ECE118 Final Project

Team 10
Mechanical Lead: Seenara Khan
Electrical Lead: Quillan Zhen
Software Lead: Nick Kuipers

June 8 2023

Contents

1 Introduction

This paper presents an in-depth analysis of an autonomous robot designed with the capability of precision targeting and autonomous navigation. For the final project the theme was "Slug World Cup," with the challenge of creating a fully-autonomous, self-propelled robot that can successfully navigate an 8' by 4' playing field and leverage existing field features (tape on the field, track wires, beacons at the goal) to navigate around an obstacle and fire ping-pong balls into a goal guarded by a goalie that shifts randomly from left to right.

The robot presented in this paper is the BOAT ("Best of All Time") whose notable features include mecanum wheels, which gave the bot an advantage in maneuverability by allowing for strafing left, right, and overall movement in any direction while maintaining the bot's orientation. The BOAT also incorporates an ESP32 microcontroller with in-built FreeRTOS, an architecture that operates with the same purpose as the ES Framework employed by the PIC32 microcontroller also incorporated on the bot – the ESP32 was intended for use in a broader navigation framework (discussed later in this report) but ultimately was utilized to run the flywheel and stepper motor of the bot's ping-pong ball launcher. Further, standard features of the BOAT include four tape sensors and a track wire sensor for leveraging present features on the field, four bumpers on its corners for collision detection against a wall or obstacle, as well as both a 2kHz and a 1.5kHz beacon detector for "seeing" the goal and goalie, respectively. This hardware was ultimately leveraged into a simple state machine that allows the bot to orient itself towards the goal, move to the 1pt line of the field while avoiding obstacles where necessary, fire ping-pong balls at the goal, and return to the reload zone in a consistent cycle.

This paper will discuss the three major topics covered in developing the robot, in order of precedence through the design process. The Mechanical section will cover the physical makeup of the bot and the planning behind it, including the Computer-Aided Design (CAD) work involved in bringing the BOAT from an abstract drawing to a tangible frame and the process behind refining the initial design. The Electrical section will cover the integrated electrical systems of the BOAT, schematics of custom-made parts where needed, and challenges involved with final integration including wiring management and on-the-fly adjustments to existing circuits. The Software section will cover the test benching of individual electrical components and sensors of the BOAT, and final integration of all systems into a comprehensive state machine operating between both the PIC32 and ESP32 microcontrollers.

2 Mechanical

2.1 Preliminary Design

During the initial phase of the project, we had envisioned the utilization of 3D printing technology to construct the entire launching mechanism of our robot. However, practical constraints pertaining to time, materials, and execution led us to reconsider this approach. Later, we decided to temporarily suspended the development of the launcher and redirected our focus towards the bot's body.

In our design, we organized the robot into three tiers: the top for the launcher, the middle housing the UNO, and the bottom tier accommodating the wheels and H-bridges. To keep the bot held up, we had created 4 walls total for the chassis and 2 for the middle tier. We faced some challenges with the bumpers, initially designed too long, causing the bot to exceed the specified cubic dimension limit. The bot's final size was 9.42 x 10.5 inches, largely due to the the size of the four wheels. This required us to maintain a compact layout to stay within the 11 x 11 inch restriction.

2.2 Final Design

In our final design we kept the overall concept the same (other than the launcher), however, we did change the size of the bumpers, and ended up not using the walls for the second teir. We used four metal dowels to keep the top teir up, and have the UNO caged inside. The reason for this change was because of the amount of wires we had connecting in intricate ways, we needed more room to make these edits, so that is why we tossed the walls on the side, and used the dowels instead.

Shown below:

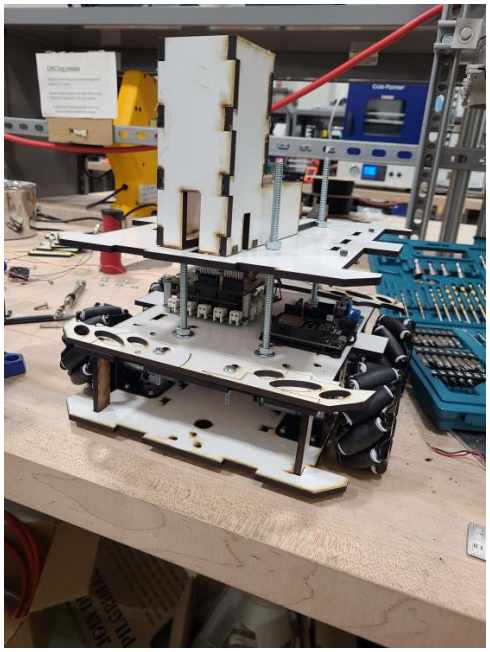


Figure 1: Skeleton Of the Final Design

2.2.1 3D prints

During the development of our bot, I made several small parts to fit all the sensors we included. Some parts stand out because they played a big role in how our bot worked. For instance, I made a stand to keep the flywheel steady. At first, I thought about a holder to surround it, but we found out that we needed to reach the motor from the back. So, I made a simple change and created a flat stand with screw holes for mounting, shown in Fig 2.1.

To hold the Time of Flight (ToF) sensors, I created a small holder, as you can see in Fig 2.2. The space in the middle lets the ToF sensors see out, and the four holes around the outside are for screws. Fig 2.3 shows how I tackled the problem of mounting the stepper motor. Since there wasn't a straightforward way to attach it, I decided to glue it down and add a 3D printed mount. Lastly, I made a shaft extender

for the motor because its shaft was too small. This extender was crucial in getting the balls to the flywheel, and finally what we called the "scooper" which is what feeds the balls into the fly wheel and at the same time stops the balls from falling.

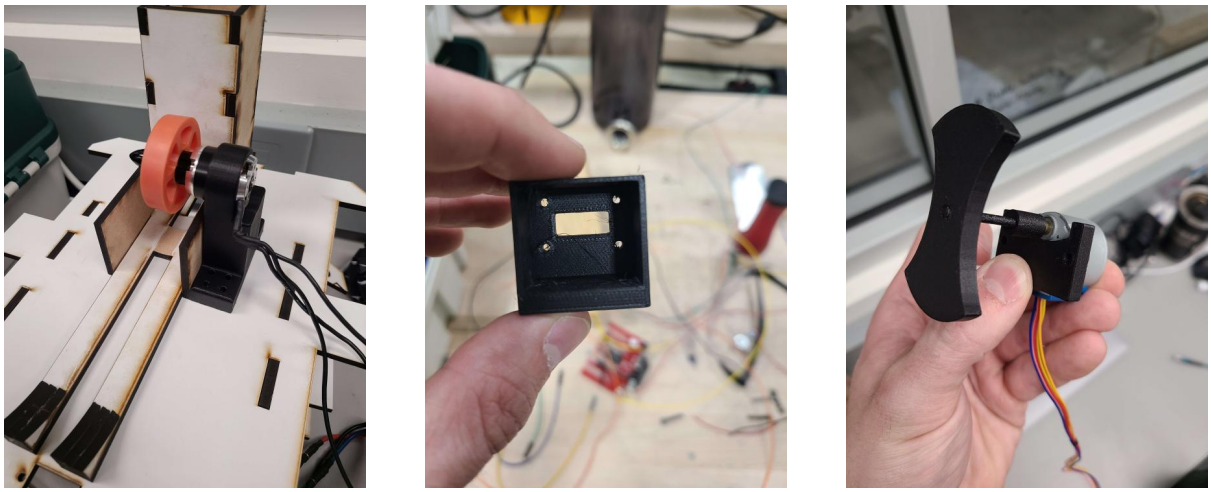
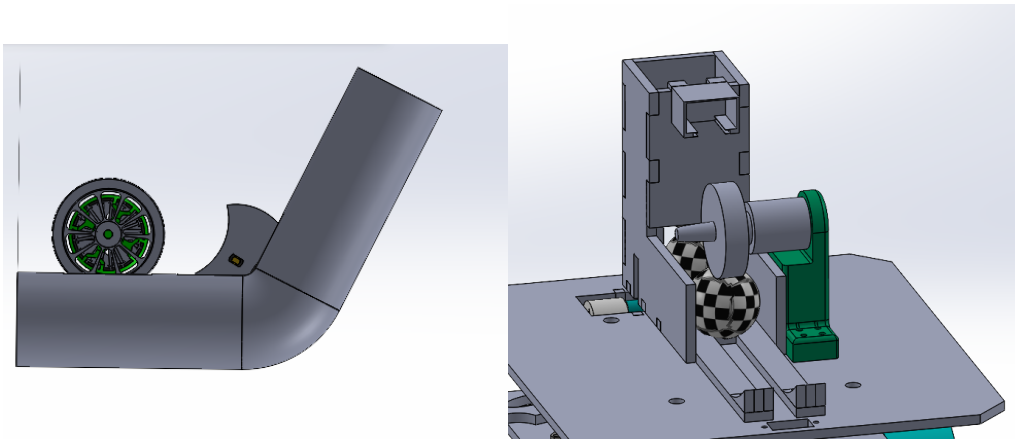


Figure 2: 3D Prints

2.3 Launcher: Old Vs New

Once we finished building the chassis, we started to focus on the launcher. We thought about changing from our old design to a curved tube shape, like you can see in Figure 2a. This could have been made from regular PVC pipes. But we ran into problems because we needed pipes of a certain size. After giving it some thought, we decided that this plan would be too much of a hassle to carry out.

Later, I made the launcher from a material called MDF. We were running out of time, so we quickly used hot glue to stick the launcher on the top level. This MDF launcher was made of five pieces that fit together. The front piece was meant to keep the balls in, but we left a space at the bottom so the balls could roll out. The two side pieces had a small tab to stop the stepper motor from hitting them. The back piece had a gap that let the "Scooper" move freely. The bottom piece was one solid part with a split in the middle to help the ball glide smoothly. At first, there was a tab in the middle, but we had to sand it down because it was causing problems.



(a) Old (b) Final Design

Figure 3: Old vs New launcher

3 Electrical

3.1 Navigation

Our robots navigation revolves around the collaborative usage of track wire detector, bumpers, tape sensors, and the 2k beacon detector.

3.1.1 Initial Goals and Functionality

Our preliminary design features seven unique sensors for localization: the IMU, ping sensors, motor encoders, a 2k beacon detector, tape sensors, track wire sensors, and bumpers. However, we figured that the ping sensors' response times may be too slow for the game, so we opted for the time of flight (ToF) sensors instead. Here, the IMU, three ToFs, and potentially the motor encoder sensors would easily assist in calculating the x and y coordinates of where the bot was by tracking the movement of the bot, and its distance from each wall and obstacles. With three tape sensors manning the front of the bot and one in the back, we could use that to know where the bot is on the field in regards to the field, so when it is in the center of the field where the black tape is in a cross shape, it can make adjustments and shoot at the goal. The track wire sensor would be a safety net, as well as the initiator of our movement from the reloading zone. To start off, the bot can move along the three point zone but never touch it until it meets another event. If the bot ever moved too far forward, it would never go into the penalty zone. Lastly, the 2k beacon detector ensures that the bot is moving towards the goal by allowing it to go when a signal is present, and the bumpers at the front and back of the bot served as collision detection, in which we would move directly away from the point of contact.

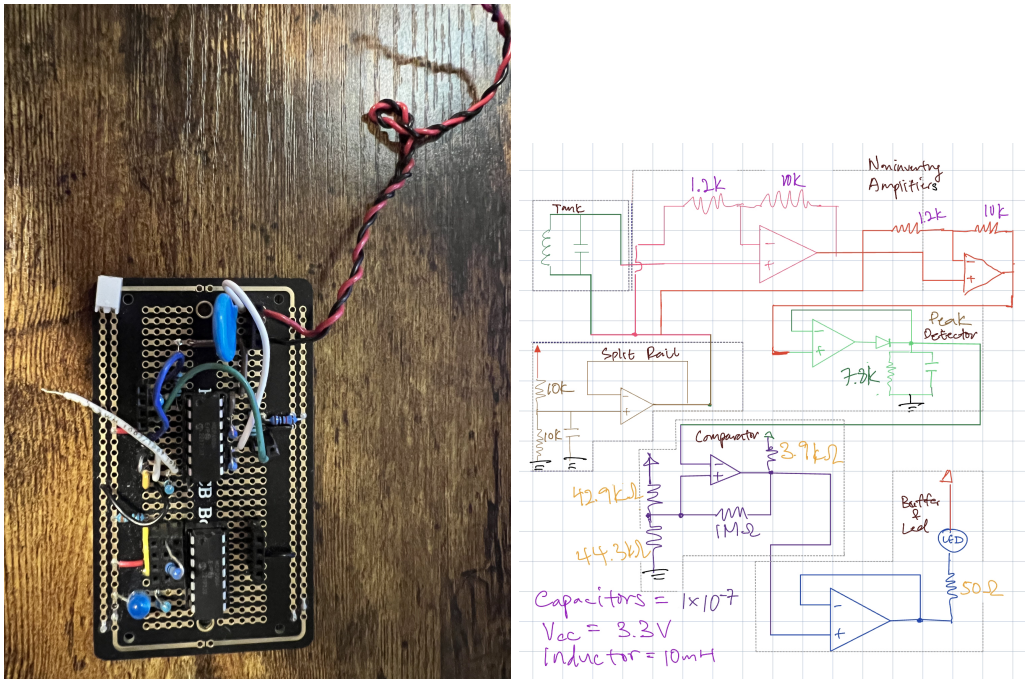
3.1.2 Deviations and Issues

Despite the high hopes, the ToF and IMU sensors took too long for us to fully operate, so we decided to drop them and maximize the usage of our other sensors. Losing these two sensors meant that our bot would be heavily reliant on the tape and track wire sensors for navigation. This brought up several issues as the those two sensors were not fully reliable due to their inconsistencies and limitations. First of all, the tape sensors were working as intended, but might have worked too well as it turned on and off due to the white text on the black tape of the field. Its over-precision would leave some difficult edge cases for us to solve later. Secondly, after spending way too much time tuning the track sensor, we figured out that our inductor for picking up a track wire signal was not effective after around four inches. Although reactive, we were afraid that the track wire may end up alarming us about the penalty zone too late, and that the bot might move too fast for it to detect the three point zone.

3.1.3 Circuit Building: Track Wire Sensor

Following the schematic below, a track wire sensor with a later implemented software hysteresis was built. Through two non-inverting amplifiers with resistor values of 1.2kΩ and 10kΩ, the initial signal from the 10mH inductor would be gained by 86. From here, I built a sharp peak detector with an RC of 0.00078 using a 7.8kΩ and $1e^{-7}$F to see how noisy the resulting signal would be. Between zero to four inches, the desired signal was shown on the oscilloscope, but from a distance further, the signal would change drastically. It was strange because the frequency would still hover from desired and quickly back to reading noise. A comparator and led was also built onto this perfboard to test how well it can send an on/off signal to the software. However, due to strange behavior beyond four inches, we opt for a software hysteresis instead for more stability.

From the image, the very top left features a power source connection, next to a tank circuit on the right. The top left and bottom right op-amp of the first chip are non-inverting amplifiers, the top right is a split rail, generating a 1.625V virtual ground for all circuits leading up to the peak detector on the bottom left. The output the peak detector would go into the negative terminal of the disconnected comparator, which would have a lower bound of 2.2V and an upper bound of 2.7V, lighting up the led when breaking through the upper bound. Due to time restraints, we could not fully implement the track wire.

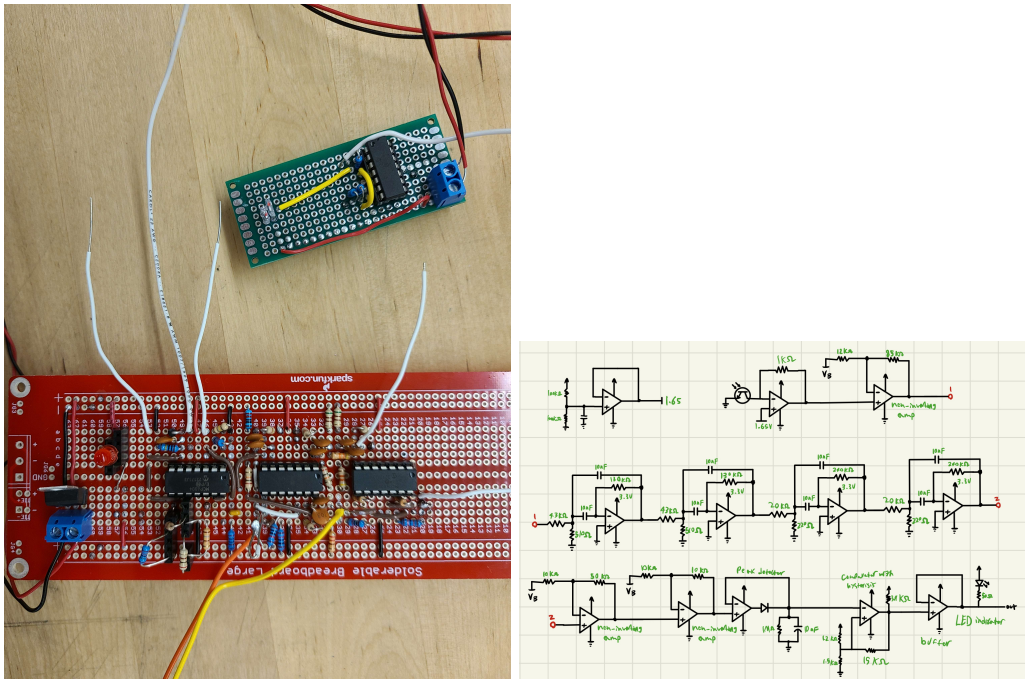


(a) Perfboard (b) Schematic

Figure 4: Track Wire Detector

3.1.4 2k Beacon Detector

The bread and butter of our robot begins with the 2k beacon detection. The entire circuit is split into two pieces for flexibility. On the green perfboard, two IR sensors are put in series to effectively double the size of the input signal. The chip on the board holds a split rail on the bottom and a phototransistor amplifier on top. The signal from that circuit is quite clean, so it is amplified through a non-inverting amplifier with a gain of 8 before being put into an 8th order Chebyshev filter. Furthermore, the resulting signal from the filter is then amplified with a gain of 12, in which its output is put into a peak detector. A strong and accurate comparator and hysteresis combination is used to turn the resulting signal digital, in which our software prompts our robot to activate when a 2.0k signal is detected.

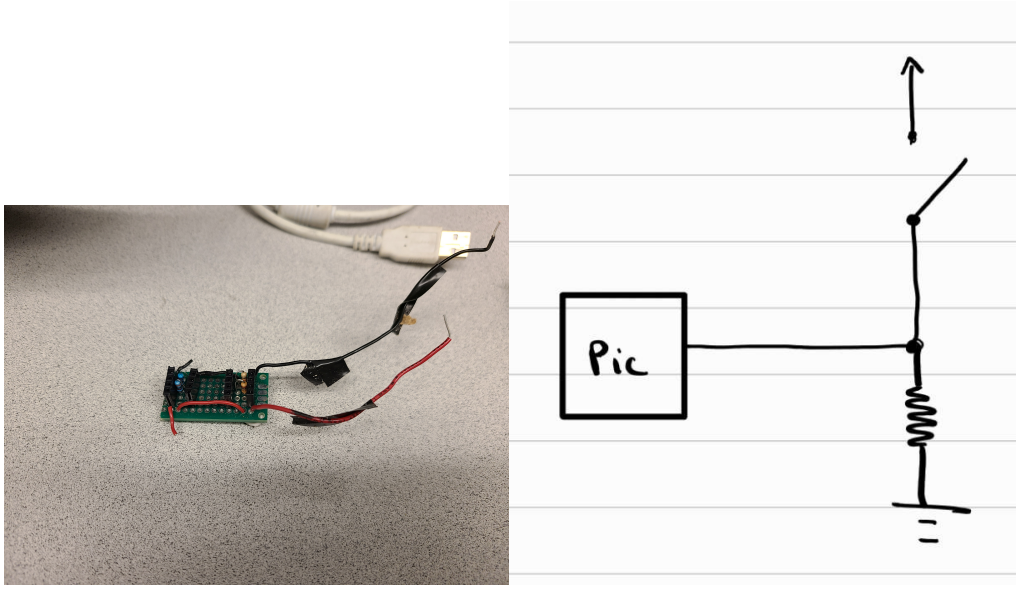


(a) Perfboard (b) Schematic

Figure 5: 2k Beacon Detector

3.1.5 Bumpers

Our bot uses standard limit switches as bumpers for collision detection. We thought bumpers were necessary despite already having a strong navigation system for the competition because the opposing robot may crash into us. For our switches we implemented hardware debouncing using a breakout board to lessen the load on the software.



(a) Perfboard (b) Schematic

Figure 6: Bumpers Breakout Board

3.2 Movement

Our movement features the popular mecanum wheels, which is capable of strafing side to side on top of its other motor abilities.

3.2.1 Mecanum Wheels

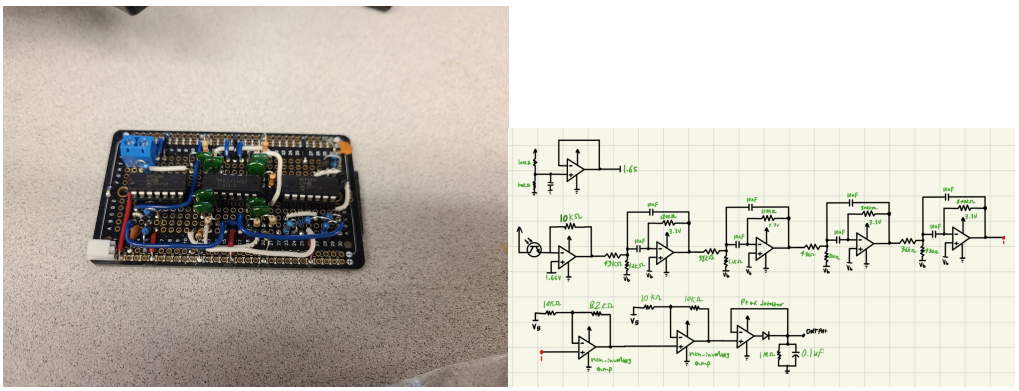
Using the power of trigonometry, the mecanum wheels are capable of going forward, backwards, turning left, turning right, as well as strafing left and right. On our robot, we connected four wheels to two H-Bridges, which occupied 18 ports on the PIC32. Each wheel would have a minimum PWM speed of 30, and the wheels on the right side would have its wiring for inputs A and B reversed, so we won't have to compensate for it in software. These wheels give the advantage of being able to always face towards the goal while navigating the field, while still being able to make minor adjustments as needed. In addition, being able to move in all directions allow us to follow the tape a lot better.

3.3 Launcher

Our launcher system features a 1.5k beacon detector, incredible DC motor, ESC, and stepper motor.

3.3.1 1.5k Beacon Detector

Initially, we planned to utilize a powerful LiDAR to keep track of the goalie. We would simply sense a further distance when Sammy has moved away, and then shoot. However, due to time restraints, we decided to use a 1.5k beacon detector instead, where our robot would scan for a 1.5k frequency signal, face away from it, and shoot. The 1.5k beacon detector features a software hysteresis, which sends a signal for launcher activation at just behind the one point line. These circuit features an 8th order butterworth filter, but stops at the peak detector and utilizes a software hysteresis.



(a) Perfboard (b) Schematic

Figure 7: 1.5 Beacon Detector

3.3.2 Flywheel and Stepper Motor

Our most important aspect of our launcher system is our flywheel with our DC motor, and our stepper motor feeding mechanism. The flywheel is connected to an ESC, which is connected to our arduino, which allows us to control when the launcher activates. This allows our strong flywheel to launch ping pong balls at extremely high speeds. Our feeding mechanism was not as impressive, however, as the stepper motor did move as fast as intended. Using an H-Bridge, the stepper motor's speed were capped due to hardware limitations. To compensate, we adjusted the stepper motor to feed all three ping pong balls at once and let the flywheel take care of the rest.

3.4 Wiring and Power Distribution Board

Our bot design contains three separate layers. In its bottom layer, we have all 4 tape sensors, motors, two H-Bridges, and our three ToF sensors (which were not used). We drilled holes in the second layer to allow wire connections between the bottom and middle layer. Majority of the connections went to the PIC32, with the exception of the ToF sensors, which were connected to the Arduino ESP. The middle layer is our brain, containing both our microcontrollers (PIC32 and ESP32), bumpers, power distribution board, and bumper breakout board. The top layer contains all of our sensors and hardware required for the launcher mechanism: the flywheel, ESC, H-Bridge, and H-Bridge, as well as all our beacon detectors. Strapped to the bottom of our top layer was the battery and track wire sensor.

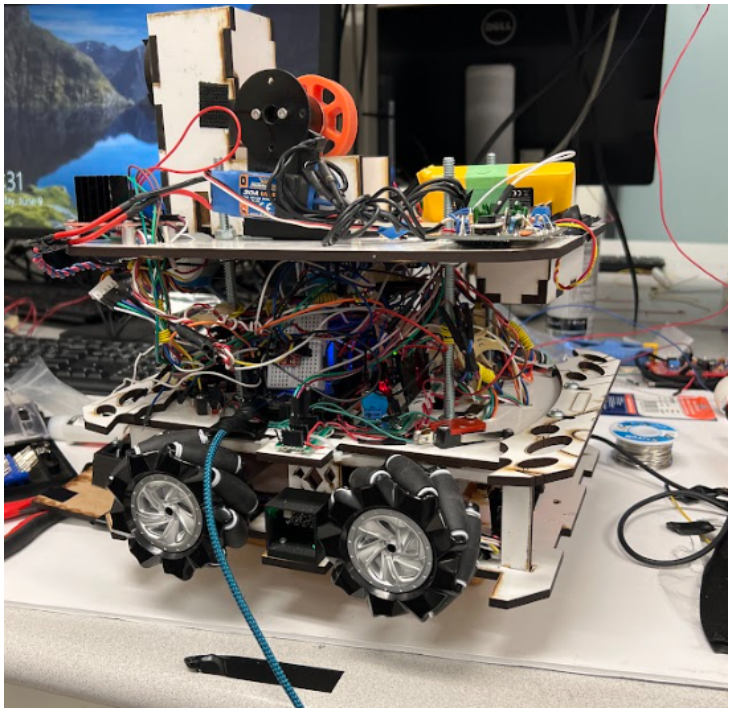


Figure 8: Three Layers

All 3.3V required circuits, such as the bumpers, track wire sensor, and the 1.5k beacon detector, connected to the 3.3V side of the power distribution board. All 9.9V connections, such as the power distribution board itself, and 2.0k beacon detector, were connected to the UNO32 power distribution board. 5.0V connections, such as all the H-Bridges and the stepper motor were also connected to the power distribution board.

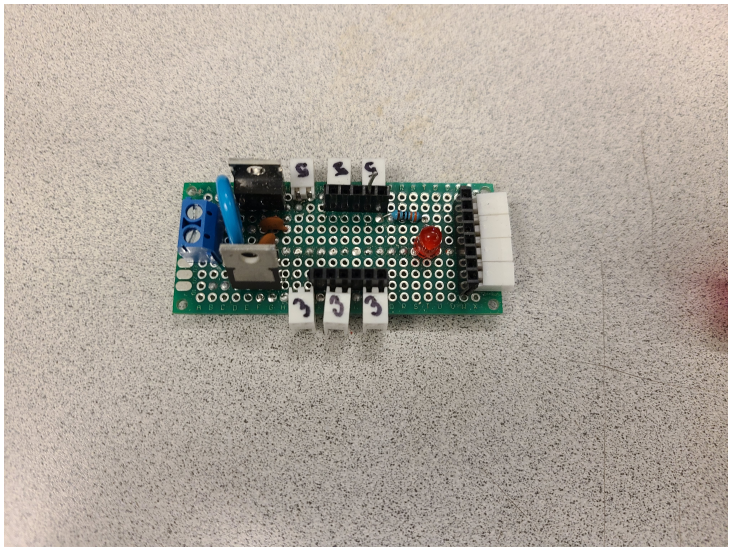


Figure 9: Power Distribution Board

4 Software

4.1 First Draft: State Machine 0.5 and Localization System

The first design iteration of the BOAT was planned to incorporate a two-dimensional localization framework leveraging three Time-of-Flight (ToF) sensors (one on each side and the third on the rear of the bot) at a height below the top of the walls of the field. These ToF sensors would be incorporated using a multiplexed I2C connection to the onboard ESP32 microcontroller, which would handle pulling the ranging measurements from each sensor and predicting the current location of the robot on the field using a combination of facts known from the project prompt (the field's dimensions are known, the bot will always start in the reload zone, the tapes on the field do not move) to handle any potential drift over time.

The first iteration of the BOAT state machine can be observed in the figure below:

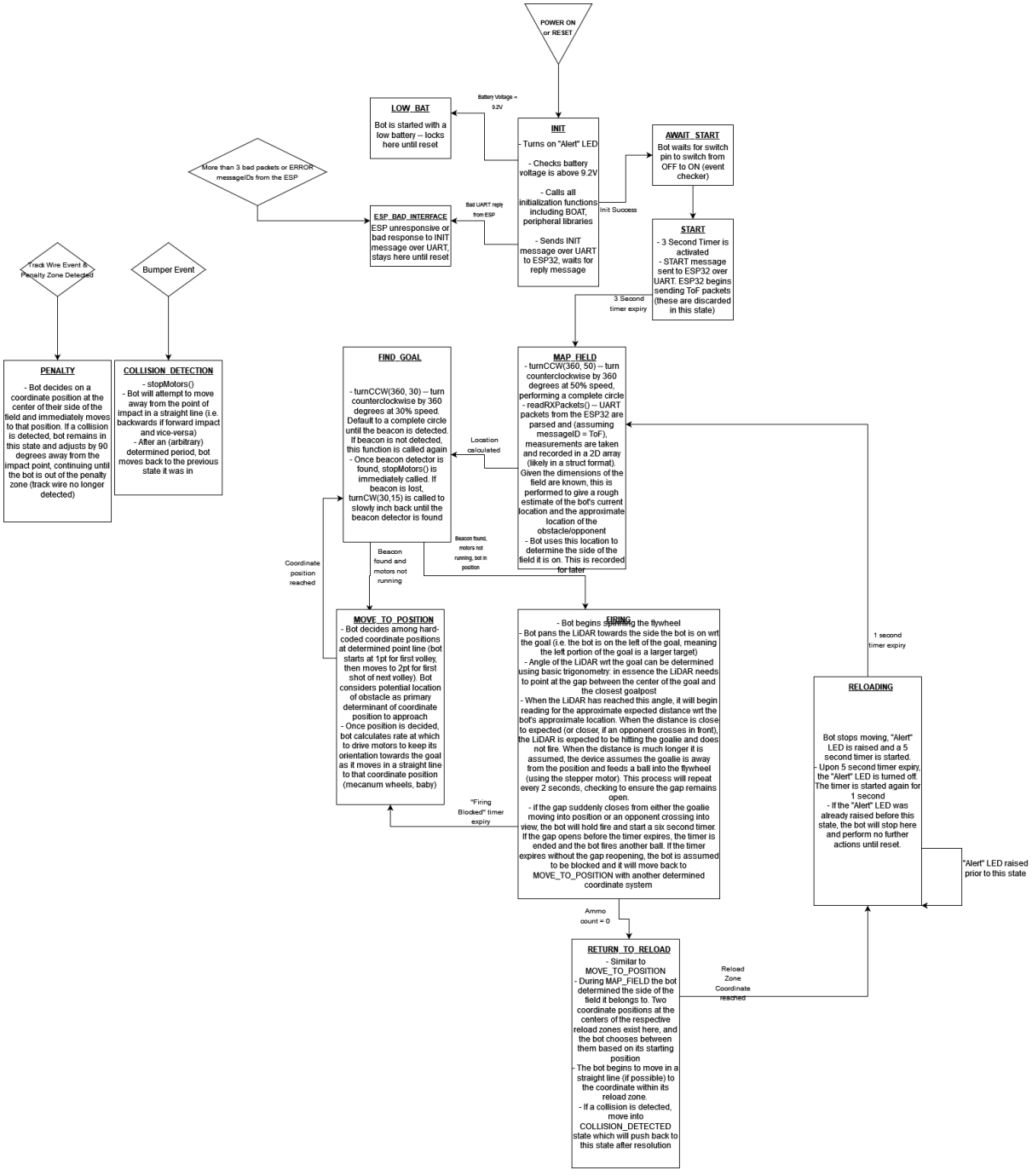


Figure 10: BOAT State Machine 1.0

Upon power on the BOAT would enter its INIT state, where it would activate its respective libraries (launcher protocol, sensors, movement systems, etc) and send a UART handshake to the ESP32 microcontroller (UART will be discussed in section 4.2.1). If a library returns an error during initialization, the battery is determined to be too low (below 9.2V) or the UART handshake fails, the system will lock itself into an INIT FAILED state where it will light an alert LED (located at the top of the bot) and perform no other actions until the system is reset. During design this was intended as a catch-all for potential bugs or other errors present during initialization that would prove detrimental to the bot.

If no errors occur during the INIT state the bot will move on to the STANDBY state, where it performs no other actions until the start switch is set from OFF to ON. At that point, the bot will run a three second counter (allowing the user setting the switch to pull their hand away) before moving to the ORIENTING state, which simply spins the bot counterclockwise until it receives a signal on its 2kHz beacon detector (a '2k event' in the ES framework). Once this signal is received the bot then reads the ToF sensor on its rear as it pans slowly back and forth searching for the minimum value it can find (this is assumed to be directly perpendicular with the wall, thus facing the bot forward and parallel to the side walls). This orientation in tandem with the mecanum wheels can be well-leveraged as the bot maneuvers along the field and targets a specific area of the goal.

In MOVE TO POSITION, the bot enters its first sub-state machine. Here, the bot approaches within a few centimeters of its closest wall (at this point taking note of the side the bot started on) while keeping its orientation, then moving forward and following the wall. The rear ToF sensor takes consistent measurement of the distance traveled by the bot (its maximum range is 4 meters, which is plenty more distance than the length of the field) and keeps alert for when the bot reaches a specific distance from the rear wall (within the 1pt zone). If a collision occurs during this travel, the bot backs away from the collision and strafes to the opposite wall before continuing to move forward. Once the pre-determined distance from the rear wall is reached, the bot stops moving, then strafes away from the wall roughly sixteen inches so that it is directly facing one of the open sides of the goal (not consistently blocked by Sammy). If a collision occurs here, the bot will maneuver away from the collision and turn in that direction until the 2k beacon is detected, then turn away for a moment to aim "roughly" at an open gap.

In FIRING, the bot is no longer moving (to prevent current overdraw and burning out fuses) and begins to spin its flywheel in the launcher system. The 1.5kHz beacon detector is utilized to determine when to spin the stepper motor (the feeding mechanism) and when to wait (in essence, ping-pong balls are only fed when Sammy is not blocking that section of the goal). After three shots are fired (roughly 5-10 seconds of spinning the stepper motor), the flywheel is stopped and the bot moves to its RETURNING state where it returns to its closest wall and follows the wall all the way back to the reload zone. If the bot experienced a collision that caused it to move to its opposite side, the bot will then strafe to its respective side in the reload zone. Finally, its RELOADING state is activated where it makes no movements and simply operates a three second timer.

4.2 PIC32 and ESP32 UART Interface

4.2.1 UART Packet Protocol

Utilizing two separate microcontrollers creates extra challenges in integrating a comprehensive robotic system, the primary challenge being how these microcontrollers communicate with one-another. While there are various well-defined protocols that exist, it was decided to use a packet-based UART protocol based directly on prior coursework in ECE 121: Microcontroller System Design. In that course this protocol was used to interface a PIC32 with a python-based PC application using a USB connection leveraging UART. The packet structure was very simple and can be best depicted by the figure below:

HEAD	LENGTH	(ID)PAYLOAD	TAIL	CHECKSUM	END
1	1	(1) <128	1	1	\r\n

Figure 11: UART Packet Structure (courtesy of ECE 121, Lab 1 Document)

Here, each number in the figure indicates the number of bytes involved in each section of the packet. Each device would have an interrupt dedicated to processing any incoming bytes on their respective UART RX register, and when they weren't actively processing a new packet they would read and discard each incoming byte until a "head" byte (hardcoded to be the hex value 0xCC) was detected, which would activate the packet read protocol. This discarding of unwanted bytes at the beginning has the added value of allowing for both devices to use the same UART used by their USB and console interfaces, provided no print statements are executed while they send a pre-built packet over their UART TX register.

Once a head byte is received the next byte is expected to be a valid length (in binary) between 1 and 128, which depicts the expected length of the payload (next series of bytes). The payload is led by the message ID, a defined value (a separate messageIDs.h library was created to differentiate these) that directs the receiving microcontroller on how to process the incoming packet, followed by the actual data of the packet. The tail byte, hardcoded to be 0xB9, denotes the end of the payload, and the checksum (independently calculated by the receiver) is cross-checked for packet integrity. If the checksum sent by the transmitter and that calculated by the receiver are different, the packet is discarded. Otherwise, the packet is ended with a carriage return and new line character.

4.2.2 Integration Between Both Microcontrollers

This UART interface was designed to be simple and incorporate as little packet variety as possible. In essence, this interface would begin with a handshake protocol where the PIC32 would send a handshake to the ESP32 (which upon initialization would simply wait for the handshake) containing a set of four hard-coded bytes. The ESP32 would receive and process the packet, divide each byte by 2, and return these bytes as a separate handshake packet. The PIC32 would read these bytes and checksum, and if it reads the expected values (again, hardcoded), it will denote the handshake as successful. From here, the ESP32 would begin to send periodic "ranging" packets containing the most up-to-date values of the ToF sensors (each a 16-bit value) and the approximate two-dimensional coordinate location of the bot at that given time (a separate 16-bit value for X and Y).

4.3 Second Draft: Revisions and Scrapping of Advanced Features

4.3.1 Time Constraints and Challenges

Electrical development of the bot proved to be a significantly-larger time sink than originally anticipated, which provided a great deal of additional challenges to software development. In essence, resources (namely, time) needed to be pulled from software development to ensure the electrical systems would be developed in time for hardware debugging, integration, and calibration. This unfortunately led to the situation where the software design was harshly neglected until less than a week before the deadline – given the nature of the originally-enthusiastic considerations of a comprehensive UART system integrating two microcontrollers to utilize a localization framework in real-time, multiple features ended up having to be scaled back considerably.

4.3.2 Revisions and Cuts to Final Product

The first major scale-back was the UART communication system. While this directly translated to the PIC32 from a prior library written in ECE 121 and worked without issue, and while a UART connection between the two microcontrollers was verified in sending individual bytes over the connection, building the protocol on the ESP32 provided a great deal of roadblocks not initially anticipated. Had the software portion been given more time this almost certainly would have been ultimately developed, however after multiple failures and the knowledge that the localization system was not even prototyped, let alone functional, it was decided the software suite would rapidly change course and do away with the UART system. This meant the ToF sensors, while still providing measurements to the ESP32, would no longer be usable to the PIC32 (a solution was attempted using a logic level converter, however it was discovered these are only effective with digital signals and not analog ones so this was scrapped as well), meaning the advanced localization capabilities envisioned for the bot would not come to fruition.

The boon that saved the bot was the team's insistence on incorporating the standard tape sensors, track wire sensors, and bumpers so as to not "put our eggs in one basket" as it were. This allowed us to fall back on these systems when the more advanced framework failed and devise a simpler, albeit considerably-less robust, state machine that would still fulfill the requirements set by the project outline of the course.

The ESP32, meanwhile, would operate its own small state machine with two states: a RANGING state that would read the ToF sensors, and a LAUNCHER state that would run the flywheel and feed ping-pong balls to the flywheel for ten seconds.

4.4 State Machine Final Version

The below figure denotes the state diagram for the final state machine of the BOAT:

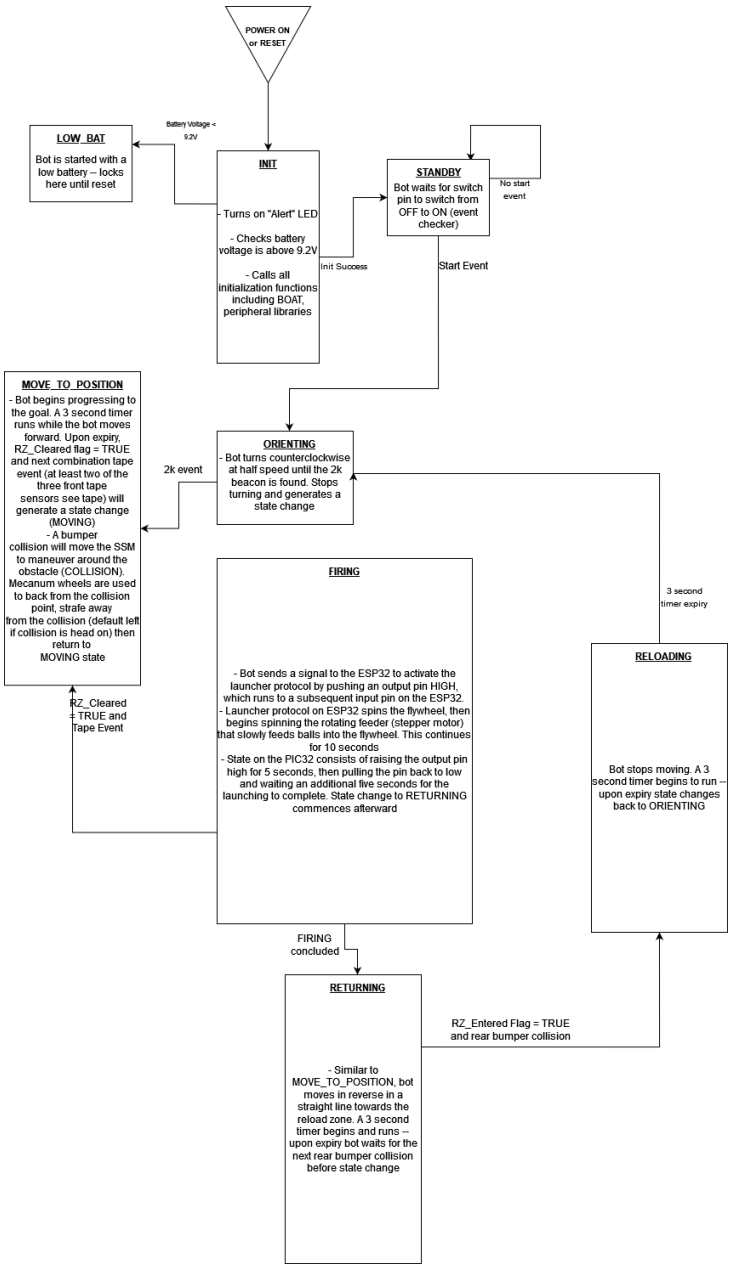


Figure 12: Final State Machine Diagram, BOAT

As with the first version, upon power-on the BOAT begins in its INIT state where it initializes its libraries (similarly, on power-on the ESP32 will similarly run through its startup sequence, which is important for the ESC attached to the flywheel which requires a seven-second startup signal). Here, if a library fails its initialization or the battery is below 9.2V the state will switch to an encompassing LOW BAT state which will light an LED next to the launcher and remain in that state (with no further action) until the bot is reset (ideally with a new battery). Else, the bot will move to the STANDBY state where it will wait for the start switch to be activated.

Upon activation of the start switch the bot will move into its ORIENTING state, which simply has the bot spin counterclockwise until the 2k beacon is found. At that point, the bot stops its motors and moves into the MOVE TO POSITION state, where it moves directly towards the goal. A three second timer is activated and runs while the bot moves forward – upon expiry the next tape sensor event covering more than one front tape (indicating the 1pt line has been reached) will move the state to FIRING. If the bot encounters a collision during this phase, the timer will be stopped (if still running), the bot will reverse and strafe from the point of impact (default left if head-on collision), and begin moving forward again until the 1pt line is detected.

In the FIRING state the PIC32 simply raises one of its digital pins to HIGH – this is connected to an input pin on the ESP32, which reads this and transitions its own state machine to the launcher protocol. Here, the ESP32 performs the heavy lifting and spins both the flywheel and stepper motor for ten seconds while the PIC32 runs a similar timer to give the launcher adequate time to fire its ping pong balls. The flywheel, while employing a powerful drone motor, is driven at a very slow speed sufficient to "bounce" the balls into the goal, which proved significantly more reliable than attempting to fire directly into the goal.

After ten seconds, the bot moves into its RETURNING state where the PIC32 reasserts control and puts the bot into reverse. Similarly to the MOVE TO POSITION state, a three second timer begins and elapses while the bot is in reverse. If a collision is detected here, the bot similarly moves and strafes away from the impact and continues reversing. After three seconds the next rear bumper event will be considered to be the rear wall (reload zone). If the bot had a prior collision it will then strafe in the opposite direction than it did during object avoidance once it reaches the reload zone so that it is in its correct portion of the zone. Finally, the bot enters the RELOAD state here which simply operates a three second timer and performs no other actions until the timer expires, allowing for the user to reload the ping pong balls for the next volley.

This state machine is significantly less complex than the initial design version and did not cover for every edge case, however it proved sufficient enough to perform the minimum specifications detailed in the project guidelines. This was, also, far quicker to implement, which was critical in the aforementioned minimal time allotted to the software portion of this project.

5 Conclusion

The BOAT is a comprehensive autonomous robot system with the sole purpose of performing under the rules and constrictions of the "Slug World Cup" competition and field. By leveraging the capabilities of its tape sensors, bumpers, beacon detectors, and mecanum wheel drive this robot was ultimately successful in performing the minimum requirements of maneuvering the playing field and scoring three goals in two minutes. Though the end result is a success, however, that success is bittersweet: the initial vision of the BOAT involved incorporating three Time of Flight sensors and a one-dimensional LiDAR to effectively leverage the capabilities of the ESP32 microcontroller to run a real-time, two-dimensional localization system tracking the bot's most likely location at any given time. The ESP32 would have performed the bulk of the localization processing and sent periodic UART packets to the PIC32 containing up-to-date coordinate information. However, due to various challenges discussed throughout this report causing the project to slow drastically, these capabilities had to be scaled back dramatically and late in development. The BOAT's ultimate performance, while successful, suffered as a consequence and its ultimate software suite was far simpler and less robust than initially desired.

With all this said, this project proved to be an immense learning experience about project management and integration of various skill-sets into producing a comprehensive and complex system to satisfy design objectives. For many students in this Mechatronics course including members of this team they began their work with no prior experience in a robotics project anywhere approaching the scope demanded here. The resulting toolkit, both physical and mental, will be invaluable in future academic works as well as in the broader industry.