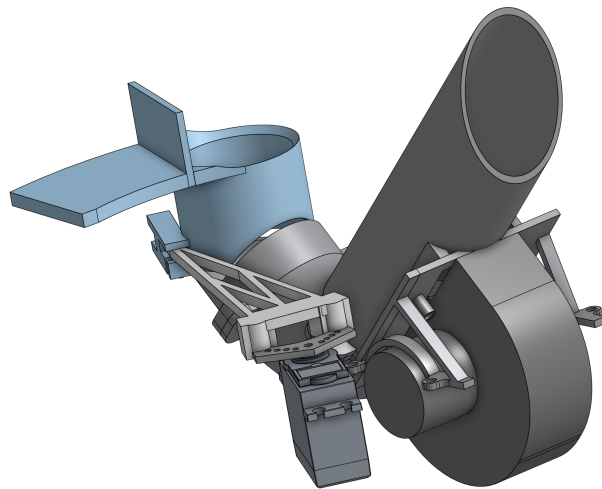


Figure 12: Ball Slider

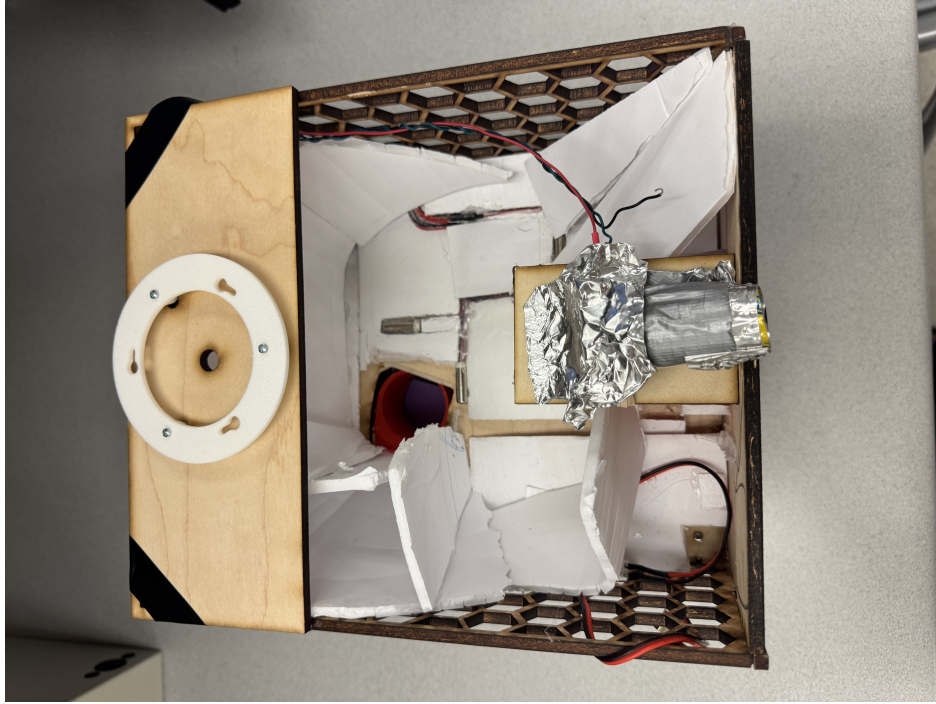


Figure 13: Top view of final robot basin interior

cycles were successful. A jam became apparent when the balls settled into a formation that the wall could not break up. This was fixed by adding foam core to create a clearer angle inside the basin, guiding the balls toward the hole, as well as by adding a taller wall.

Figure 13 clearly shows the use of foam core, as well as the silver vibration motors. The vibration devices did not work because the glue and foam core dampened the vibrations almost completely, even at maximum voltage. Otherwise, the layout shows the basin angled toward the hole, with the wall extending upward from the hole.

We found that using more than 30 balls, all 40 mm in diameter, was too much, while using fewer was almost too little. There was a sweet spot where the robot could fire consistently with a maximum capacity of 30 balls. The ball basin could originally fit up to 87 balls.

4.1.4 Layer 3: Ball Basin

At the very top, the beacon mount can be seen in white on the left, and the beacon detector can be seen on the right. The beacon detector was wrapped in reflective tape so the IR from the beacon would not trip the filter circuit. Mounted under the basin were two power switches and the remote switch on the Uno32.

4.2 Electrical Design

4.2.1 Trackwire Detector

The obstacles contain a loop of wire at a height of three inches off the ground that is driven with a square wave at approximately 25 kHz. When an inductor is held near this wire, it

can detect the signal. With a tank circuit to resonate more at the desired frequency, an amplifier, and a peak detector, we aimed to avoid obstacles by detecting this signal. As we only really cared about whether the signal came from the left or right, we wanted to see if we could use multiple inductors to drive the same input, allowing us a greater area of coverage to ensure the bot could never hit the obstacle from the front and not see it. We used the signal generator running a square wave at 0.4 V peak to peak at 25 kHz and probed running multiple inductors in parallel. With experimentation and some math in Desmos, we found that the best solution was to feed the input with multiple identical tank circuits and have the option to add a little more capacitance to get the center frequency closer to the desired signal.

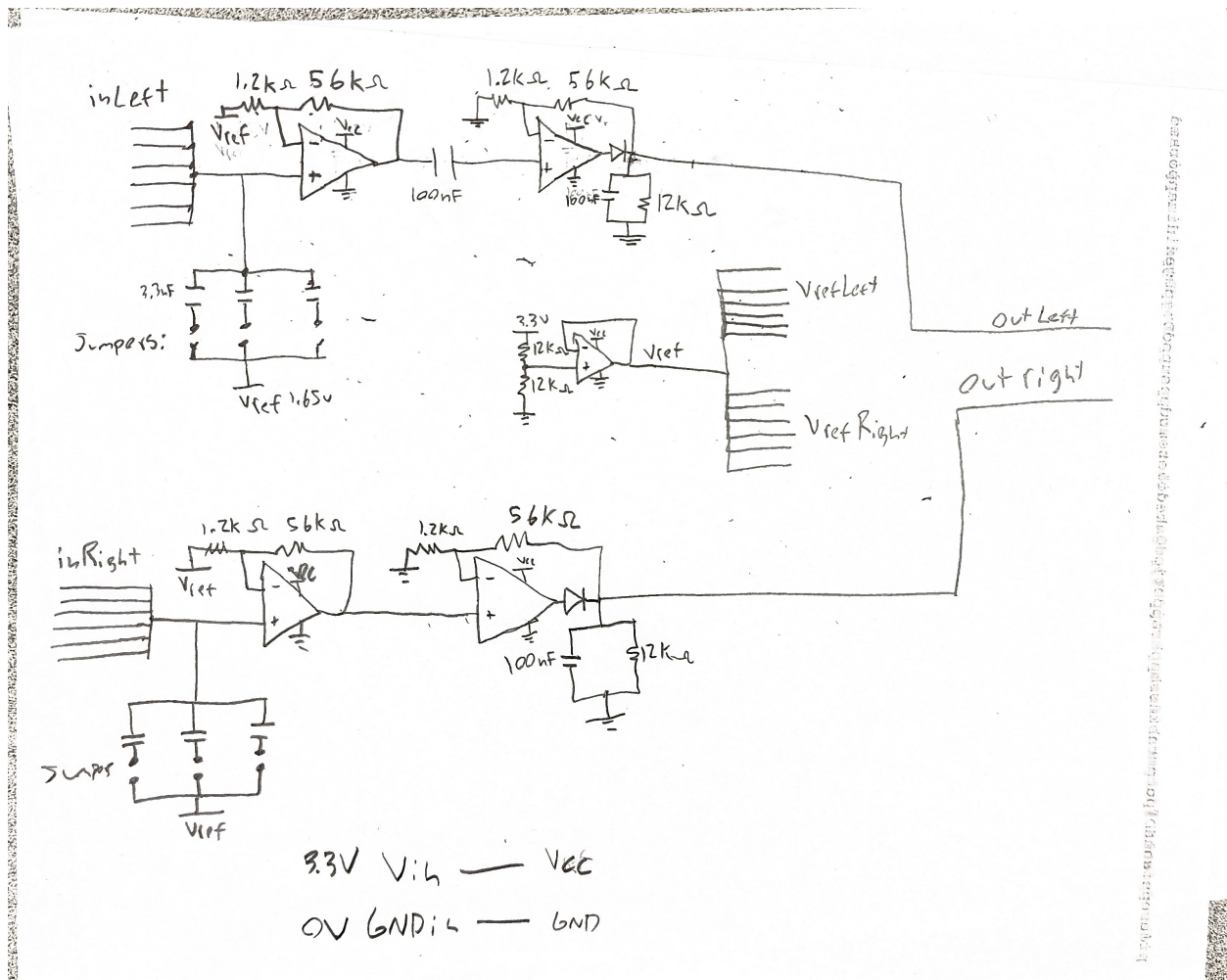


Figure 14: Trackwire Circuit Diagram(replace with neat one later)

With a single gain stage in front of a peak detector with gain, the circuit required only one IC for each side and fit on a small proto board. We added connectors that allowed us to plug in as many tank circuits as we desired up to six to each side and jumper pins to adjust the capacitance. For the tank circuits, we cut and sanded a proto board into many

very small proto boards and put a tank circuit on each with leads to connect to the main board.

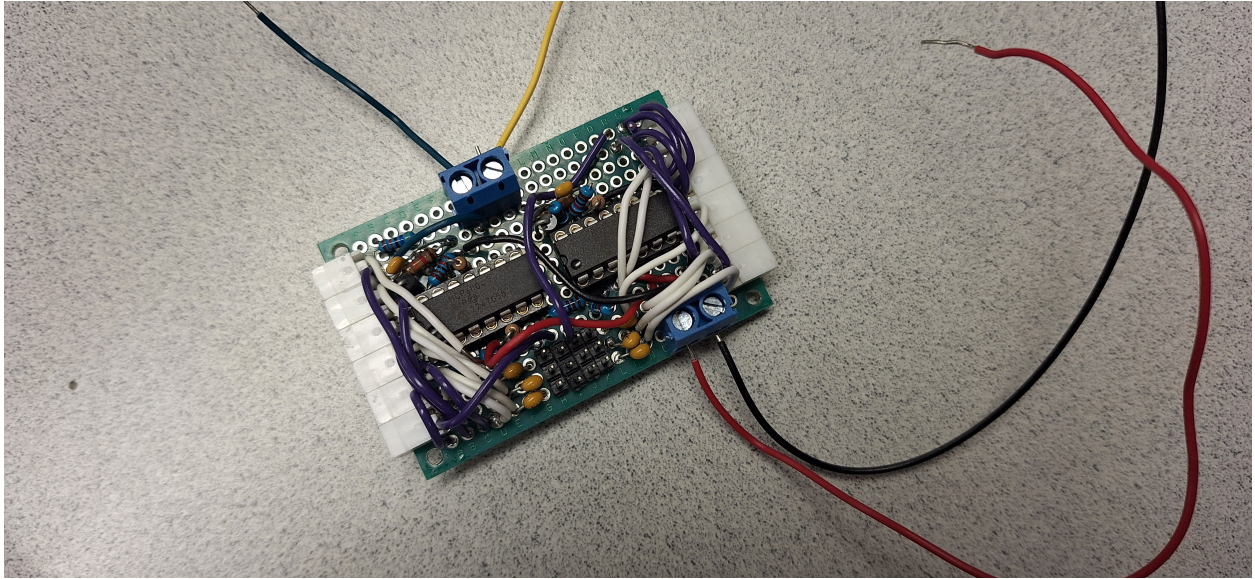


Figure 15: Trackwire Detector

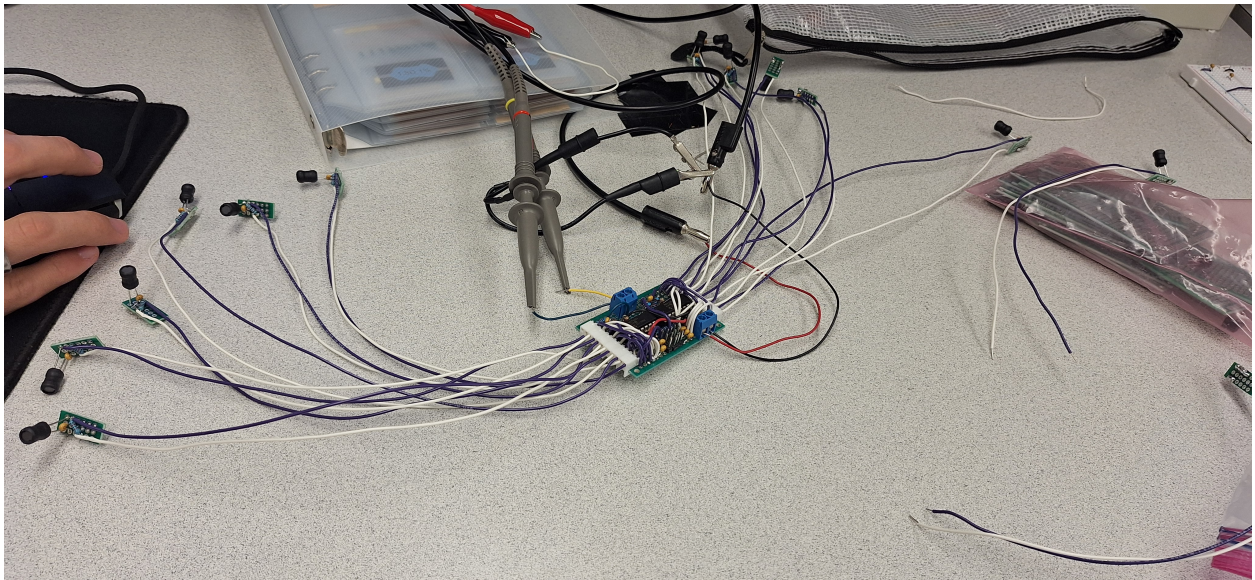


Figure 16: Trackwire Detector With Tank Circuits Connected

The trackwire detector worked great and was able to see the obstacle from about 5 inches away when it was powered with a 9.9 V battery. Unfortunately, when our power distribution broke, it destroyed the left side, so we transitioned to using forward facing tape sensors rather than rebuilding the trackwire detector.

4.2.2 Beacon Detector

The beacons on top of the bots emit a square wave of IR light at around 2 kHz. To identify this signal, we used an IR photo transistor and filter the output to isolate this signal. Although we had a working beacon detector from the labs, it was not strong enough to detect the opponent from the 16-foot span of both fields required for seeing the opponent from the ISZ. Moreover, our previous detector was highly dependent on the beacon, as they varied and were often around 100 Hz away from 2 kHz. Knowing this, we redesigned the filter to have a larger pass-band. We spent a while with the Analog Devices Filter Wizard to create a filter with a 200 Hz pass-band that was strong enough to reject 1.6 and 2.4 kHz and stable enough with our component tolerances. While testing our two stage filter, we found that it was not enough and tried adding a third stage. We settled on simply duplicating the two stage filter after testing. To test, we used the signal generator to drive a biased square wave at 50 mV and 5 mV peak to peak to represent the input from the phototransistor from within a few feet and at 16 feet, respectively. We then recorded the peak to peak for signals we wanted to accept and reject and modified until we were satisfied with the behavior.

Frequency (kHz)	Output Voltage (mV)
1.5	29
1.6	45
2.4	92
2.5	59

Table 1: Two-stage output with 50 mV input.

Frequency (kHz)	Output Voltage (mV)
1.9	43
2.0	72
2.1	75

Table 2: Two-stage output with 5 mV input.

Frequency (kHz)	Output Voltage (mV)
1.5	10
1.6	18
2.4	52
2.5	29

Table 3: Three-stage output with 50 mV input.

Frequency (kHz)	Output Voltage (mV)
1.9	42
2.0	110
2.1	121

Table 4: Three-stage output with 5 mV input.

Frequency (kHz)	Output Voltage (mV)
1.5	9
1.6	13
2.4	38
2.5	20

Table 5: Four-stage output with 50 mV input.

Frequency (kHz)	Output Voltage (mV)
1.9	66
2.0	233
2.1	243

Table 6: Four-stage output with 5 mV input.

On the soldered version, we placed non-inverting amplifiers with potentiometers between each stage, allowing us to adjust the gain to tune the filter after its construction. To the input of the filter, we placed a trans-resistive circuit driven by four phototransistors in parallel connected to the proto board with a screw term block and some long wires. Wiring four of them made our input slightly more consistent, and we tested this before installing. Finally, we placed a peak detector after the filter and read that as our output using the analog pins on the microcontroller. With the beacon 25-30 feet away and the potentiometer gains at minimum, we printed the ADC output and saw that it hovered at 2 counts but went up to 5 when pointed at the beacon. Meanwhile, we could hold the phototransistor microphone directly up to the obstacle and still only read 2 counts.

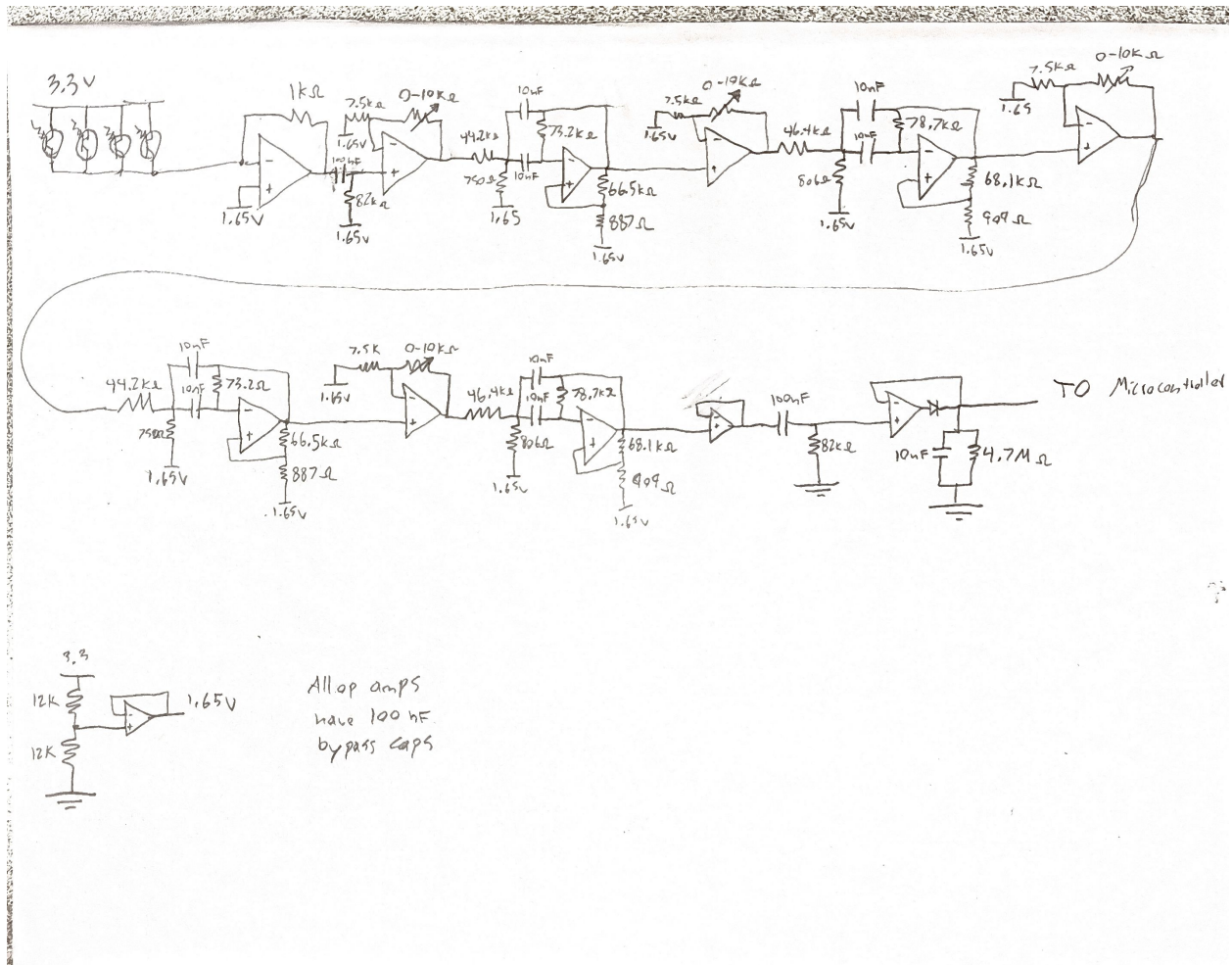


Figure 17: Beacon Detector Circuit (replace with neat one later)

As we worked on the bot, we eventually found that it worked best with all the gains turned all the way up. At one point, the peak detector broke, and we transitioned to using a digital peak detector instead, directly reading the output of the last filter stage. To ensure we did not break the circuit as we moved it around, we designed and printed a mount to hold it safely.

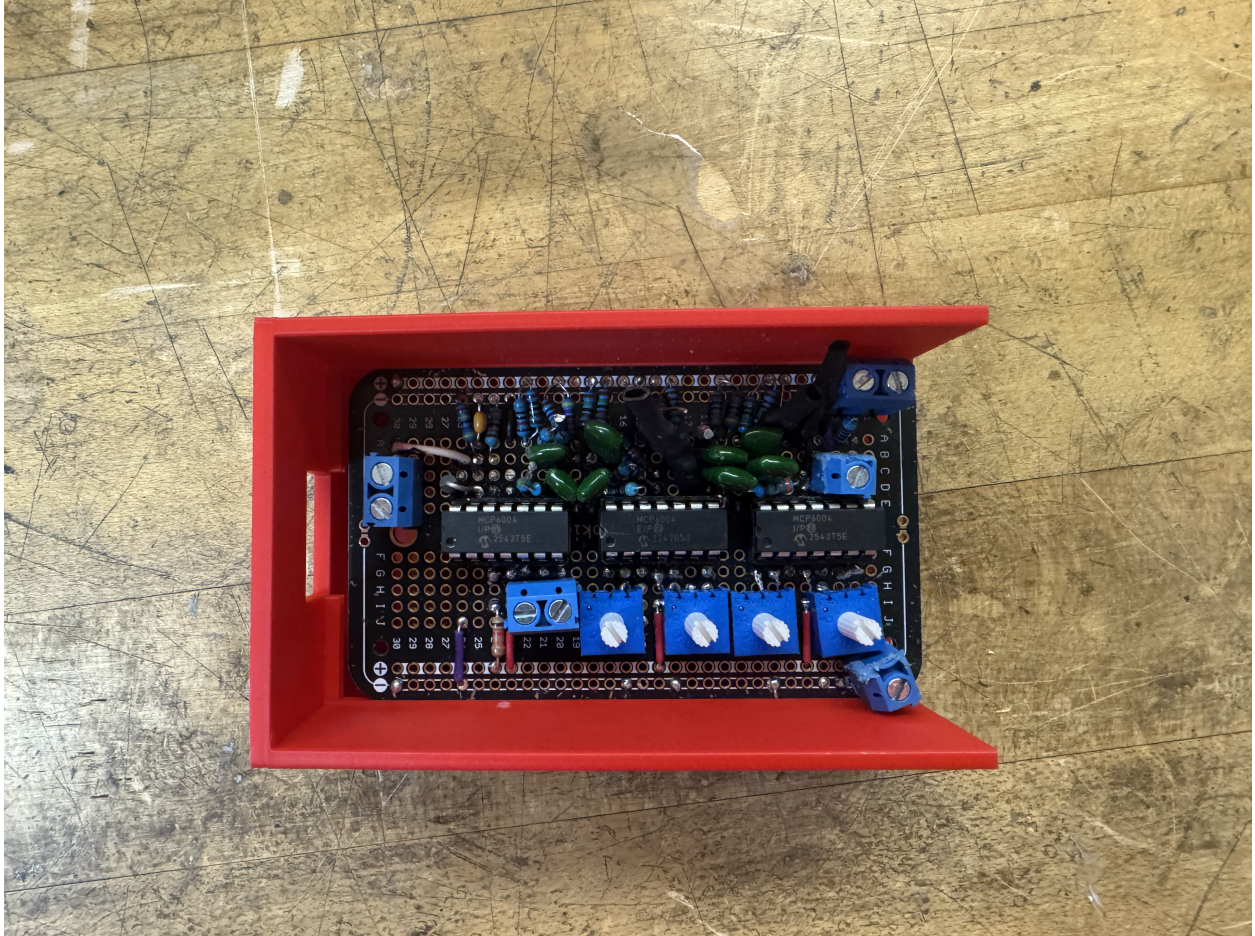


Figure 18: Beacon Detector in Protective Mount

4.2.3 Power Distribution and Wiring

One of our greatest struggles in this project was our power distribution. The wheels and Uno stack took the 9.9 from the battery, the servo and flywheel took 5 V, and the beacon detector, trackwire detector, and tape sensors took 3.3 V. We fried 3 power distribution boards before coming to our final design. In the first iteration, we used a diode in series with the input for reverse polarity protection, and the 3.3 V LDO was fed from the 5V LDO. When this went bad, we remade it without that diode, with the 3.3 V LDO getting power directly from 9.9V, and with room for heat sinks. At some point the 3.3 V LDO went bad, so we rebuilt it again with an additional 3.3 V LDO. Again, the 5 V LDO went bad. After closer inspection, we found that we were likely drawing too much current. Each LDO can only source 1 amp, and the flywheel and servo both can draw about that much. Finally, we made a power distribution board with four 3.3 V LDOs and two 5 V LDOs, essentially allocating one per sensor or actuator once tape sensors were grouped together.

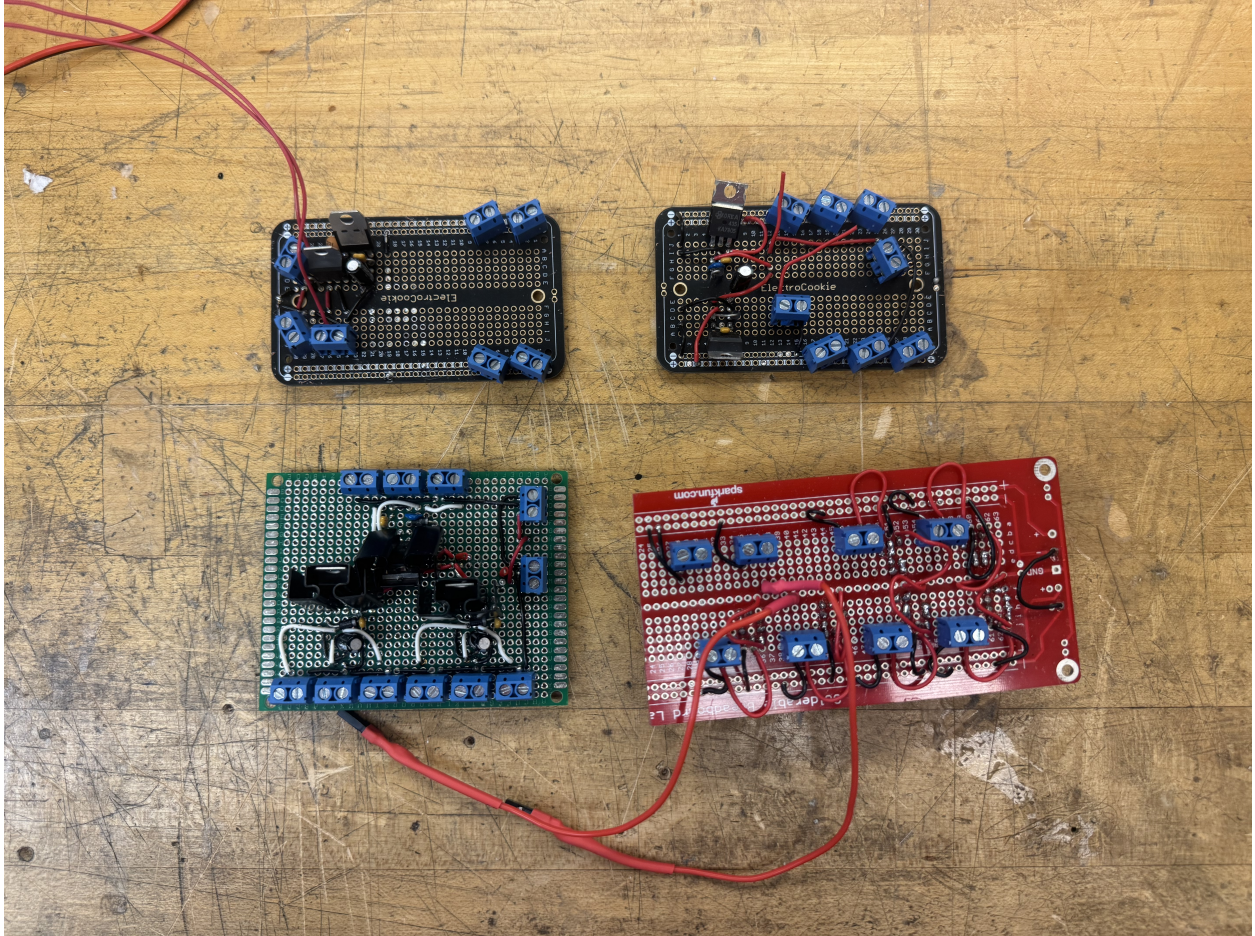


Figure 19: All Iterations of Power Distribution. Working One is Red with LDOs on Back

The initial design had a second H-bridge to drive the flywheel and rotating ball loader. They would each only be rotated in one direction. However, we switched from a DC motor driven ball loader to a servo driven one and were dissatisfied with the power the flywheel had when giving 3.3 V to the H-bridge, despite testing the flywheel with 3.3 V from the DC power supply. Instead, the flywheel is driven by a DS3658 Low-side driver with all sinks ran in parallel. This configuration can sink 2.4 A and makes much more sense for driving the flywheel in a single direction. We ran into issues with our Uno stack LEDs getting power even when it was shut off, as well as some strange behavior where the board would constantly need the bootloader reflashed. A piece of metal bridging some leads on the stack and a correction to the DS3658 wiring fixed these issues. We were powering the chip with the 9.9 V from the Uno stack but sinking 5 V from the LDO. Instead, we powered the driver from the same 5 V it was sinking through the flywheel. The tape sensors were simple to wire but often gave problems. After verifying that they output low enough and high enough logic signals, we used the potentiometer to tune them for the tape detection then used jumper cables to connect them to the stack. Wanting a more permanent solution, we used 2X36 male pin headers to make a more rigid, soldered connection. While these connectors worked fine for the wheel drive pins and analog pins, they failed twice on the tape sensors. It is

important to get the solder joint well adhered and the pins long enough to stay in the Uno stack. Soldering good connectors was very helpful in managing our cables, and heat shrink kept them insulated and organized. For our analog pins, we wrapped them in grounded shielding wire to protect from noise.

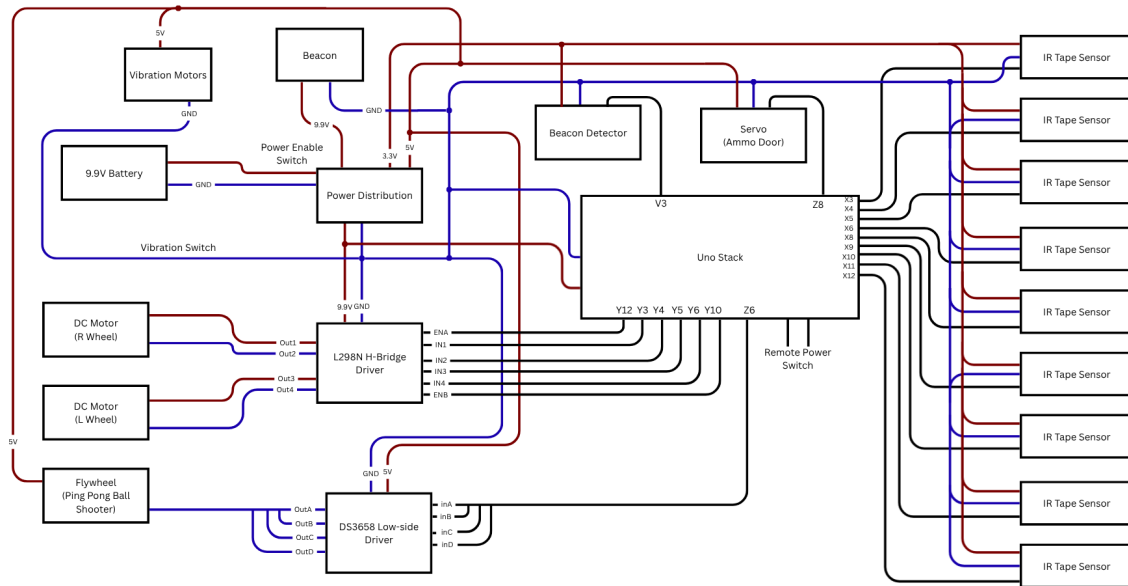


Figure 20: Final Block Diagram

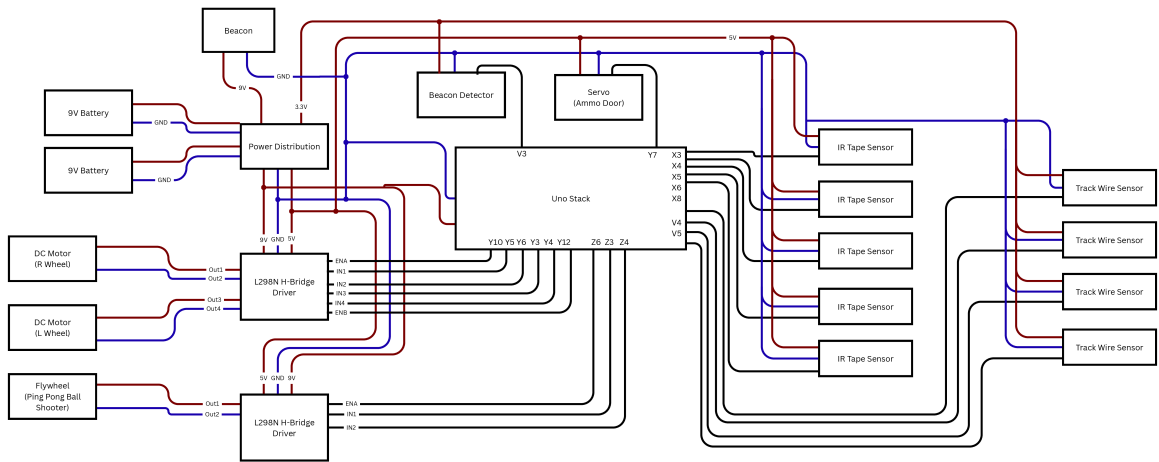


Figure 21: Block Diagram Version 1

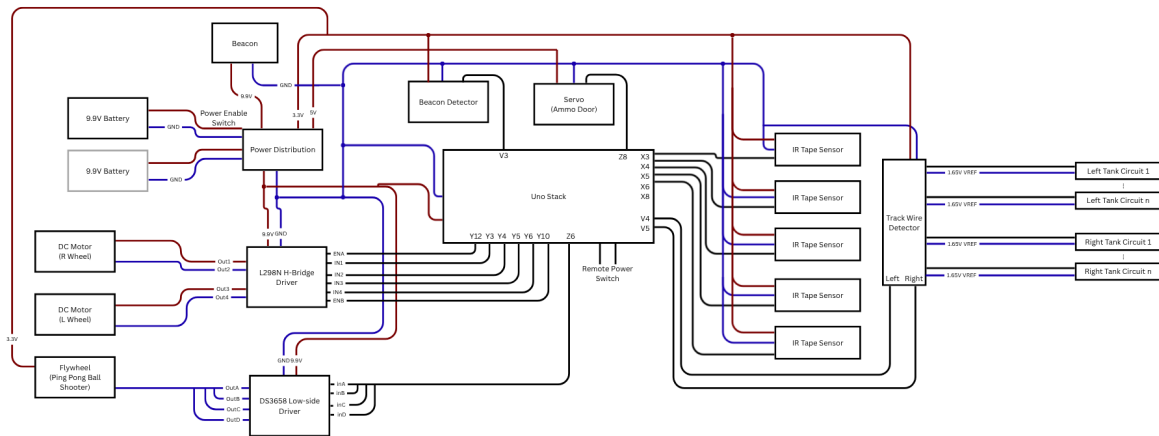


Figure 22: Block Diagram Version 2

4.3 Software Architecture

4.3.1 ES Framework

4.3.2 Hierarchical State Machine

Top Level All of the states in the top level of the Hierarchical State Machine (HSM) contain their own sub state machines that handle the robot's movement. Each sub state machine manages the behavior required for its specific task, while transitions between the top level states are triggered by events posted from within those sub state machines. These events bubble up through the framework and signal when the top level should transition to the next state.

The robot begins in Orient, where it looks for the opponent's IR beacon. Once the beacon is found, the robot will be facing both the opponent and the Front Border. The Orient sub state machine then posts FOUND_PEAK, and at the top level, SetAlignExit(ARRIVED_FRONT_BORDER) is set so Align knows where it should transition to next. Every time Align is entered, it will move the robot forward until both right and left front tape sensors are touching the tape, aligning the robot with the border.

After the first alignment, it will turn about 90 degrees, so the front and back tape sensors on the right side of the robot are positioned the inside of the front border. It will then move forward until the sub state machine posts ARRIVED_RIGHT_BORDER and then traverse along the right border towards the ISZ. If it encounters an obstacle, the state machine transitions into FollowRightAvoid. once it reaches the Left Border, it will then align itself with Left Border in the Align state, then continue moving along the left side of the field by using FollowLeftLine.

Similarly, FollowLeftAvoid mirrors the behavior of the right side, eventually transitioning back into FollowRightLine. While moving along the Right or Left Border, the robot will enter the ISZ when the front center tape sensor has been triggered, indicating it has reached the ISZ.

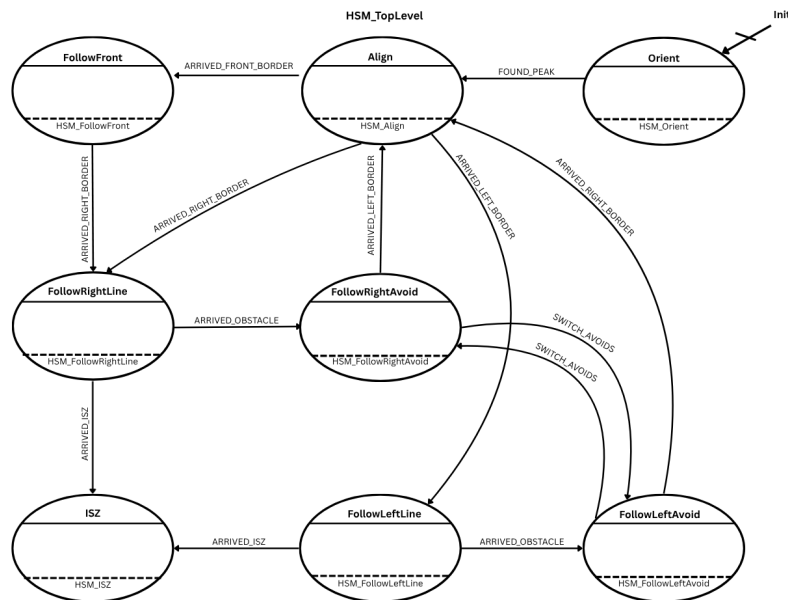


Figure 23: Top Level of HSM

Orient originally had logic to find the Front Border and follow along it until reaching the Right Border. However, we had issues with getting the robot to make more controlled movements near the front right corner of the field. We also realized that in order to have more consistent obstacle avoidance, the robot needed to make proper 90 degree turns so it would be perpendicular to the obstacle before avoiding it. This led to adding the FollowFront and Avoid sub state machines.

Now, Orient is responsible only for locating the opponent's IR beacon. The robot first turns in one direction to note the highest signal, then turns in the opposite direction to find that peak signal. Once found, the robot will stop and post FOUND_PEAK while facing the Front Border.

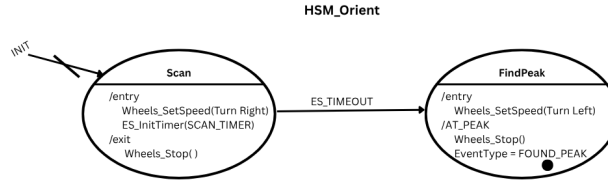


Figure 24: Sub HSM: Orient

Align is entered after the robot approaches the Front, Right, and Left Borders. The robot will first backup until either all of the front tape sensors are inside of the border or until a short timer is complete. Then, the robot will move forward until either the left or right tape sensor goes from light to dark. If the right side is triggered first, the robot will pivot, turning left, until the front left tape sensor also sees the tape. This ensures that the robot is aligned with the tape on both sides. Lastly, the robot backs up until the front center tape sensor gets off of the tape.

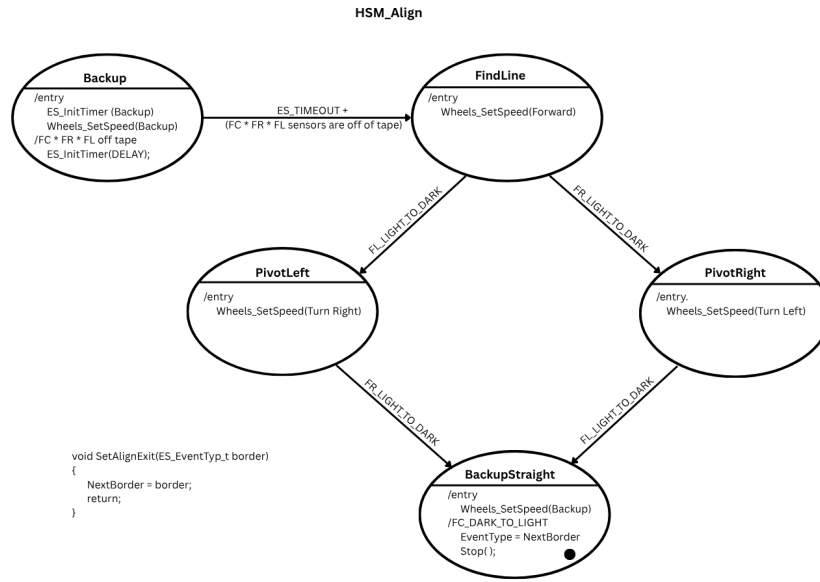


Figure 25: Sub HSM: Align

Follow Front uses the front and back tape sensors to pivot turn, ending with the robot parallel with the Front Border while facing the Right Border, with the front right tape sensor positioned on the inside of the tape.

In OffLine, the robot will move forward while turning right toward the tape. Once the tape is detected by the front right tape sensor, it will enter OnLine, where it moves forward while turning left. The robot will continue to bounce between these two sub states until the center tape sensor is triggered, indicating it has reached the Right Border.

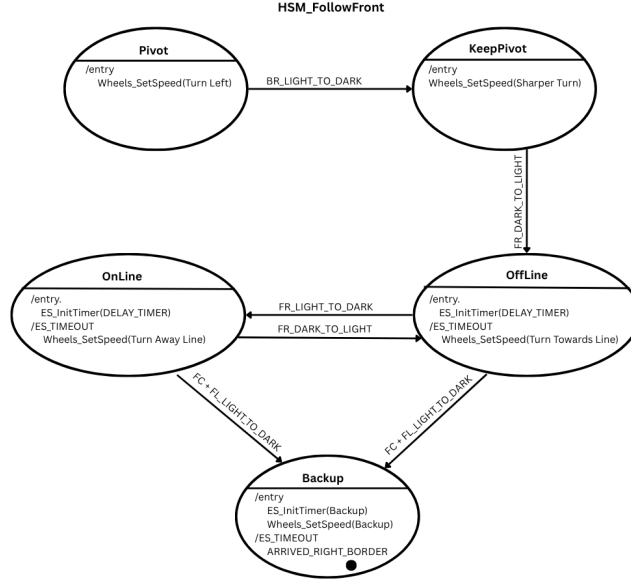


Figure 26: Sub HSM: Follow Front

Follow Right Line uses the front and back tape sensors to make a pivot turn as close to the corner tape as possible. This helps avoid the first obstacle in cases where it is positioned on the Right Border. Without this sharper turn, the large size of the robot can make it face outward as it reaches the first obstacle, causing inconsistent obstacle avoidance.

During the KeepPivot state, a short timer is used to check whether one of the tape sensors has detected an obstacle. If so, the top level transitions into FollowRightAvoid. Otherwise, it transitions into OffRightBorder. Similar to the previous sub state machine, OffRightBorder and OnRightBorder move left and right in order to follow along the tape.

If an obstacle is encountered, the robot transitions into FollowRightAvoid. Otherwise, it continues moving forward until the center tape sensor detects tape, indicating that it has reached the ISZ, where it then performs a small backup maneuver.

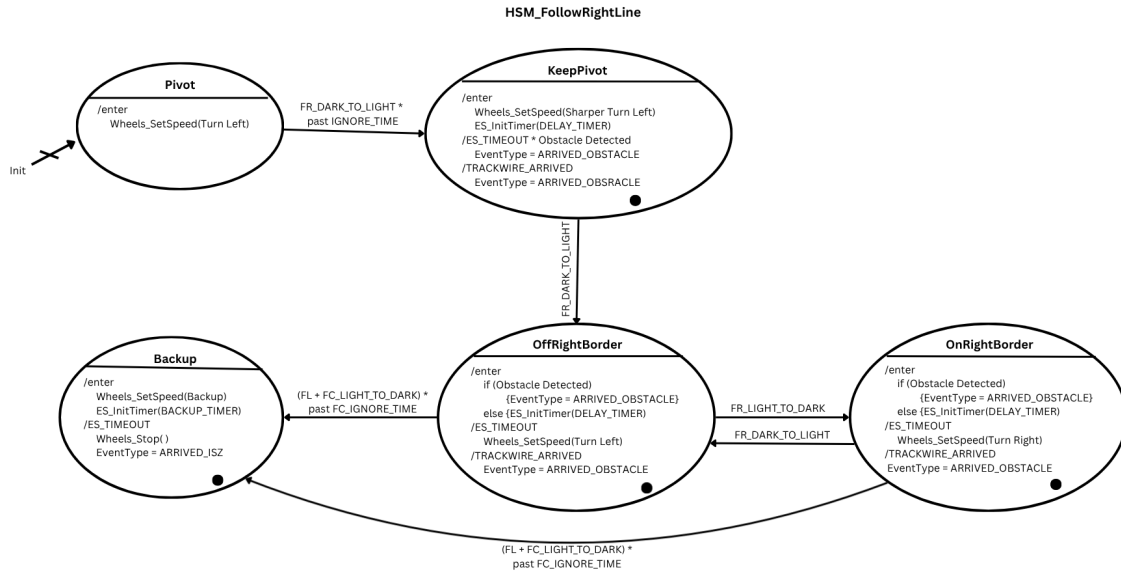


Figure 27: Sub HSM: Follow Right Line

Follow Left Line closely mirrors FollowRightLine, but we did not need the delay timers in OnLeftBorder and OffLeftBorder. Another difference is that KeepPivot did not need to detect an obstacle in any of the test runs we ran with the robot.

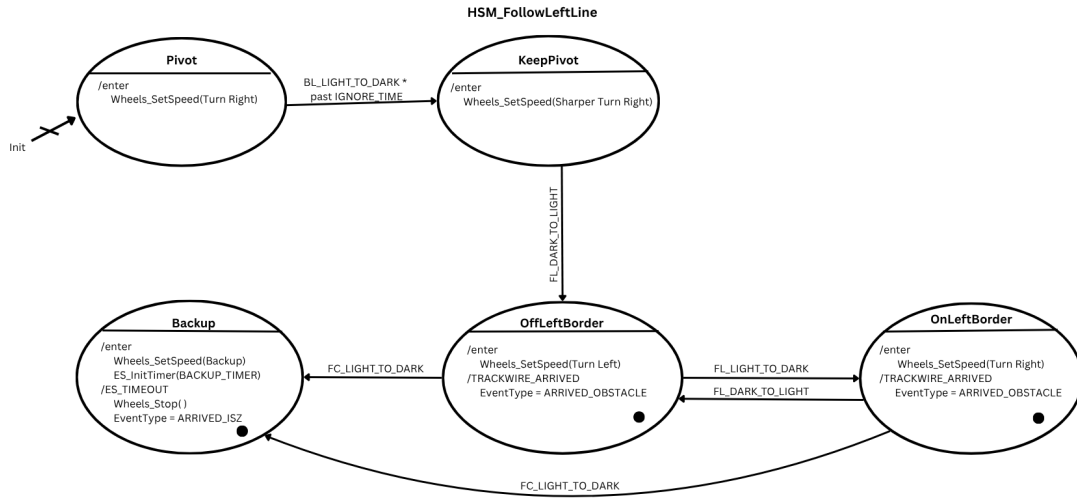


Figure 28: Sub HSM: Follow Left Line

Follow Right Avoid starts by moving back, away from an obstacle and then turns about 90 degrees left. The robot will then move straight, tracking which "zone" of the field it is in by using the tape sensors. The robot starts in the right side since it was next to the right border. Once the front left or front right tape sensor passes tape, it is in the center zone (StraightZoneC). There is a small IGNORE.PERIOD timer that was added because the crosses in the center tape would cause the robot to skip to the next zone prematurely.

The robot will continue to move forward and transition into the left zone once the back right tape sensor detects tape. It will then backup until the center tape sensor is off of the tape and post ARRIVED.LEFT.BORDER for the top level to handle.

In any of the three zones, the robot may encounter an obstacle on the right or left side. It will then transition into the appropriate sub state that backs up and turns away from the obstacle for a set amount of time, then continue to move forward.

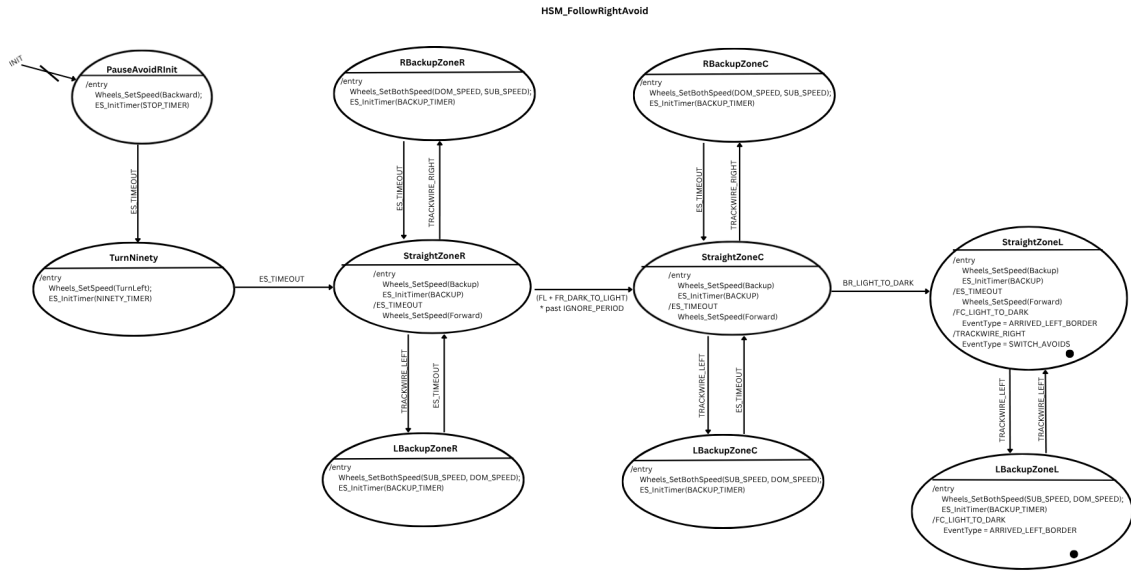


Figure 29: Sub HSM: Follow Right Avoid

Follow Left Avoid mirrors FollowRightAvoid exactly, where it will track the position of the robot by noting the tape sensors being triggered, turning away from obstacles along the way.

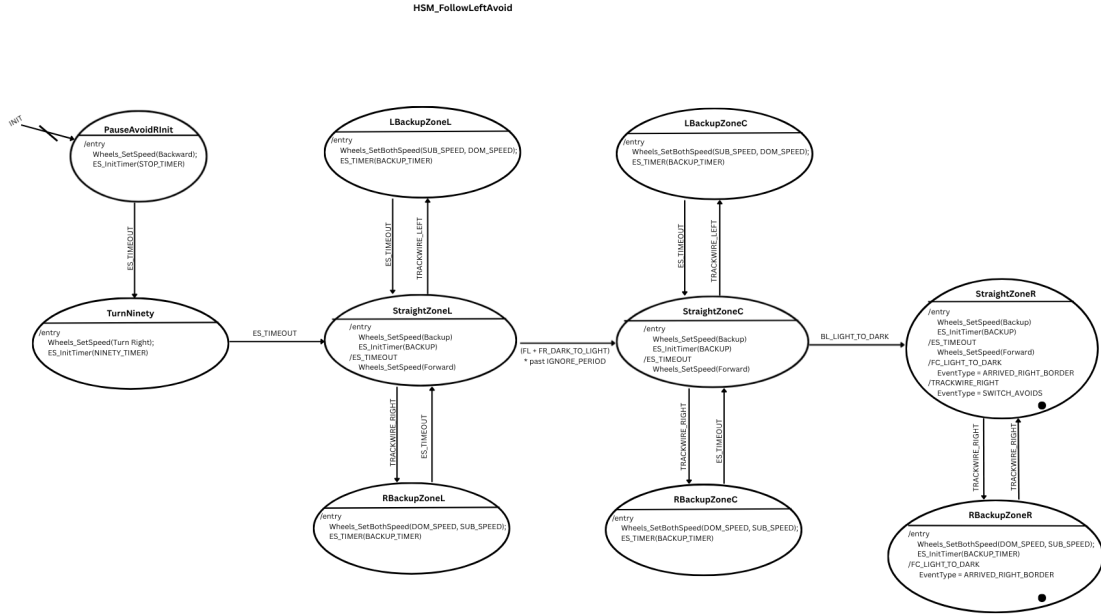


Figure 30: Sub HSM: Follow Left Avoid

ISZ begins by turning toward the center of the field and moving forward, away from the borders. This was necessary because the spins performed during Scan and FindPeak would sometimes cause the robot to rotate off the border.

Once positioned closer to the center, the robot begins scanning for the strongest IR beacon signal. Originally, the motors would either move too quickly to properly detect the peak signal, or too slowly, where the motors could not overcome friction and would fail to move altogether. To solve this, a pulse-like movement was added during Scan, allowing the motors to run at full PWM while pausing frequently enough to take accurate sensor readings.

After completing the scan, the robot turns in the opposite direction and stops once the peak signal has been found. The state machine then enters CloseDoor, where the servo closes and the flywheel launch power is set. The launch power is slightly decremented after each shot to account for variations in distance and battery performance.

Next, the robot enters OpenDoor, where the flywheel is brought up to speed and the servo door opens to release the ball. After firing, the robot performs a short forward and backward jolt to help shift the remaining balls and prevent them from getting stuck in the hopper.

After every 9 shots, the robot returns to FindPeak in order to reorient itself, since the repeated jolting can cause it to shift away from the target.

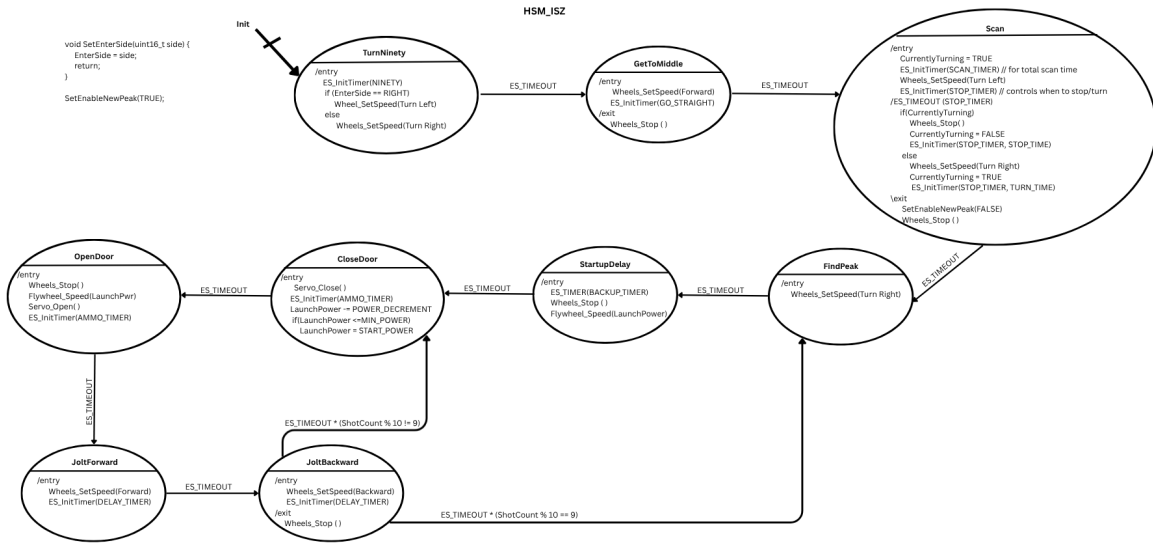


Figure 31: Sub HSM: ISZ

4.3.3 Event Checkers

Using a separate MPLAB-X project and conditional compiling, we created and tested the event checkers. These read the hardware and post events when the hardware changes state.

Tape is detected by reading the digital inputs on port X, where all the IR sensors are plugged in. For each of the pins of interest, if the previous value is different than the current, than an event is posted and the last is updated. The sensors are debounced by reading the free running timer and only checking every 5 ms. This could have been done by a simple periodic service, but having this in the event checker file lets us test all the event checkers in the same place.

Obstacle detection was initially done by reading the analog left and right channels of the trackwire detector. The pins are read and smoothed with an Exponential Moving Average (EMA) filter then compared to both thresholds and the previous state, adding hysteresis. By only printing the EMA output, it was easy to find ADC count thresholds for different distances. After a basic event checker that essentially works as a digital comparator with hysteresis, we added another threshold check so that it posts TRACKWIRE_ARRIVED, TRACKWIRE_DEPARTED, and TRACKWIRE_STRONG events, allowing more control for distinguishing between the obstacle closer and farther back the field. The parameter of the event tells whether it was the left or right sensor. Unfortunately, the trackwire detector broke, so we replaced it with forward facing IR sensors that work the same as tape sensors.

The code is slightly different, as there are two sensors for both the left and right. Thus, we use a mask with both pins to determine if either of them are on and send events according to the last event sent and the current state of the two sensors.

Beacon detection went through several iterations but began similar to the trackwire detector. Reading the peak detector through an analog pin, it is smoothed with a EMA filter. Our first idea was to track whether the signal was increasing or decreasing and use decreasing events to change the direction of turning. By saving the previous voltage and checking whether the difference between the previous and the current is greater than a threshold, and posting to a service that would aim at the beacon. After realizing how much the beacon signal reflects from both the opponent and the robot's own beacon, we changed methods, as this would detect local minima without anyway to know it was the maximum signal. Instead, the state machine makes the bot scan in a full circle, and the event checker keeps track of the maximum value it sees. It then posts an AT_PEAK event whenever it comes back to the peak. The event checker keeps track of whether the last voltage was at the peak, and can only send the at peak event if the last event was not at the peak and the current voltage is within a threshold of the peak, using hysteresis. There is a variable to hold the peak reading for both when the robot is close and when it is far, and the state machine can control which is enabled. When the peak detector broke, minimal changes were needed to keep this method. At first we tried to add decay to the peak detector and EMA smooth it, but after some testing it was clear that this was unnecessary, and the raw reading provided an easily and quickly detectable maximum.

4.3.4 Actuators

The wheels are driven by an enable pin using PWM and two direction pins each. The duty cycle determines the speed and can be given a value between 0 and 1000. The wheel drive code accepts input between -1000 and 1000, wrapping it if the input is outside the range, and sets the duty cycle and direction pins accordingly. We made several extra functions to set the wheel speed in different ways, but only used `Wheels_SetBothSpeed(...)` and `Wheels_Stop()`. The wheels worked best at minimum PWM frequency (100 Hz), which had the added bonus of being far from the trackwire detector center frequency. By using `#defines` at the top of the files, we adjusted the duty cycle and robot's speed easily.

The flywheel is controlled by using a PWM output and two direction pins through the H-Bridge. The software initializes the PWM pin for the flywheel's enable signal, sets the direction pins as outputs, and then starts the motor at a duty cycle of zero. The flywheel speed is set using `Flywheel_SetSpeed()` and takes an input value ranging from 0 to 1000. Since switching to the DS3658 driver instead of an H-Bridge, only the enable pin is used.

The servo controls the loading mechanism for the ping pong balls to reach the flywheel to be launched. The software initializes the servo with the RC servo library and adds the servo output pin. The servo position is controlled by setting the pulse width, with the valid range between 1000 and 2000. The difference between the "open" and "closed" positions is set to be slightly larger than a ping-pong ball.

5 Materials

The material list (Table 7) consists of the main components used in our robot. The components labeled as \$0.00 were either provided by BELS, already owned by a team member, or constructed using the provided materials.

Items that BELS did not carry or had limited quantity of were purchased online through Amazon. Those items were the caster rollers, aluminum foil tape, IR tape sensors, and protoboards.

Table 7: Bill of materials

Item	Category	Qty Needed	Pack Qty	Cost/Item	Total Cost
Ball Wheel/Caster Roller	Actuators	2	6	\$1.50	\$8.99
DC Motor	Actuators	2	1	\$15.00	\$30.00
Ping Pong Ball Shooter	Actuators	1	0	\$3.25	\$3.25
Large RC Servo	Actuators	1	1	\$2.00	\$2.00
Rollerblade Wheels	Actuators	2	1	\$5.00	\$10.00
L298N H-Bridge	Electronics	1	1	\$3.00	\$3.00
Uno Stack	Electronics	1	1	\$0.00	\$0.00
Protoboards	Electronics	5	5	\$1.76	\$8.79
DS3658 Driver	Electronics	1	1	\$6.99	\$0.00
9.9V Batteries	Power	1	1	\$0.00	\$0.00
Aluminum Foil Tape	Sensors	1	2	\$6.97	\$13.93
IR Tape Sensors	Sensors	5	10	\$0.88	\$8.79
Tack Wire Sensors	Sensors	9	12	\$0.73	\$8.79
Foam Core	Structural	1	1	\$0.00	\$0.00
1/4" Plywood	Structural	1	1	\$0.00	\$0.00
Wires	Misc.	—	—	\$0.00	\$0.00
Heat Shrink	Misc.	—	—	\$0.00	\$0.00
Resistor Kit (assorted)	Misc.	—	—	\$0.00	\$0.00
Assorted Screws and Nuts	Misc.	—	—	\$0.00	\$0.00
Hot Glue Sticks	Misc.	—	—	\$0.00	\$0.00
Electrical Tape	Misc.	—	—	\$0.00	\$0.00
Capacitor Kit	Misc.	—	—	\$0.00	\$0.00
Felt Pads	Misc.	—	—	\$0.00	\$0.00
Total				\$47.07	\$112.54
<i>*Items listed as \$0.00 were provided by the university.</i>					

6 Results and Discussion

show lots of pictures of the finished thing talk about improvements or things we would have done differently talk about important changes/fixes talk about important discoveries/things we learned things we liked/didnt like the process/working together

The final results of the project was a robot that could navigate the field, detect obstacles, locate the opponent's beacon, enter the ISZ, and shoot ping pong balls. The Build process was very involved, and over the five weeks the team worked sun up to sun down on the design, circuits and software and all the testing repairs. The final robot can be seen in 6 and following figures.

Several changes would have improved the bot, the biggest of them being reducing the overall size and changing the shape of the frame. Although the robot fit within the size limit, it was still larger enough that it sometimes ran into obstacles or came too close while turning. Shrinking the robot by about 1.5 inches on both long sides would have reduced the hit profile and made turning more forgiving. Another possible improvement would have been using four omnidirectional wheels instead of the two wheel drive castor wheel setup. This would have allowed the robot to shift sideways and correct its position without needing turns. The ball basin could have also been improved, it could physically hold 87 balls but could only operate the firing mechanism when starting with 30. Making the revolver reloader.

Several changes and fixes were made during the design process. The original gear driving flywheel was removed after testing showed that the provided flywheel motor could launch balls far enough at 3.3 V. The rotating drum loader was also replaced with a servo controlled sliding door, which was simpler and easier to control. To reduce jams, foam core was added to guide the balls towards the opening better, the slider wall was made taller and the software was changed to jolt the robot forward and backward after each shot. The power system also had to be redesigned after multiple power boards failed. The final board used multiple 3.3 V and 5 V regulators so that sensors, the servo, and the flywheel were not all relying on the same supply. The flywheel driver was also changed from an H-bridge to a DS3658 low-side driver because the flywheel only needed to spin in one direction.

There were also several pieces of the code that could have been improved. We spent a little while making wheel drive functions that were not used or necessary. Instead, we should have used the braking function of the H-bridges when set to zero and focused on developing things that were needed. The even checkers could have been put in separate modules for readability, and an overall robot module with all the adjustable #defines, shared functions, and hardware initialization. In general, a design that was easier to adjust both in software and hardware would have made debugging and tuning much better. A few changes that we made were instrumental in the robot's success. Particularly, aligning to the tape and using the back sensors to turn perpendicular to it was far superior to backing up randomly when hitting the wall. This ensured that we came at the obstacles at a consistent angle, making the timer-based turn there feasible. The trackwire detectors would have worked great and been more adjustable, but starting with forward-facing IR sensors would have been simpler and quicker.

The main thing learned from the project was that integration is usually harder than making each subsystem work by itself. The shooter, sensors, motors, beacon detector, and software could all work separately, but combining them created new problems. The flywheel

affected the power system, the power system affected the sensors, the robot size affected obstacle avoidance, and the anti-jam motion affected aiming. Some parts of the final design worked well, such as the dual-layer platform, removable tape sensor mounts, fixed-angle shooter, and servo slider. Other parts, such as hot-glued mounts, vibration motors, and crowded wiring, were less reliable. Overall, the project required constant communication and teamwork because mechanical, electrical, and software changes all affected each other. The final robot was not the original design, but it became a working system because of repeated testing, redesigning, and fixing the problems that appeared along the way.