ECE118: Mechatronics

Final Project:
The Spy Who Slimed Me

Brian Naranjo, Manuel Lira, Jornell Quiambao
(brnaranj@ucsc.edu)(mlira2@ucsc.edu)(jquiamba@ucsc.edu)

Professor: Michael Wehner
TA: Tony Li and Zach Potter
December 7, 2019

Contents

1 Lab Goal **3**

2 Design Brainstorming **3**
2.1 Preliminary Design One: The Side Shootah . . . 3
2.1.1 Design Flaws . . . 5
2.2 Preliminary Design Two: The Cerberus . . . 6
2.2.1 Design Flaws . . . 8
2.3 First Draft of State Machine . . . 8
2.3.1 Top Level . . . 8
2.3.2 SMART_SEARCH Sub State Machine . . . 9
2.3.3 Find Side Sub State Machine . . . 10
2.3.4 Shoot Sub State Machine . . . 12

3 Electrical Components **13**
3.1 Beacon Detector . . . 13
3.1.1 Difficulties and adjustments . . . 14
3.2 Trackwire Sensor . . . 14
3.2.1 Difficulties and adjustments . . . 15
3.3 Bump Sensors . . . 16
3.3.1 Difficulties and adjustments . . . 16
3.4 Tape Sensors . . . 17
3.4.1 Difficulties and adjustments . . . 18
3.5 Ping Sensors . . . 18
3.5.1 Difficulties and adjustments . . . 19
3.6 DC motor and Stepper Actuation . . . 20
3.7 Complete Bot Integration . . . 21
3.7.1 Difficulties and adjustments . . . 22

4 Mechanical Design **22**
4.1 Overview . . . 22
4.2 Base/First Platform . . . 22
4.2.1 Difficulties and adjustments . . . 23
4.3 Skids . . . 23
4.3.1 Difficulties and adjustments . . . 25
4.4 Floor Tape Sensors . . . 25
4.4.1 Difficulties and adjustments . . . 26
4.5 Bumpers . . . 26
4.5.1 Difficulties and adjustments . . . 27
4.6 Motor Mounts . . . 27
4.6.1 Difficulties and adjustments . . . 28
4.7 Second Platform . . . 28
4.7.1 Difficulties and adjustments . . . 29
4.8 Third Platform . . . 29
4.8.1 Difficulties and adjustments . . . 30
4.9 Barrel/Barrel Mount . . . 30
4.9.1 Difficulties and adjustments . . . 33

4.10 Carousel Mechanism . . . 33
4.10.1 Difficulties and adjustments . . . 35
4.11 Fully Assembled Bot . . . 35
4.11.1 Difficulties and adjustments . . . 37

5 Software Implementation **37**
5.1 Test Harness . . . 38
5.1.1 Blocking Code . . . 38
5.1.2 ES Framework – Event Checker . . . 39
5.1.3 Difficulties and adjustments . . . 39
5.2 Libraries . . . 39
5.2.1 Difficulties and adjustments . . . 41
5.3 Ping Sensors . . . 41
5.4 Final State Machine . . . 42
5.4.1 Beacon Detection and Tape Avoidance Process . . . 43
5.4.2 Beacon Found and Bumping Process . . . 44
5.4.3 Tower Navigation and Trackwire Process . . . 44
5.4.4 Sweeping and Finding Hole Process . . . 45
5.4.5 Shooting a ball Process . . . 45
5.4.6 Escaping and Finding Next Beacon Process . . . 46
5.4.7 Difficulties and adjustments . . . 47

6 Billing List **48**

7 Videos **48**

8 Conclusion **48**

9 Acknowledgements **49**

1 Lab Goal

This year's Mechatronics final project is James Bond themed and is appropriately titled "The Spy Who Slimed Me". We were tasked with using all the skills we learned throughout this course to construct an autonomous robot that can locate three beacons, navigate to them without crossing tape barriers, detect the correct one of three total sides, and deposit a ping pong ball inside the hole thus disabling the beacon. The constraints for this project included ensuring our bot fits in a box with dimensions 11'x 11'x 11', and we spend no more than $150 on all parts for the robot. Building this project was a long iterative process as will be discussed throughout this report. For clarity, the complete bot design is described in terms of its electrical circuity, mechanical construction, and software implementation within individual sections.

2 Design Brainstorming

sec:brainstorming The first stage within our deign process was focused on evaluating potential bot designs and considering which mechanical design would be most effective in navigating the towers and shooting a ball through the correct hole. When discussing potential design features, we focused primarily on mechanical aspects and state machine implementation as we thought of a proper approach to navigate around the field and traverse the target. The prototype designs detailed in the following subsections display key design features and state machine concepts which inspired our final robot design.

For our initial prototyping process, we focused more on planning the shooting and reloading mechanism as well as the sensor layout as these are the integral parts for the bot functionality. We decided that it was best to use our available time with the TA's to receive useful feedback and refine our design for the shooting mechanisms and leaving the smaller parts for later. The shooting mechanisms were different between our two preliminary designs just in case one was an issue, than we could easily adjust to the secondary method.

One issue we had when coming up with a robot design was finding a way to make it as compact as possible to allow space for a large ammo capacity. However, we also determined that a better method was just to ensure our design is very precise so that not much if any excess ammo is required. The two shooting methods featured on our preliminary bot designs includes a single solenoid powered shooter, and the other a gravity fed, servo controlled, ramp shooter that will guide the balls from the feeder to the 8" tall hole. The preliminary designs are discussed in more detail below.

2.1 Preliminary Design One: The Side Shootah

This design implemented a circular base to easily cut corners and align parallel to the wall where it would use a side-mounted solenoid based shooter to fire a ping pong ball directly through a tube at a height of 8" from the ground. The idea behind the side-mounted shooter is that once a bump is detected, the bot can orient itself parallel to the wall, where traversal around the target is made much more simple. Then if the right side is detected, tape detection can be made as simple as merely driving forward and backward along the wall until the tape is detected. The reloading mechanism is based on a spring system which would be compressed by the weight of the balls so each time that a ball is fired, less force will be placed upon the spring and it will push the available ammo upwards. After estimated how much vertical space would be required for mounting the Uno and power distribution board, we found that with this reloading mechanism, we were looking at a maximum ball capacity of 4 ping pong balls with a standard 1.5" diameter.

As for the sensors, a pair of tape sensors are attached to the outside of the shooter, directly under the cannon and separated by a distance slightly smaller than the width of the tape in order to ensure that the correct hole is detected and the ball is consistently shot through the target. The beacon detector is located at the top of our bot to remain as close to the same height of the beacon as possible while remaining within the cube of compliance. For the bumper designs, we were thinking about going with whisker type bumpers that extend from the center bumper and spread past the bot so that if the bumper is not pressed, we can easily clear turns, yet still be able to detect collisions and resolve them without getting stuck. The trackwire sensor will also be mounted somewhere on the side shooting mechanism so that once parallel, we can check for the trackwire side. In order to properly align our bot so that we are parallel with the wall, we will be using two side-mounted ping sensors to detect when both sensors are getting a close reading and the bot is within some threshold of being close to parallel. The ping sensors would also be useful when traversing around the target.

Some sketches outlining the internal mechanism and component arrangement are shown below. The first figure, Figure 1, is a front view outlining how the balls will be spring-loaded within the vertically extended barrel and the general look of the bot. The second figure, Figure 2, depicts a top view of this bot design, where it is more apparent where all of the sensors and drivers are located. The last figure, Figure 3, is a side-view outline of the bot design, showing how the tape sensors will be mounted on the side shooter as well as below the bot.

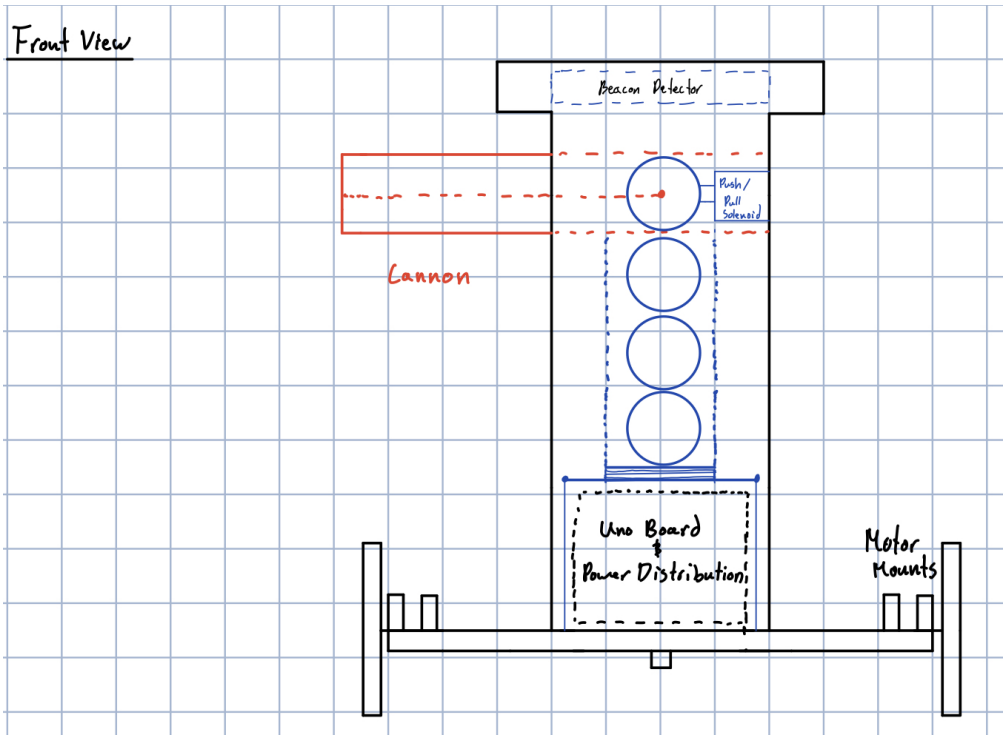


Figure 1: *Design One: Front View*

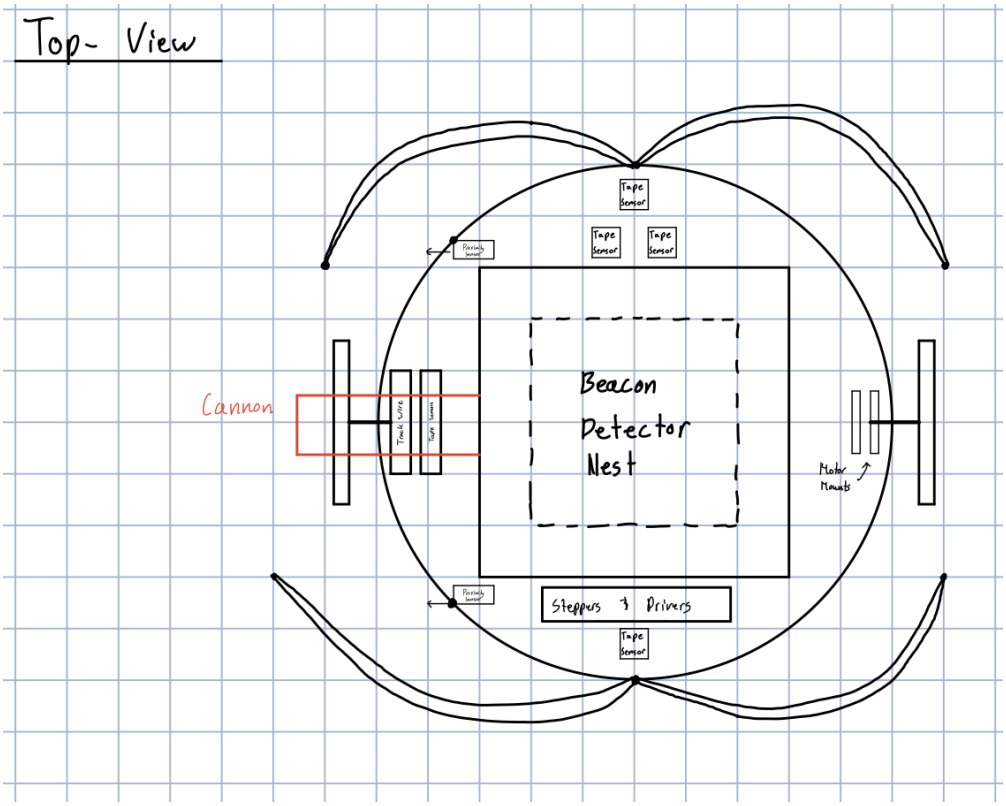


Figure 2: *Design Two: Top View*

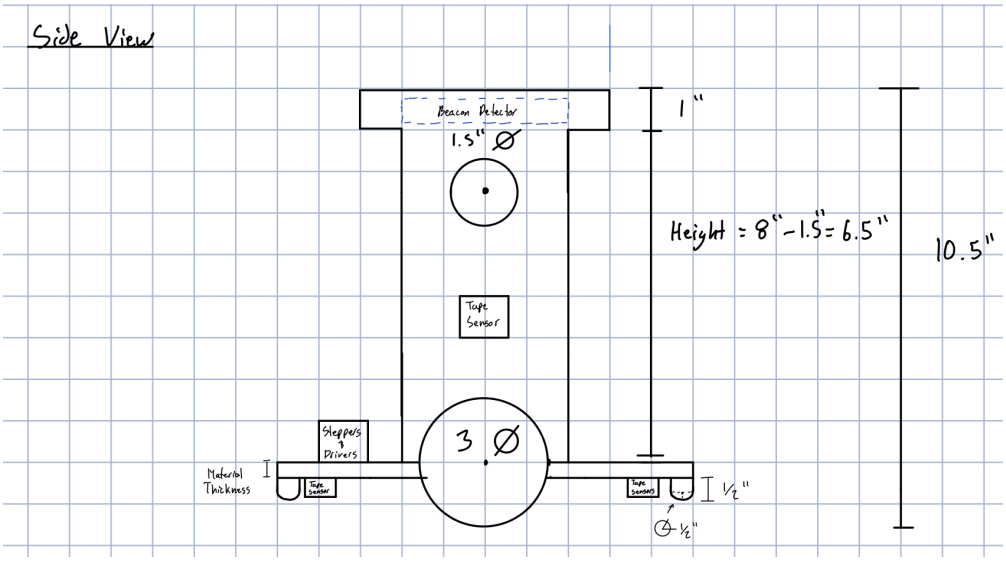


Figure 3: *Design One: Side View*

2.1.1 Design Flaws

When presenting this prototype design to the class TAs and tutors, we were told that although the spring loaded mechanism could work to push balls up to a shooter, we could run into many issues,

and the solenoid wouldn't be able to push the ping pong ball with enough force to launch it out of the spring's tension when fully stretched out. Also, due to advice from another tutor, we were advised to plan for missing no balls but be a larger capacity is always better because you never know if only one more ball would be needed for that check off.

For this reason, we decided to go with a gravity fed launcher with a horizontal based ball reloading mechanism to allow for a larger ball magazine. Many groups seemed to also go with a side-mounted shooting design, which makes sense because for this task it seems way easier to traverse the tower by driving straight and turning rather than having to back up and turn every time. Besides the flaws observed after talking to the judges, this design had many good concepts that carried on to final bot design.

2.2 Preliminary Design Two: The Cerberus

For our alternative design, we sketched up a three-ramp shooting system which would be gravity fed and use a servo to control the shooting of the ball. The bot would consist of a rectangular base with an equivalent width for the front of the bot so that once aligned, all of the shooting tubes will be perfectly aligned with the holes on that side of the wall. The idea behind this shooting mechanism would be that once a side is bumped, the side would be checked for a trackwire signal, where if not detected it would have to traverse the tower. Once it is detected, the bot would keep driving into the ball, backing up and pivoting, until the bot is aligned with the wall. Then once the tape sensors are aligned to the holes of the right side of the target, the correct hole is detected and that specific servo would turn on thus dropping the ball.

The bot would be multilayered to provide space for each component to be mounted, where the whole top layer would be dedicated to the ball gravity ball slide launcher, extra balls, the servos to control shooting, and the beacon detector. After estimating the total clearance required to hook up the drivers and UNO board, we determined that we had around 3.5" height to work with for the top layer of the bot. For this design, space seemed to be a key limiter, considering than a larger portion of it would have to be to be dedicated to the servos, the slides, and the balls on each slide.

As for the sensors, the bump sensor and tape sensor designs are implemented here again, however with this rectangular base the tape sensors would have to be separated out more and three tape sensors would be required for the gun rather than two on the side shooter. The trackwire would also have to be mounted facing towards the front of the bot instead of towards the side for the side-shooter design. In order for better detection of the trackwire, the outline is shown to use two trackwire sensors at each end of the bot to ensure that no corner case is ran into and when both edges are detecting trackwire, we are directly in front of the right side of the target.

Some sketches showing the general outline of the ramp system as well as the component layout can be seen below. Shown in the first figure, Figure 4, the three ramp design is clearly shown, where the tape sensors are attached directly underneath each shooter as well as the trackwire sensors and the UNO distribution board. In the second figure, Figure 5, The control for the ramp shooting system is shown, where a small incline is used to hold three balls and spin stepper motor or 360 degree servo until a single ball is released and the second ball is blocked. After estimating the total dimensions of the rectangular bot, we determined that we could hold around three 1.5" ping pong balls for each slide which would be helpful if we were looking to shoot three beacons and we run into the odd chance that all three beacons have tape on the right side.

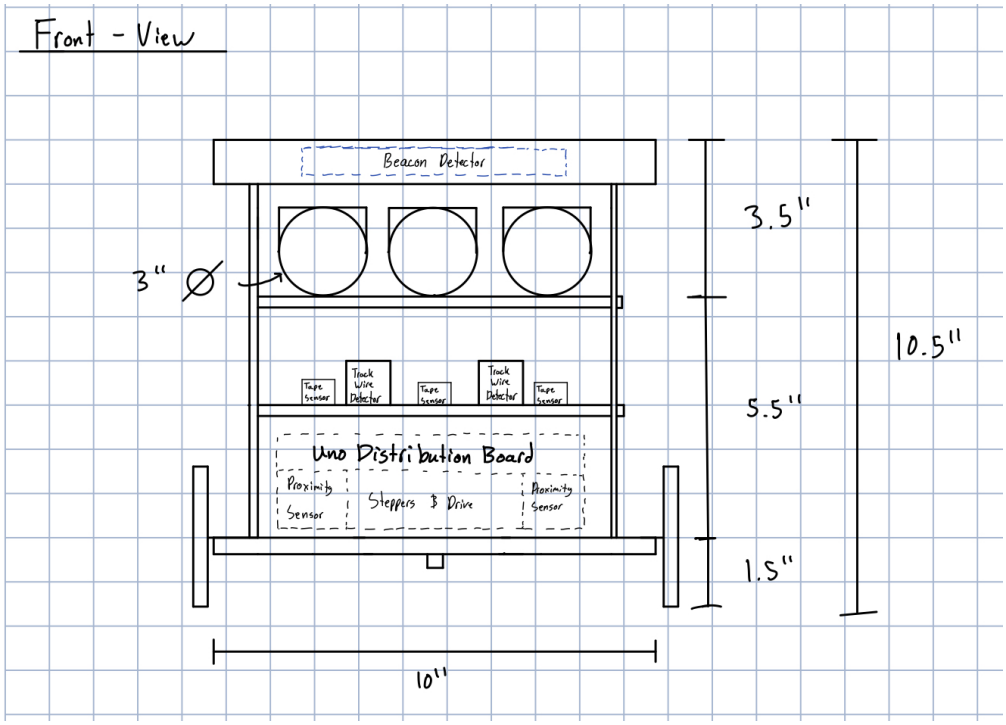


Figure 4: *Design Two: Front View*

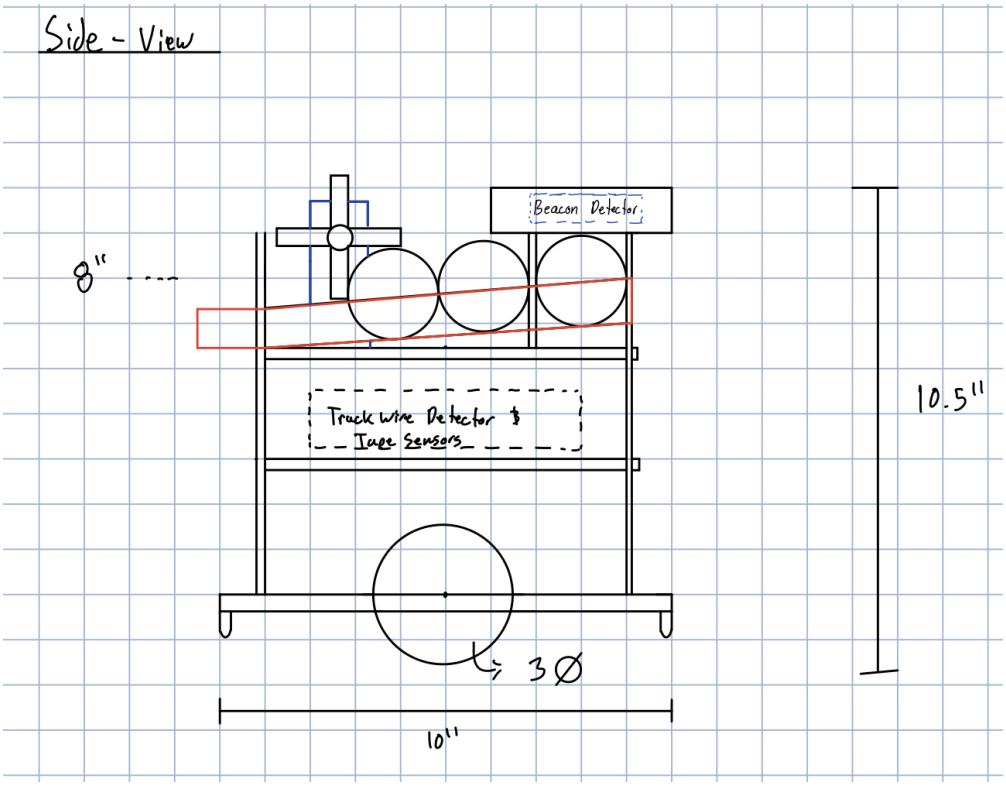


Figure 5: *Design Two: Side View*

2.2.1 Design Flaws

When demonstrated to the TA's and tutors, during the preliminary design review we were asked how we were planning to traverse the tower with the forward shooting mechanism, since we would have to use alot of timer based events to have to back up turn drive the next side and turn to bump straight into the wall again. Although, they did not critique the stepper and gravity based ramp design, we found that putting three servos on top of the bot would be difficult to do considering how big these components are and how little space there would be in between each of the slides. Taking this two issues into consideration, we looked into using the positive aspects of this design and combining it with those of the previous design.

Therefore, we found that for our initial bot design, we would start with a gravity based shooting system rather than solenoid actuated since speed was not required to put the ball through the hole. Also the shooting system would consist of only a single ramp system that is side mounted to allow for easier maneuvering around the tower and max ball capacity. As for the weaker aspects of this design, most of the mechanical framework for this bot was scrapped, since it made traversal around the target difficult with its bulky rectangular base and large shooting platform, which could potentially get stuck on the wall or corner.

2.3 First Draft of State Machine

As we discussed possible design features for our bot, we were routinely thinking about our approach to the task as we simulated the course and brainstormed ways on how we would control the robot's behavior so that it is programmed to locate the right hole as fast and accurately as possible, and it can use its shooting mechanism to consistently feed a ping pong ball through the target. For our initial state machine design, we went for a simple two-level hierarchical state machine design. We initially believed that the robot's movement was simple enough, that we can compact the full functionality in two levels, however we observed throughout the design process that there was extensive revisions to be made.

The full prototype hierarchical state machine including its sub state machine implementation is outlined and sketched in the figures below, where the final implementation was discussed further in Section 5.

2.3.1 Top Level

For the top level of the state machine, outlined in Figure 6, the general behavior we tried to implement begins with the initial states DETECT_BEACON, where our robot will be spinning to first detect the beacon. Once a beacon signal is detected, the state machine will transition to the APPROACH_TARGET state, where the motors will stop and begin to drive forward so that the bot will directly approach the target while avoiding tape. When a tape event is triggered while the robot is approaching the target, the state machine transitions to a state called SMART_SEARCH, where a sub state machine is activated and used to bot so that the bot stays within the tape. Once a bump event is detected while in approach beacon, we know we have hit the target and so, the state machine transitions into the state FIND_SIDE, where another sub state machine is called to control the robot's traversal around the target. While the bot is maneuvering around the tower, if the trackwire side is detected, the robot will park itself parallel to the wall and then the state machine top level will transition to FIND_&_SHOOT. Within this state, the robot will drive slowly back and forth along the wall until the tape designating the correct hole is detected.

Once it is detected, our servo actuated shooter will be powered and a single ball would be shot through the hole. After the ball is shot, a timer will be initialized and on it's timeout event, the bot

will move away from the tower that it shot at and spin again in order to detect the second target. The full state machine is then repeated until it detects and shoot at a third beacon, where it then transitions to the final state STOP_BOT which just turns on it's motors or does a little victory dance. We found that although this hierarchical state machine had good concepts, much of it was changed during the actual bot implementation, including this final state where we instead coded our bot so that it will continually shoot and target and find another beacon. This was implemented to take care of the situation in which our bot happened to go to the same beacon twice and we want to have more chances of making at least one ball in each of the three beacons.

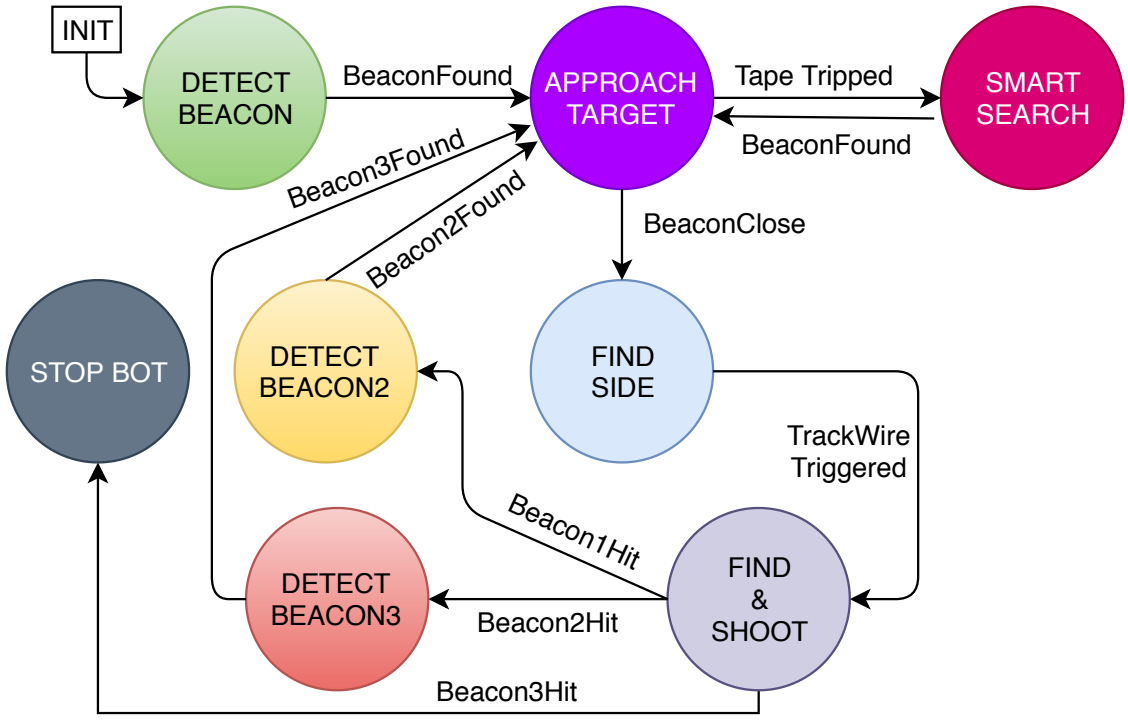


Figure 6: *State Machine V1: HSM Top Level* [3]

2.3.2 SMART_SEARCH Sub State Machine

This sub state machine, shown in Figure 7, was activated when the robot is driving forwards and the floor tape sensors are triggered. Depending on which tape sensors are tripped, the robot moves away from the tape event so that it stays within the field and then the robot spins back to its original direction so that it finds the same beacon instead of a new target. Once the appropriate tape event is handled and we are back on white MDF, then we transition to the state APPROACH_TOP_LEVEL where an event is fed back into the top level HSM so that it transitions back to APPROACH_TARGET and the bot moves forward again.

Initially, we were set on using four tape sensors to ensure that more than half of our bot remains within the tape boundaries: three in the front and one on the back. When the back tape sensor is hit, the bot moves forward and when the center tape of the bot is hit, the turn is randomized, where based on this random turn, the bot spins itself back to detect the original beacon. For our final bot design, the amount of tape sensors used was drastically reduced to two tape sensors which simplified our tape navigation code so that we only had to account for tripping either the front left or front right tape sensors.

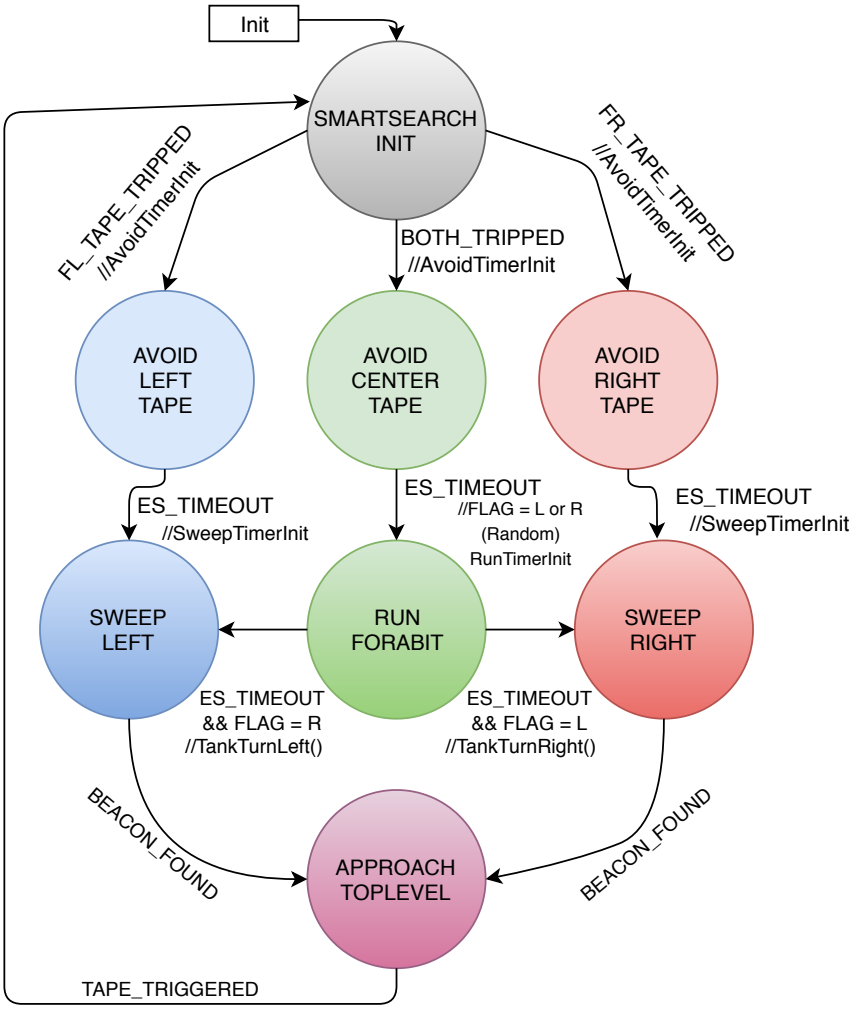


Figure 7: *Sub State Machine #1: Smart Search* [3]

2.3.3 Find Side Sub State Machine

The Find Side state machine, sketched in Figure 8, was implemented to control the complex traversal of the tower based on ping sensor events and timers. The general idea is that the bot hits a beacon and the bot then tank turns to the right until the side mounted ping sensors both detect an object indicated that the robot is parallel to the wall. Once parallel, the trackwire is checked. For our initial design, we were going with a two trackwire design, where they would be aimed at the far ends of the bot, so that when both are triggered, we are sure that this is the right face of the tower. When the trackwire is detected, the orientation of the bot is checked, where our bot repeatedly transitions to three states which moves out bot in a slight s pattern to pivot our way close to the tower and become as parallel as we can. As soon as the bumper placed on our cannon is hit, we would trigger an event to the top level state machine, where it would transition to FIND_&_SHOOT and then this sub state machine would be turned off and initialized on re-entry. The final bot design did not include a front bumper, so instead focused on a turning method that would make us hit the cannon bumper, so we would only have to worry about getting ourselves parallel to the wall.

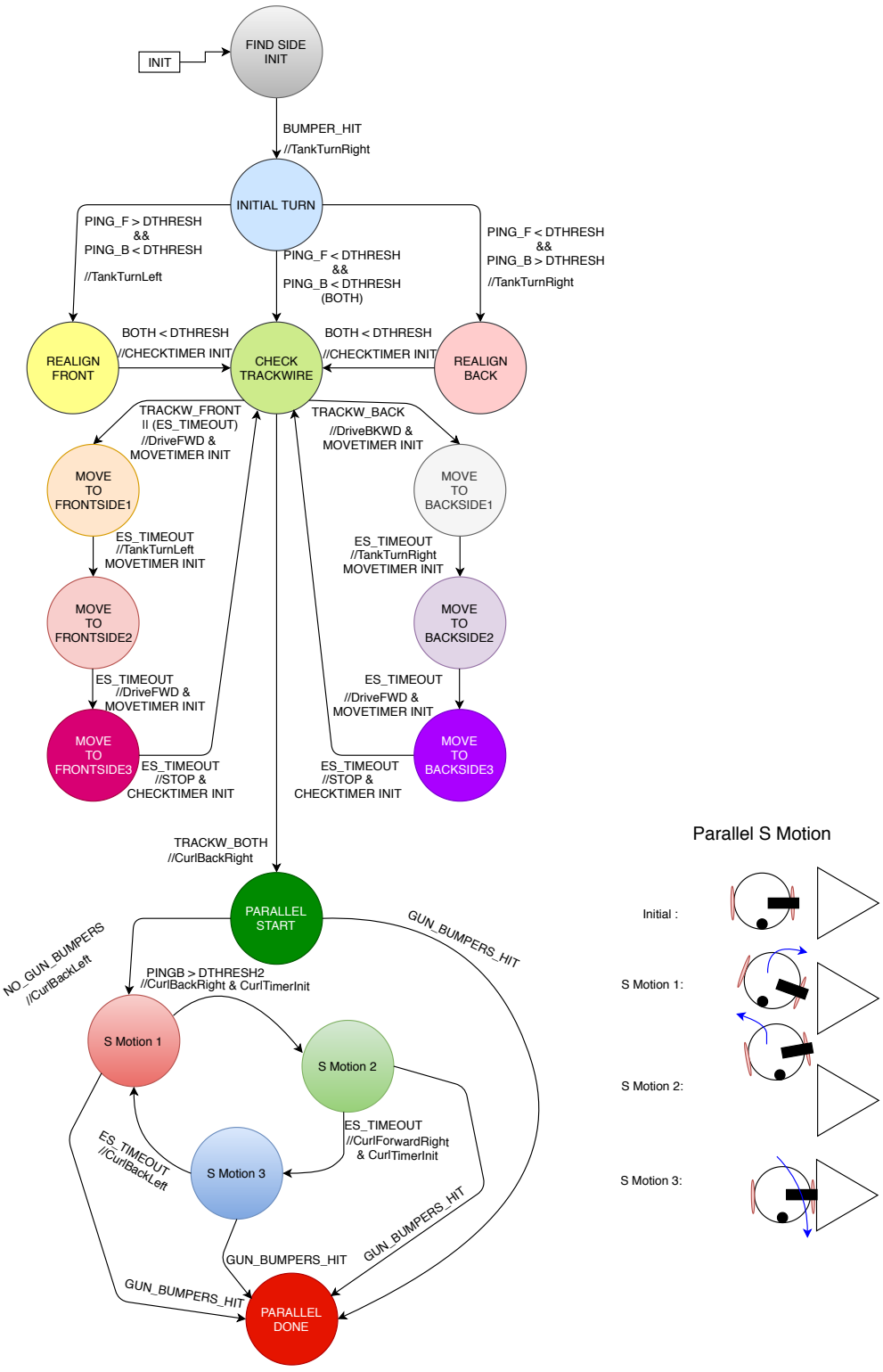


Figure 8: *Sub State Machine #2: Find Side* [3]

2.3.4 Shoot Sub State Machine

The last sub state machine, shown in Figure 9, focused on targeting the correct hole and actuating the shooting mechanism to deposit the ball in the hole. This state machine is activated once we exit the FIND_SIDE state, meaning that we are parallel and close to the correct side of the target. Therefore, we would only have to worry about driving forward and backwards along the side of the wall to sweep and detect the black tape. To determine when we reach the edge of the wall, we are using ping sensor events. When we observe an object on only one ping sensor, where we were originally parallel and both were detecting an object with a similar distance, then this indicates that the ping sensor is now reading empty space and we are about to drive past the tower. Therefore, based off the Ping Far events we have set up, we are transitioning back and forth between SWEEP_BACKWARD and SWEEP_FORWARD to ensure that we stay along the wall and are solely checking for the tape underneath the right hole. Once the tape is detected by both sensors, we will transition to the SHOOT state, which will turn on our stepper motor for a given amount of time. After a timeout event occurs, the stepper motor controlling our shooting mechanism is shut off and the top level state machine transitions into DETECT_BEACON2 which will detect a new beacon.

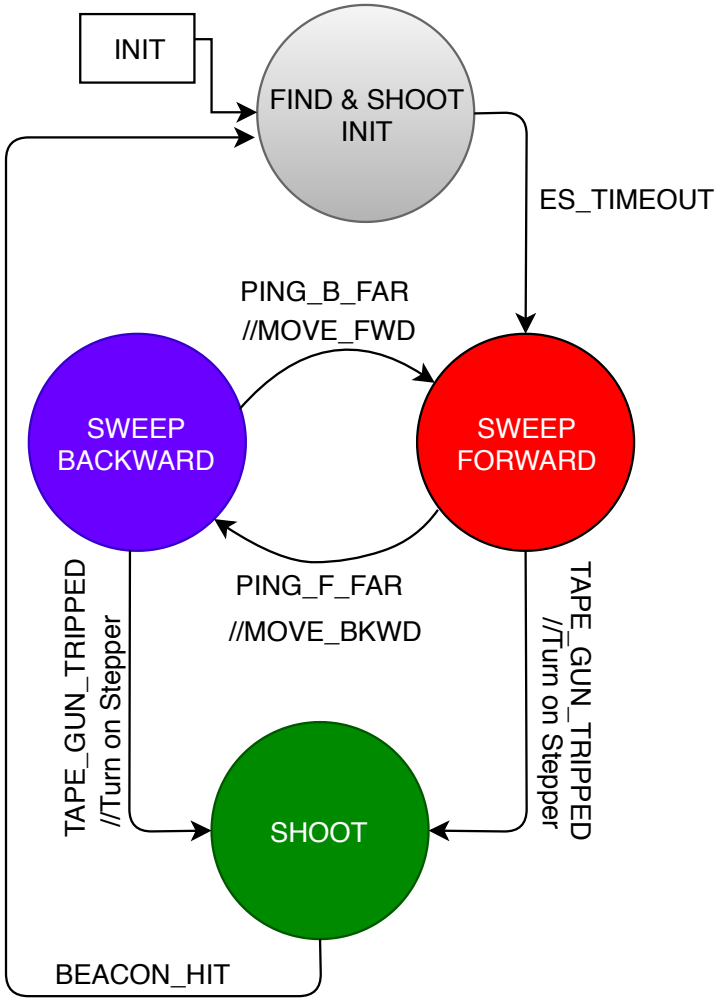


Figure 9: *Sub State Machine #3: Find & Shoot* [3]

3 Electrical Components

3.1 Beacon Detector

For our beacon detector design, we reused a tested and working design from Lab 2, which can be seen in Figure 10. To initially detect the 2 kHz IR beacon signal, a photo transistor with a 1 kΩ sourcing resistor tied to ground was used. In order to amplify the desired 2 kHz signal and filter out noise, a four-stage deliyannis bandpass filter with a total combined gain of 2874.96 was used.

In order to provide the most accurate attenuation range and amplify solely 2 kHz signals, the same deliyannis bandpass design, adapted from a Texas Instruments filter design tutorial [4], was used with the same 2 kHz corner frequency and two different Q factors: 10 and 6.6. Splitting the gain within four stages rather than having high Q factors provides the best chance of having our filter gain peak as close to 2 kHz as possible. The filtered signal is referenced using a 1.65 V split rail buffer so that the full signal is observed and a smaller signal 1.7 V amplitude signal experiences clipping.

The next stage of the design consists of a peak detector with a 10 ms time constant so that its output charges faster and approaches as close to the peak of the bandpass filter as possible. The last stage, responsible for converting the circuit's analog signal into a digital signal, consists of a comparator with hysteresis, whose high threshold is set at 2.1 V and the low threshold is set at 2.0 V. With this thin boundary, and an offset of 1.65 V from the split-rail buffer, a filtered signal with an amplitude larger than 350 mV is required in order to turn on our beacon detector. After testing the soldered circuit with a beacon powered at 5 V, we observed this beacon detector circuit to function from a range of 1/2' to approximately 11 1/2'.

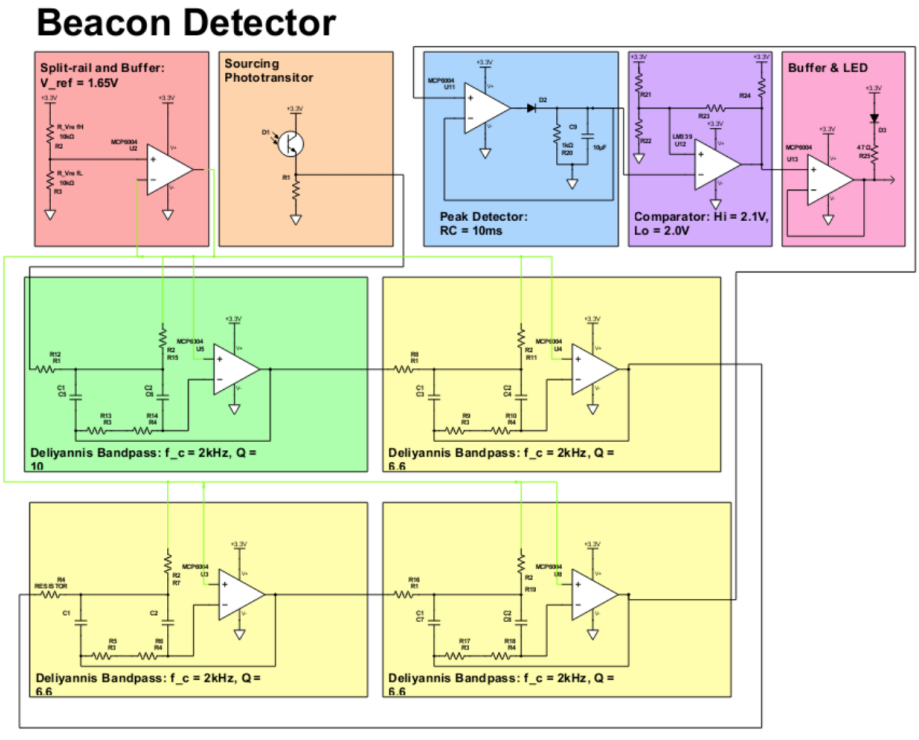


Figure 10: *Beacon Detector Schematic.*

3.1.1 Difficulties and adjustments

When mounting this beacon detector on our bot, we found that it was difficult to place it where we wanted because the board itself was bulky and the photo transistor was soldered directly onto the board. Therefore, we desoldered the phototransistor and soldered on male header pins instead. We then soldered female header connectors to the ends of the phototransistor so that it was easy to connect to the board and easy to mount on the top of the bot. Also, there was no ports in which we can take the output of the circuit and relay it to digital or analog ports. To fix this issue, we bought screw terminals and soldered them to the board. On one port, the analog signal from the peak detector was connected, and on the other port, the comparator digital output was connected. We found that making both accessible was useful for testing purposes and just in case we want to switch between digital and analog beacon detection.

Luckily, we did not encounter much issues with the actual beacon detector circuit. The slight adjustments we made to the protoboard were more for implementation with the robot to make it easier to mount and connect to the UNO.

3.2 Trackwire Sensor

The second crucial electrical circuit we had to prepare for the robot was a trackwire sensor, since this was necessary in order to determine the correct side of the three-sided tower. We previously have designed a track wire detector for Lab 1 of this course, so we merely adapted that circuit. We found that for correct wall detection, we do not want too much range for our trackwire sensor, since this could cause us to detect the trackwire through the other walls.

For our design, shown in Figure 11, we used a tank circuit with a resonance frequency of 23.2 kHz so that the 25 kHz trackwire is still able to induce some voltage out of this stage. Next, we sent that signal though a a noninverting amplifier stage with a gain of 11 in order to amplify the size of the signal, so that we can observe noise that needs to be filtered out. After doing this, we saw that there was still substantial 30 mV of 60 Hz noise in our signal from our power supply, so we chose to use a first order high-pass filtering stage with a corner frequency of about 17 kHz. This provided about -49.04 dB of attenuation for the 60 Hz noise and the 25 kHz trackwire signal was now easily visible.

The output of this high-pass was then fed into two more stages of non-inverting amplification each with a gain of 11. The amplified signal is fed into a peak-detector with a time constant of 100 msec to provide a steady signal into the comparator. Note that our op-amp stages are all referenced to ground, since we were more concerned with the frequency of the signal. Doing this for our op-amp stages allowed for a greater voltage range when working with the comparator threshold bounds, considering that now we can utilize signals containing amplitudes of 3 V rather than only 1.5 V. We decided to use a single-threshold comparator with a low threshold of 0.4 V considering that this provided us with an appropriate range of detection, and also we did not need to worry too much about debouncing of the signal, since we only want to check for the trackwire signal briefly. With a total gain of 1331, this trackwire circuit was soldered onto a PCB board and it was tested to detect the right side of the wall at a distance of about 4 inches away. and it does not detect the track wire when directly touching the incorrect sides. For our PCB board design, we added a 3.3 V regulator to make it easy to implement with the power distribution on the bot. Also, we added a terminal to connect the analog output of the peak detector and the digital output of the comparator so that they can be accessible to the UNO for checking of the trackwire events. This was very useful, as we had to switch from digital to analog processing after the comparator threshold on our circuit board was found to be too low.

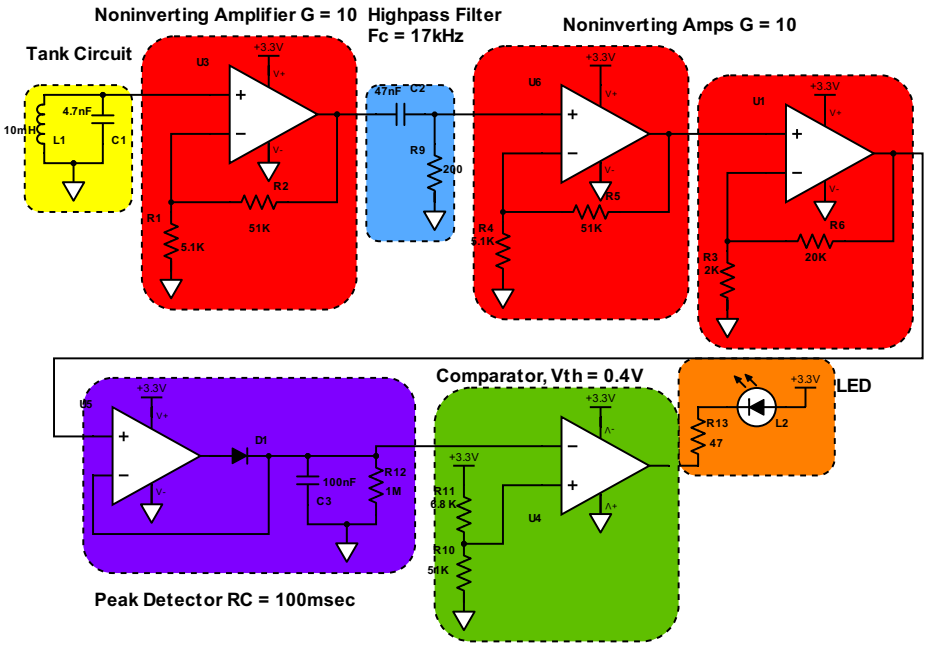


Figure 11: *Trackwire Sensor Schematic. [2]*

3.2.1 Difficulties and adjustments

We had originally planned on using two trackwire circuits, placed on each end of the bot so that by checking both we would have robust trackwire detection. We did solder up two functioning trackwire circuits, however we found that although both were functioning and based off the same design, they contained very different ranges of operation. One was tested to be working at about 2 inches from the wall, and the other had a range of about 5 inches.

We decided to test how both could be implemented with the bot, and what we found was that the trackwire with the longer range would correctly detect the right face of the tower, but if the bot was close enough, or if the inductor on our circuit was aimed at the corners of the tower, the trackwire would be detected even if it is on the wrong side. Our soldered circuit with a closer working range, however, was able to correctly detect the side lined with trackwire and does not detect the corners of the wrong sides of the tower. The only challenge was this, was that we would have to mount the inductor at the edge of our bot and we would have to code our maneuvering so that we would be parallel and right next to the wall when we are checking for trackwire.

This method worked during initial test runs where we would program the bot to approach the tower, hit it in the center and tank turn to align with the wall. If it was not a side with trackwire, the bot would continue moving forward, and turn the corner, however if it were, the bot was programmed to stop. For these tests, the trackwire circuit was taped onto a layer of the bot. Since we wanted to be able to adjust the position of the trackwire, we unsoldered the inductor from the board, attached header pins to it, and extended the inductor leads with female headers. After some successful tests with the stationary bot, we found that when the UNO was powered, for some reason the header cables would detect strange 5 kHz noise which would be large enough

to trigger our trackwire. This was a major issue, but we could not find the cause of the problem and we did not have the time nor energy to figure it out. So for our solution, we checked that the other trackwire sensor was working and not experiencing this strange noise with the motors and other components attached and powered with a 9.9 V battery. We directly mounted the trackwire protoboard on the bot with the soldered-on inductor facing forward and checked the analog readings we got when next to each side of the tower. By updating the event code so that it works with this analog trackwire signal detection, made it in the end more robust and efficient since we didn't have to worry about further soldering issues, and if our range was wrong, we could merely change it in code rather than on our circuit.

3.3 Bump Sensors

In order for our bot to not get stuck and be able to resolve collisions, we had to use bump sensors which would be triggered when our bot hits an object. Since we would only be looking to program our bot to move forwards with only slight back up states, we chose to only use one front bumper with a long slint in the center which is attached to the base a screw to allow space for the bumper to be able to smoothly move forward. On the left and right side of the bumper, two bump switch sensors , manufactured by [], were attached so that it could hold the bumpers steady and when bumped, the switch would provide enough spring force to push the bumper back to its resting position. We had some issues when using our bump sensors with the UNO digital input pins, so after much testing we found that the configuration shown in Figure 12, provided the most reliable indication for when the bump sensors were pressed. The only drawback is that when bumped the digital signal is low at about 80 mV and when unpressed, the signal is high at about 1.8 V. This is fine, however we just had to careful and account for flipping this digital signal within software so that the events are properly triggered.

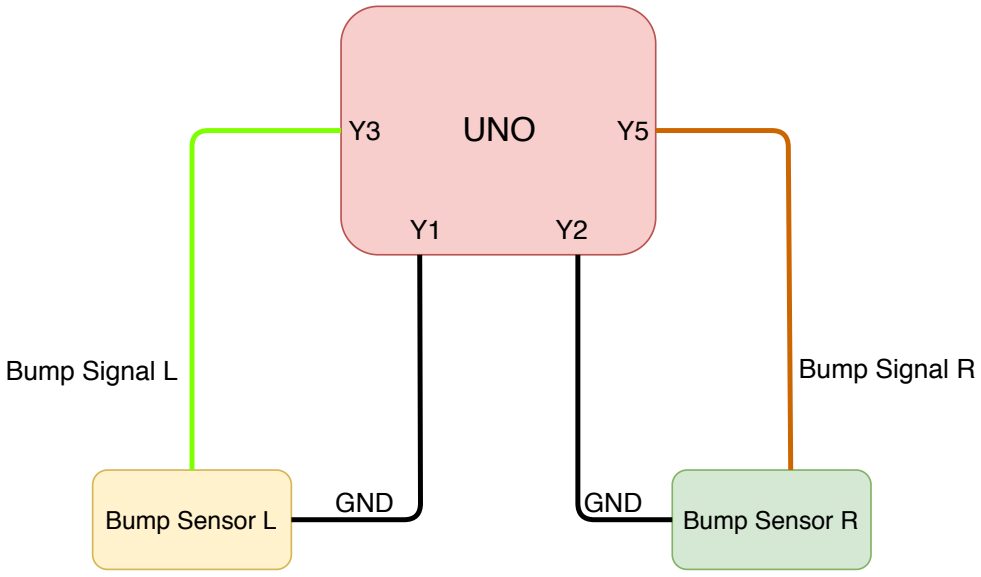


Figure 12: *Bump Sensor Schematic. [3]*

3.3.1 Difficulties and adjustments

This bump sensor configuration worked cleanly and did not cause us any major issues. The only issues we found were when intitially testing them with the UNO, as we had problems detecting

the event in software. When testing the bump sensors with simply a power supply, we found that by connecting the NC pin to ground and the NO pin to a 3.3 V rail, and scoping at the center COM port, pushing down on the bump sensor would output a clean high signal of 3.3 V and when unpressed it would output a clean low signal of at most 40 mV. However, when testing with the UNO pins, I found that if I set one of the pins to high, where it was probed at about 3.28 V, and connected the bumper identically to the previous set up, the bump sensor signal was scoped to output a clean 3.28 V signal when pressed, and when unpressed, the signal would still be clean, however it was observed at around 1.8 V instead of the expected 40 mV. This was an issue considering that our bumper would never be able to trigger events, considering that for the digital UNO pins, any voltage greater than about 200 mV would be detected as a high and thus even if the bumper is unpressed it would never be detected as low in software.

To fix this issue, we found that this 1.8 V signal was only observed when the COM connection to the UNO port was attached. Since we needed to be able to read the bump signal, we instead tried disconnecting the NO connection to 3.3 V and instead only had a connection to ground from the UNO ports and the digital connection to the UNO. During testing, we observed that when unpressed, the signal would be clean at about 1.8 V. And when pressed, the switch would close and the UNO input port would be directly shorted with UNO ground and would thus read a clean low signal of at most 80 mV. Since this option seemed to be the most reliable, we worked with the bump sensor configuration shown in Figure 12, and merely flipped the signal output in software to properly trigger events.

3.4 Tape Sensors

For our tape detection, we used the TCRT5000 sensor module, manufactured by OSOYOO [1]. The schematic for this component, outlining the internal circuitry and comparator design, is shown in the top right corner of Figure 13. With the tape sensors tested to ensure full functionality, we powered them directly with the 3.3 V regulated power supply from our power distribution board. The output of these tape sensors are then connected to the analog input ports of the UNO so that we can adjust the threshold bounds easily through software rather than by messing with the potentiometer on the sensor module. The set up of this circuitry again can be seen in Figure 13. We found that each tape sensor drew a maximum of about 20 mA each and we were only using three on the bottom of our bot and two on the side for our ball shooter, not much power was being consumed if we had them on during the entirety of the course.

When testing the analog readings from the tape sensors, we found that they were pretty accurate, but they would be most efficient at a specific distance. At an ideal distance of about 5 cm, white MDF would produce low analog readings below 100, and if on black tape, the tape sensors would produce analog readings of about 800 and above. If they are too close however, the white lettering of the tape would produce a low reading and would disrupt the tape detection as it would act as if it caught white space even if the tape sensor is aimed at the tape. If the sensor were too far from the white canvas, the distance would cause the white MDF to appear as black tape, where the analog reading would go from 600 and when the distance was large enough, was undistinguishable from black tape with a reading of about 850 and up.

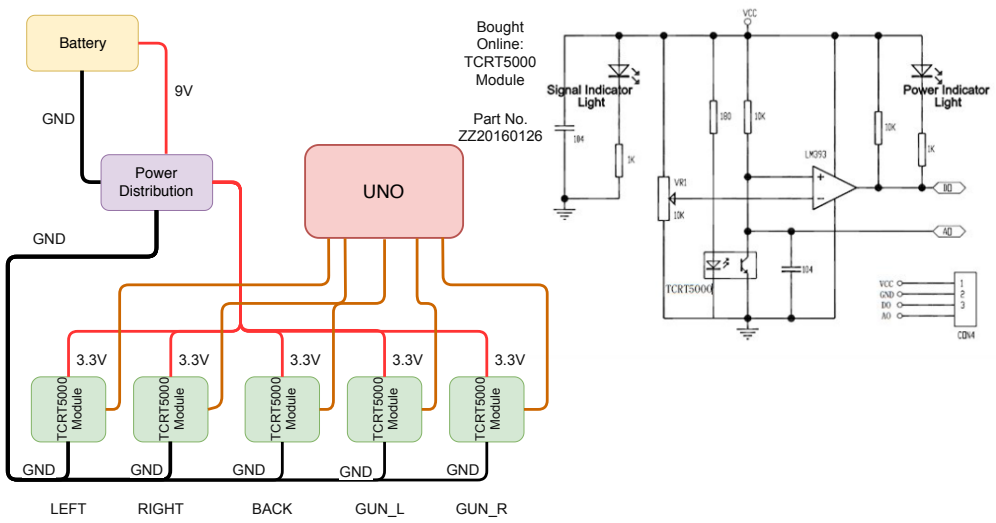


Figure 13: *Tape Sensor Schematic.* [3]

3.4.1 Difficulties and adjustments

Some difficulties we encountered, were more dealing with the tape sensors on the shooting mechanism since the floor tape sensors were underneath the bot and were shaded so we did not experience detection issues with them. However, since the tape sensors were facing forward and were dependant on our distance away from the tower, we ran into several issues. One of those issues was that there would be a point where our threshold bounds were too low and we would be too far to the wall, so when we would be parallel, have detected the correct wall and are driving along that wall to find the tape, our analog reading of the black tape would be too low and our event would never get triggered. One solution we implemented to fix this issue was to merely adjust our threshold bounds and bring our shooter tape sensors closer to the side of the bot to cut down on the distance away from them. This process took many iterations of calibrating, considering that different light levels would mess with our threshold bounds, and there would be times where the sensors would be too far away from each other so that they would both get triggered individually but not at the same time so we would never get the event where we are centered with the hole. By making it easy to adjust the mounts for the tape sensors, we found that fixing this issue was tedious but easy.

3.5 Ping Sensors

A set of ping sensor modules, manufactured by , was used by our bot to to easily detect when we are close to objects as well as exactly how far away we are from them. We found that for the most efficient use of them, they were most reliable for getting our bot parallel with the wall as well as detecting when we have driven off the the side of the wall. Due to these advantages, we chose to use their functionality for our tower traversal so that we can quickly and accurate position ourselves parallel to the wall and turn sharply around the corner once we have driven past the edge.

To set up the ping sensors in terms of circuity, we merely had to power them with a 5 V rail and provide a high signal of at least 3 V which would act as our trigger for the UNO. The output of this ping sensor would also be set at 5 V, so this would be an issue once we have to read it digitally with an UNO port since each port has a limit of about 3.4 V. Therefore, we had to make a dedicated board to step down the output echo signal from the ping sensor so that it would produce a clean 3.3 V signal that is suitable for the UNO. There is also a 5 V regulator attached to the board, so to

power the ping sensors we merely used the UNO power distribution connected to a 9.9 V battery. As for the trigger pin, we used digital ports V3 and V5 and set them high when we wanted to poll the ping sensors. By using a voltage divider, we were able to step down the 5 V echo signal and detect a 3.3 V signal with the UNO pins V4 and V6. The set up of our ping sensors is shown in Figure 14

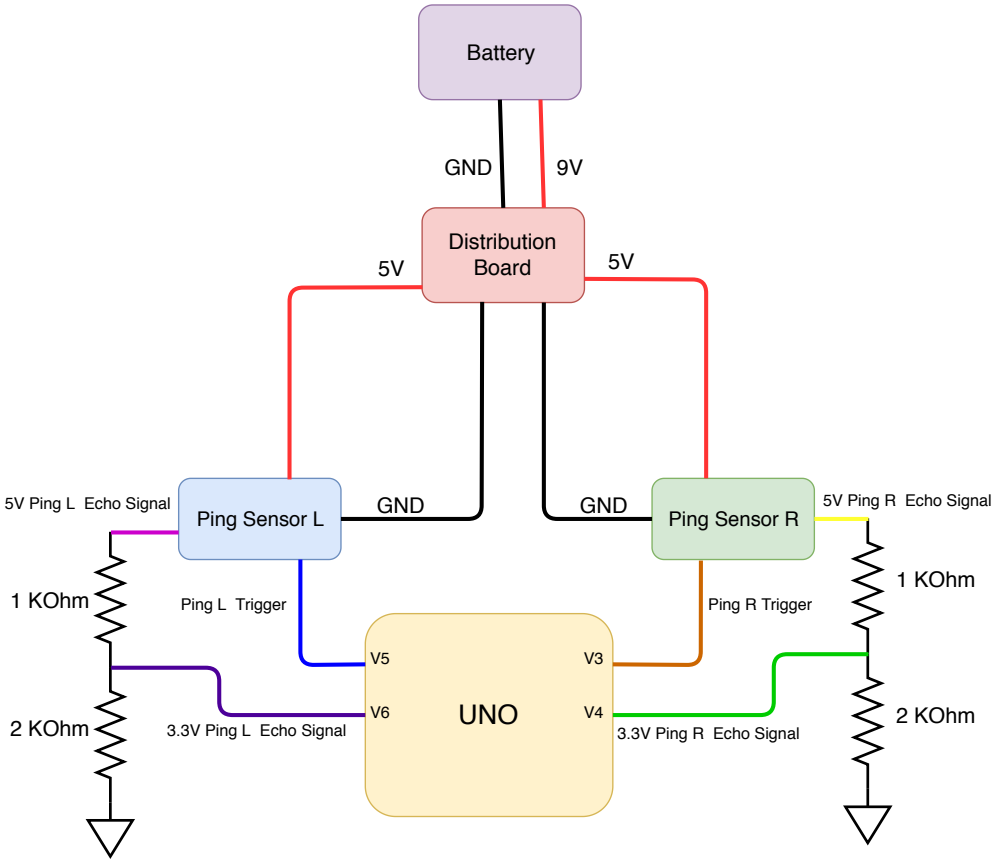


Figure 14: *Ping Sensor Schematic. [3]*

3.5.1 Difficulties and adjustments

When working with the ping sensor, we did not encounter any electrical issues where they would not be powered or would not be producing the correct signals. We only ran into cases where we forgot to plug them in or we forgot to use library initialization code to turn on the appropriate pins. We did however run into many obstacles with software implementation, since the ES framework runs off of a standard ms timer system and this timer would not provide the timing resolution required for proper distance detection. By using one of the unused timers and setting up a microsecond timer system specifically for use with the ping sensors, we were able to get proper readings down to about 3 cm. Although they were difficult to work with, the ping sensors did serve well in detecting when we are parallel to the wall as well as when we are no longer detecting a wall.

3.6 DC motor and Stepper Actuation

Since the sensor modules were more complex than the actuators on our design, we chose to describe them in more detail within seperate sections. As for our actuators, we only had to work with the DC motors as well as the stepper motors for all of our bot's physical functionality. For the DC motors, we controlled them using the H bridge module connected to a 9.9 V power and ground from the power distribution board as well as a PWM signal output from one of the UNO pins. The first motor was connected so that the positive lead is connected to A+ and the negative lead is connected to A- on the H bridge module. The second motor was connected in the same way, however, the leads were switched and connected to the B terminals so that when facing away from each other, the motors would rotate in the same direction, given the same PWM signal.

After many iterations, our final shooting mechanism was physically powered by a DC stepper motor, that we attached to spokes. The idea we had was that these spokes would hold and separate ping pong balls on a carousel type design which would have a hole at one of the slots. By stepping the stepper motor an 8th of a full rotation, the ping pong balls would be moved and one of them would fall into the slot with a cut out. To drive this stepper motor, we used the provided stepper driver module powered with the 9.9 V supply from the power distribution board. We connected the direction pin to an UNO ground pin, connected the enable to an UNO pin, and the steps to an UNO PWM output pin. To actually control this actuator, we momentarily set its enable pin to high, and input a PWM signal into the step input of the driver module. Both of these configurations are shown in Figure 15

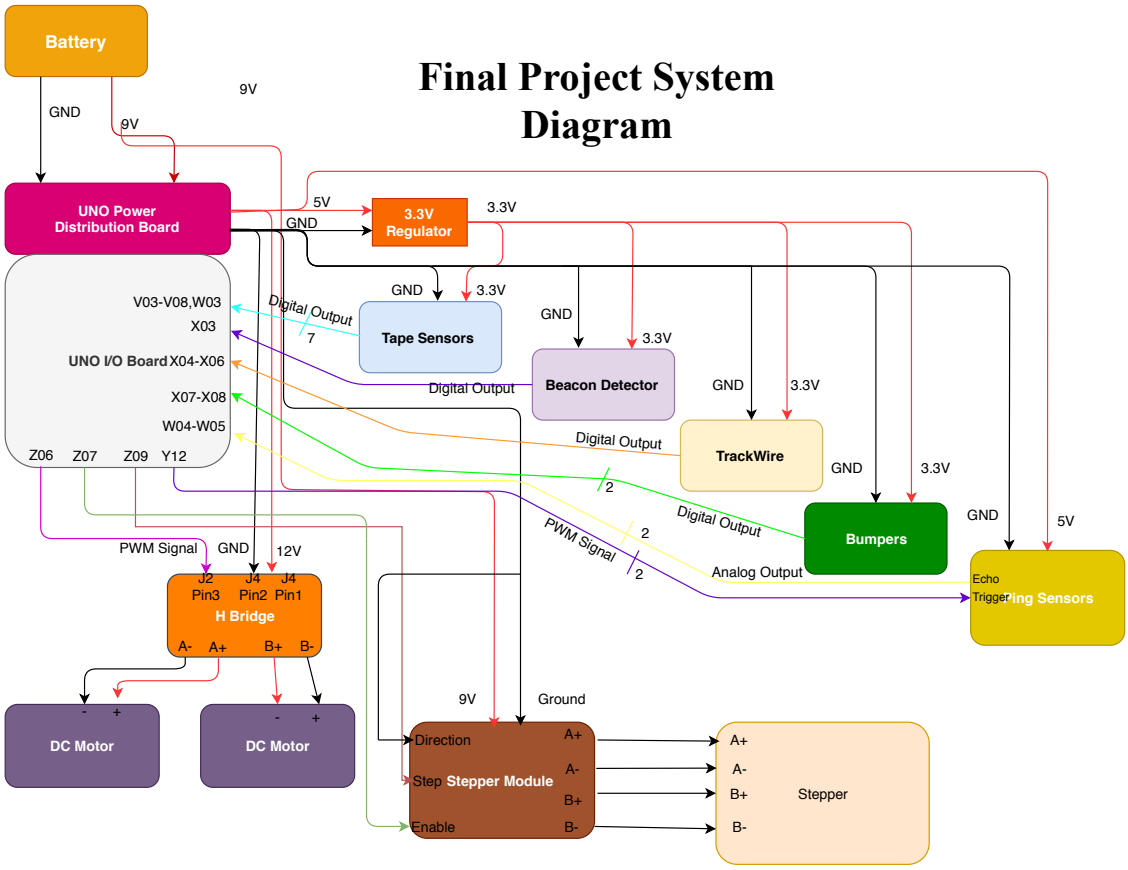


Figure 15: *Full Bot Schematic.* [3]

3.7 Complete Bot Integration

With all of the electrical components individually tested and set up, the next challenge was to connect them up on the bot and ensure complete integration of all sensor modules and actuators. The full block diagram, outlining all of our component connections with associated pins, is shown in Figure 15. When first connecting the bot, we saw that a big problem when dealing with all of these components was both wire management as well as power distribution. To power all of the necessary electronics, and make the cut down on the wiring, we soldered a dedicated power distribution board which would take in 9.9 V from the battery and use two regulators to provide a 3.3 V rail and a 5 V rail. Five sets of terminals on the distribution board provided 3.3 V to the three floor tape sensors as well as the two gun tape sensors. All together, they drew about 18 mA each totaling to less than 100 mA. The beacon detector and track wire sensors were both powered with 5 V power rails since the circuit boards had 3.3 V regulators soldered on them. The other actuators were powered directly from the 9.9 V UNO power distribution board.

The bump sensors as well as pins used to drive circuit modules were all connected to digital pins on the UNO. We chose to save the analog input pins on the UNO for our tape sensors, trackwire circuit, ping sensors, and beacon detector. In order to extend the UNO port connections, and make them look a bit cleaner, we used ribbon cable with male and female header pins attached. This was very useful for cleaning up the wiring on our bot. We tried our best to map out the connections on the bot so that the ports are closest to where the components should be placed. We made our bot with three removable layers attached with screws and bolts. The bottom most layer contained the motors and bump sensors, since we felt that these components would rarely have to be removed or detached. The UNO would have to connect to all of the sensors and actuators including the floor tape sensors at the bottom of the bot and the stepper motor driver located at the top of bot. For this reason we chose to mount it on the second level with spare foamcore and tape and used header cables to attach required connections to their respective UNO ports.

We have been advised from multiple TA's that the current being passed through the motors as they rotate the wheels, have been known to induce unwanted noise within people's circuits in the past. Specifically, with people's trackwire circuits. To test this theory, I powered up my trackwire sensor, and aimed it at the internals of my bot. When close enough, the motors definitely tripped my trackwire circuit as it experienced strange noise depending on the motion of the motors. Also, we observed that when the motors are off and when the trackwire is directly pointed at the UNO, the trackwire sensors also gets tripped by some induced noise. To avoid this issue, we decided to simply mount the trackwire circuit at the third level of our bot, where the power distribution board and the stepper driver module would be placed. The beacon detector was not affected when placed next to the motors or the UNO I/O board, which makes sense because the signal is based off of the light detected from the phototransistor. However, we still had to be able to detect the beacon across the field and the female header pin leads of the phototransistor are too short, so we mounted the beacon detector at the third level next to the trackwire so that the actual phototransistor could be attached to the front of our bot as high as possible and its view would not be blocked.

Throughout the course of testing and state machine implementation, we consistently kept good electrical engineering etiquette. Due to this, we did not burn/short/or blow up a single component that we used for our designs. Also, we were always are of our battery voltage and did not let it get too discharged and "poofy".

3.7.1 Difficulties and adjustments

The problems that we did have with our bot when integrating all of the electrical components did not to do with actual wiring of the bot, UNO ports, or induced noise between individual components, but instead we struggled with wires getting unplugged unexpectedly. In order to fix the issue of unplugged wires, we had a specific day where we took note of our connections, and cleaned out our bot entirely. Then we soldered male header pins onto wires to make specific and solid connectors for UNO port and other header connections. After we did this and taped down connections to the base layers, we did not encounter much more instances of loose connections messing up our robot's functionality.

Although our actual bot connections were solid, we noticed that if our bot did any sudden movements, turns, or changes of speed, there would be times where our bot would get turned off for a slight moment and then turn back on. We narrowed down the source of the issue to a broken battery connector which had a bad connection on one of the wires. To get around this, we cut the wire and resoldered the internal metal fibers in the wire and reattached heat shrink to ensure no shorting would occur. With this resoldered battery connector, the bot wouldn't turn off every other time our bot quickly increases its speed. Besides these loose connections messing with our bot, we found that all of our electrical components worked pretty well when all implemented on the final bot design.

4 Mechanical Design

4.1 Overview

For our implemented design, we decided it was best to implement a carousel shooting mechanism which feeds a ball to a barrel using gravity and a gear with paddles. The barrel is mounted on the side of the bot allowing us to easily move parallel to the wall and shoot. Similarly to our first preliminary design, our base is circular to allow us circumnavigate around the tower without obstruction. We decided to place the wheels within the base to prevent the wheels from being caught as well.

Our design implements multiple platforms to house the motors, boards, stepper, and more. These platforms are interconnected using threaded rods and are held in place using lock and hex nuts. Having an open frame allows us to easily have access to all our boards and electrical components. Furthermore, it allows our overall bot's height to be adjustable because we use hex nuts to hold the platforms together.

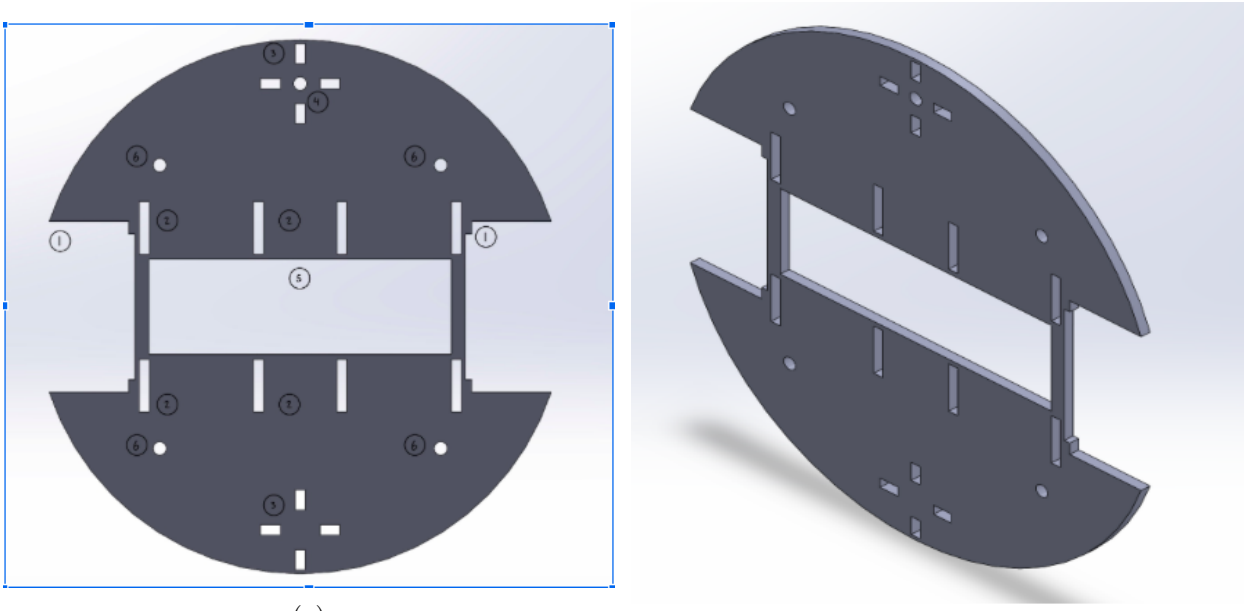
Our floor tape sensors will be placed on the front skid and our side tape sensors will be positioned below the barrel. The single trackwire is placed directly next to the barrel and our beacon detector is placed outside our carousel shooting mechanism. Furthermore, we decided to implement ping sensors on the side of our bot to help navigate our around the tower.

4.2 Base/First Platform

We decided to implement a circular base design, allowing us to tank turn near the tower without obstruction. The base of the bot has multiple cutouts that will be used for mounting of the skids, motor mounts, and rods.

The first cut allows for placement of the wheels inside the circle. Initially, these cutouts did not allow the wheels to be fully cleared. Thus, we extended the cut to allow for the wheels to be mounted without rubbing against the platform. The second cut is meant to hold the motor mounts in place. The distance between these slits is based on the motor's length. The third cut

allows for the placement of our skids, which keep our bot level. The fourth cut is meant to mount the rod that will be used to maintain the bumper in place. The fifth cut is meant for the motor's wire management, allowing us to mount the motors below the platform. The sixth cut allows for placement of the rods, allowing the base of the platform to be attached to the second platform.



(a) (b)

Figure 16: *Base Top View and Isometric View.*

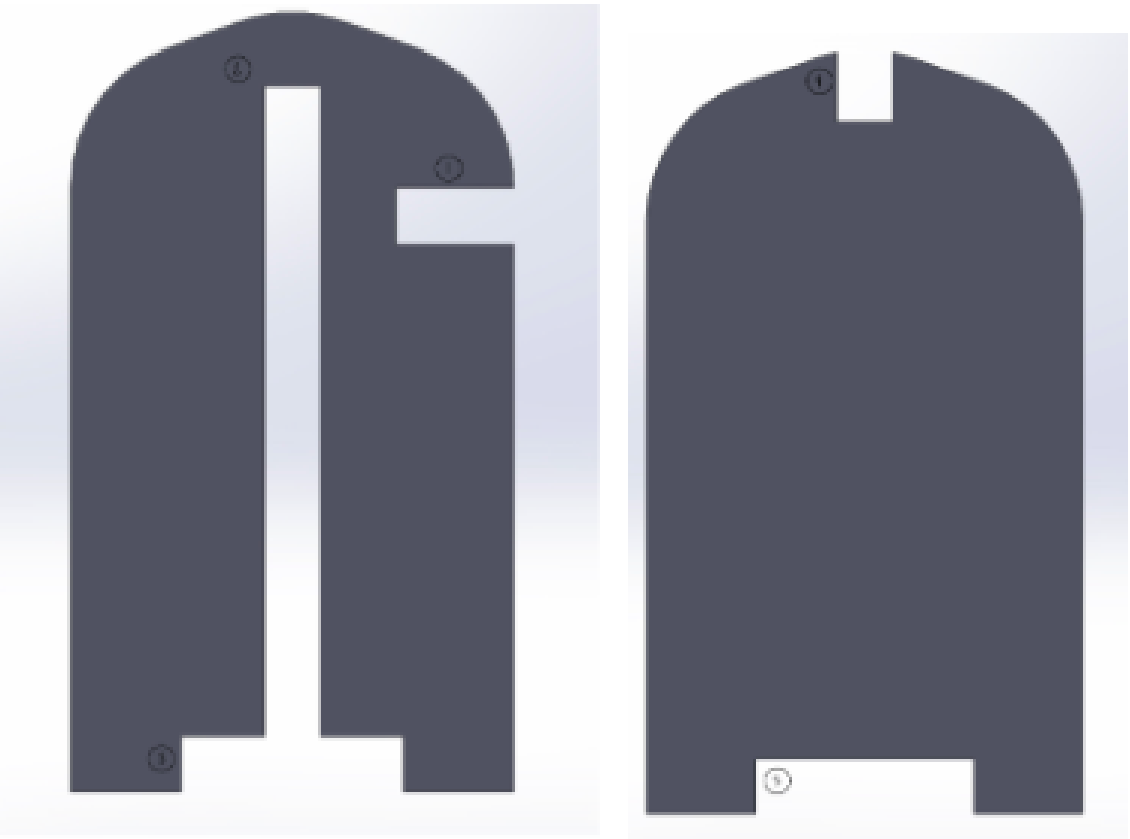
4.2.1 Difficulties and adjustments

One of the difficulties we encountered was placing the wheels inside the base. In our first design, we did not provide sufficient clearance for the wheel. As a result the wheels would rub against the platform causing the motors to stall. In order to correct this, we increased the area of the wheel cutouts. Furthermore, we also mistakenly set the MDF thickness parameter to 0.21in in our first design. The motor mount and skid cutouts were larger than the MDF thickness, resulting in loose fitting components. Adjusting the MDF thickness parameter to 0.18in in our second design resulted in proper fitting components.

4.3 Skids

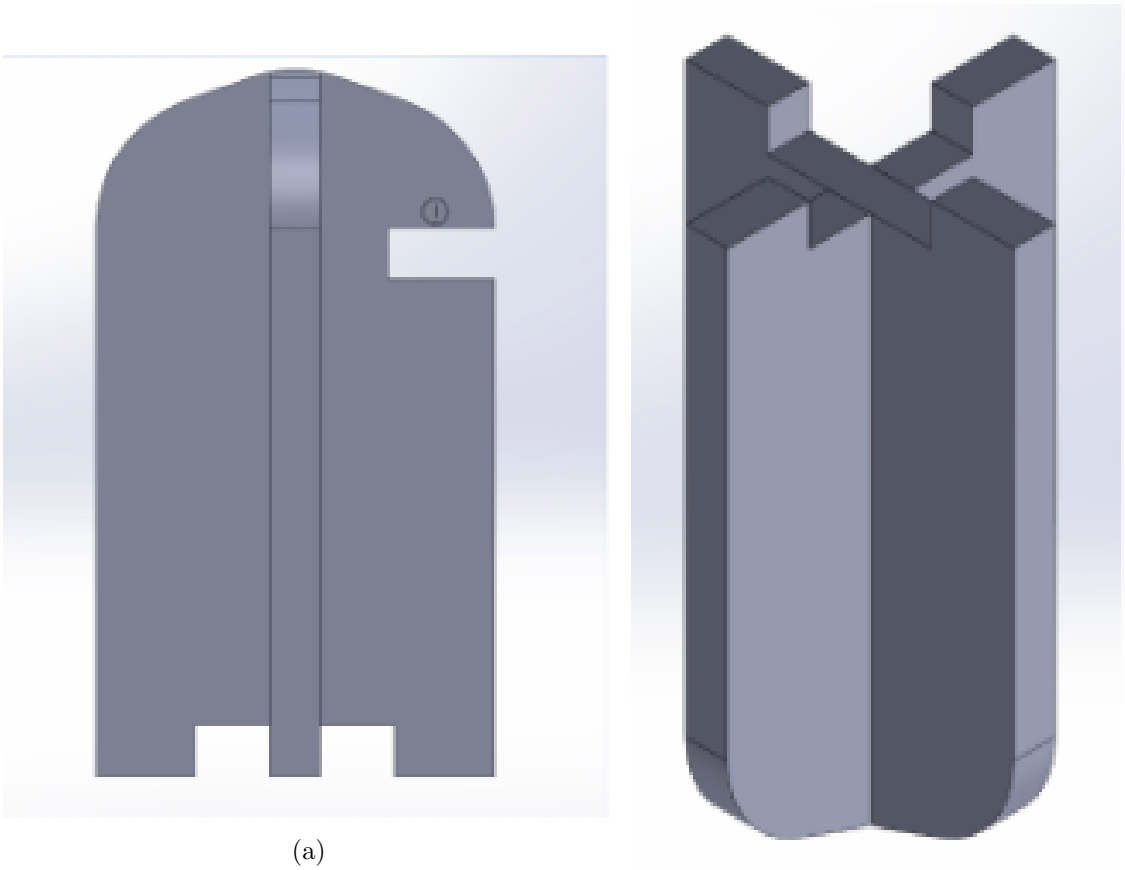
Skids were implemented in both the front and rear of the bot, these are necessary since our design only implements two wheels. The skid design we implemented was inspired by the roach's skids which is comprised of two components that interconnect.

The first skid component has three major cuts, shown in Figure 17a. The first cut will be used to place the mount for the tape sensor at the front of the bot. Attaching the tape sensors as close to the floor as possible is crucial in order to obtain accurate readings. The second cut and fourth cut, shown in Figure 17a and Figure 17b, will allow our skid components to be assembled together to create a single skid. The full assembled skid is shown in Figure 18a and Figure 18b. The third and fifth cuts will allow our full assembled skid to be attached to the base of the bot using a tab and slot method.



(a) (b)

Figure 17: *Skid Parts Side View.*



(a) (b)

Figure 18: *Assembled Skid Side and Isometric View.*

4.3.1 Difficulties and adjustments

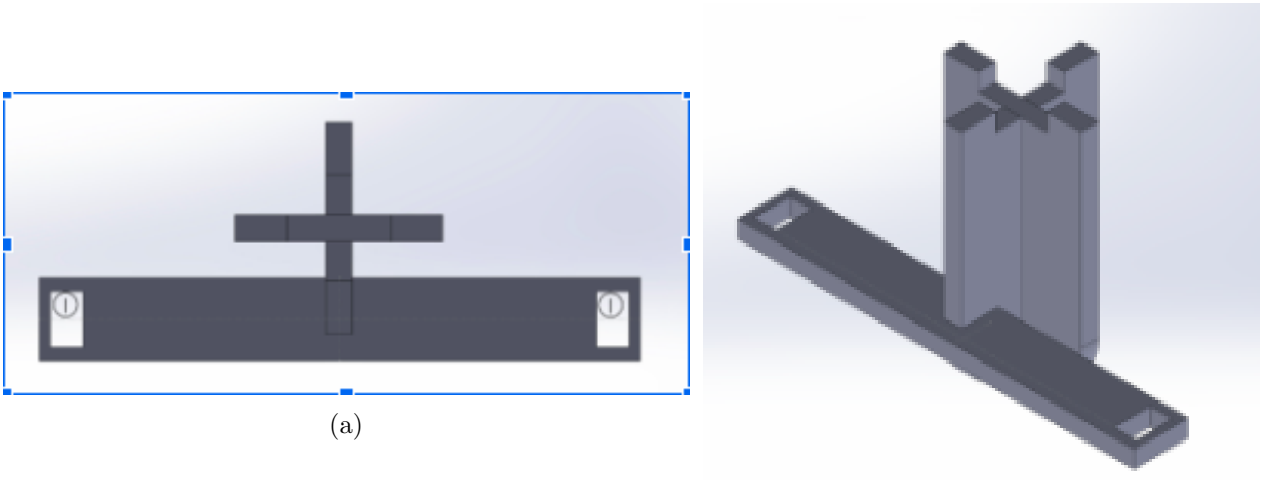
One of the major difficulties we encountered with our skids was determining the height to implement in our design. The initial height we implemented, 2 in, was not large enough and as a result our bot would tilt forwards by approximately 20 deg. Furthermore, when the bot began decelerating the base would tilt backwards and forwards by 20 deg. In order to correct this issue, our second design implemented larger skids to reduce the bots instability. These skids were designed to be flush to the floor, aiding the bots balance. However, once we began testing our tape sensors we observed that the skids got stuck once they were moving over black tape. As a result, our bot would become immobile. Instead of laser printing new skids with a lower height, we decided to sand off the skids. Although sanding both the front and rear skids slightly reduced balance, the skids no longer got stuck when moving over black tape.

4.4 Floor Tape Sensors

Our design implements two front tape sensors, attached to the front skids, that will be used to navigate the bot within the black tape. We decided that there was no need to implement rear tape sensors since there were very few cases where our bot moved backwards. Note that the tape sensor mounts are attached to the skids using hot glue.

The mounted tape sensor's first cut, shown in Figure 19a, allows for placement of the tape

sensor's IR LED and photodiode. The tape sensors are placed on the mount and are attached with tape.



(a)

(b)

Figure 19: *Tape Sensor Mounts Bottom and Isometric view.*

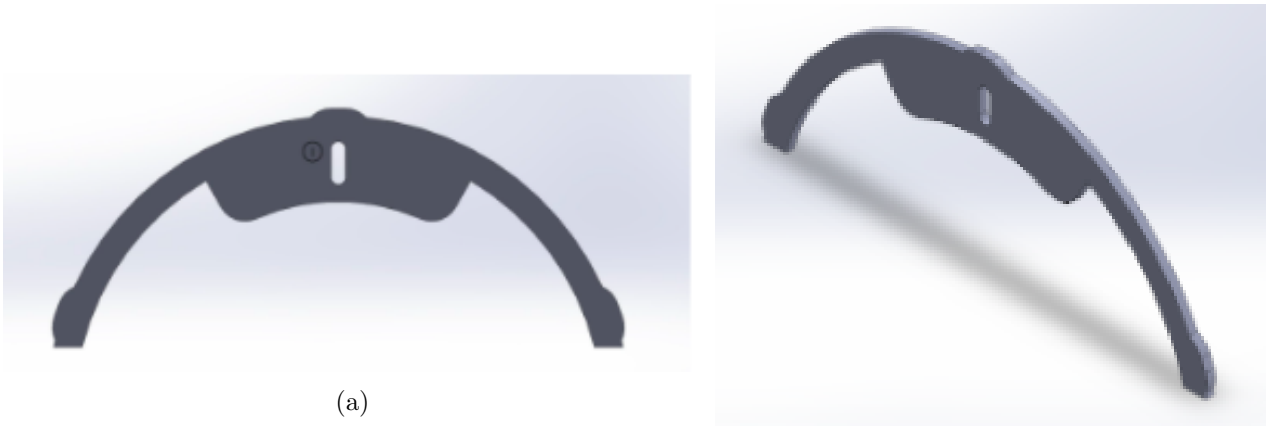
4.4.1 Difficulties and adjustments

One difficulties we encountered was managing the wires from the tape sensors. Initially in our first design, we did not provide enough clearance between the skid and tape sensors. As a result, we were unable to attach the wired tape sensors. Increasing the length allowed for the wired tape sensors to be attached without obstruction.

4.5 Bumpers

Our design implements a single front bumper, comprised of two momentary push buttons and is inspired by the roach's bumper design. The roach's design used a center screw to allow the bumper to move forwards and backwards between the platforms. Furthermore, it used momentary push buttons to keep the bumper from wiggling left and right. Unlike the roach however, our bumper is circular to prevent any obstruction between the bot and the towers.

The first cut, shown in Figure 20a, will allow the bumper to move forwards and backwards via a rod that is connected between the base and second platform. Along with the rod the momentary push buttons prevent the bumper from unecessarly jerking left and right. Our bumper design also uses three bumps on the outer radius of the bumper to clearly distinguish each bump event.



(a)

(b)

Figure 20: *Bumper Top and Isometric view.*

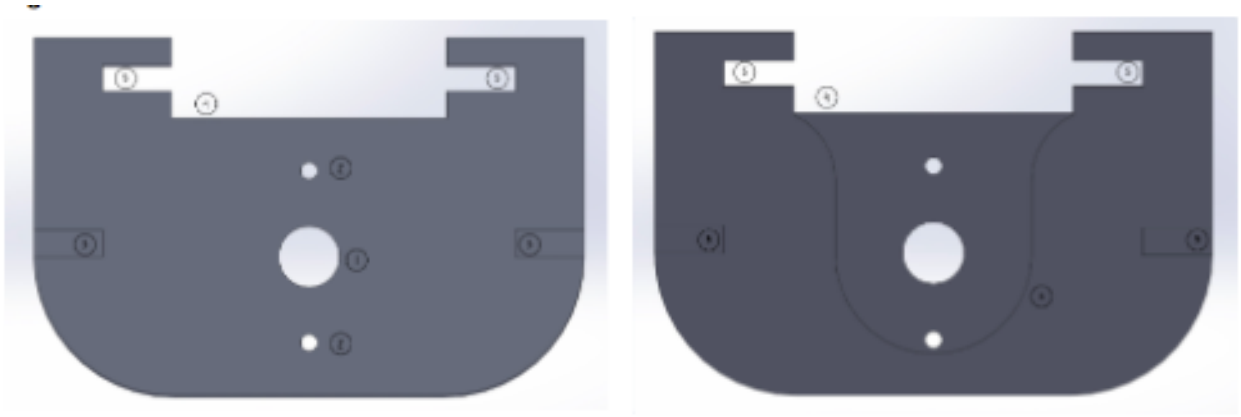
4.5.1 Difficulties and adjustments

One of the difficulties we encountered was deciding how to place the momentary push buttons. At first we thought it would be to our benefit to model the momentary push buttons on solidworks, allowing us to have an approximate idea of where mount the sensors. However, we found that our bumper worked best when the hole cutouts were made using a drill. The reason being, we were able to obtain the most responsive bumps when the momentary push buttons are pushed slightly against the bumper.

4.6 Motor Mounts

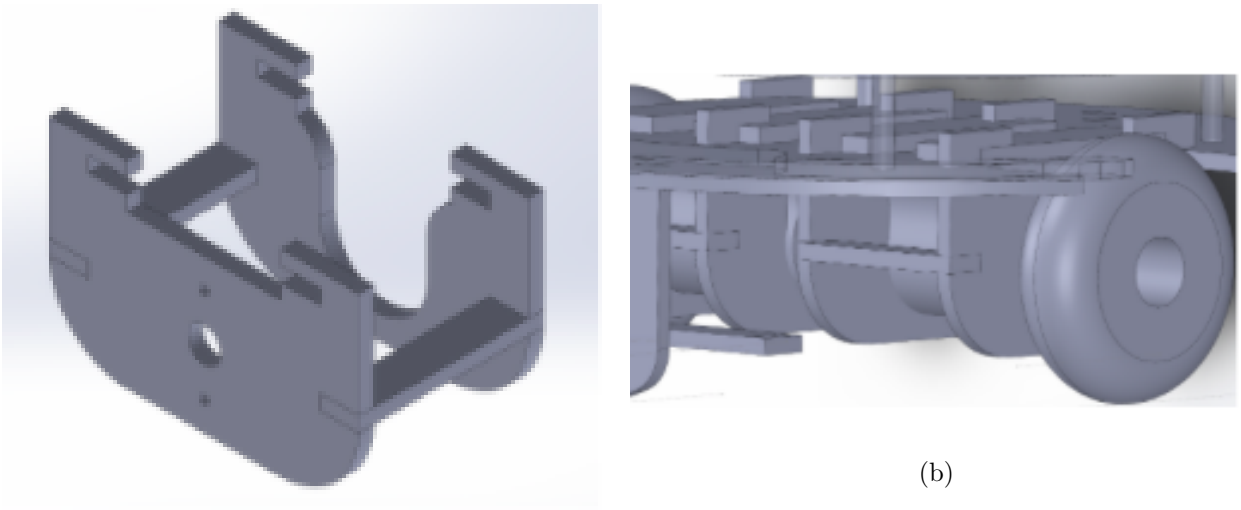
We decided to implement a motor mount design that could easily be removed from the bottom of the base, using a tab and slot method. In particular, we wanted to avoid unscrewing the motors from the mounts when we disaasmbled our bot.

The first cut, shown in Figure 21a, provides sufficient clearance for the motor's shaft. Initially, we planned on simply mounting the motor without screws. However, we found that screws were needed to keep the motors stationary. For this reason, we implemented the second cut for the motor's screws. The rear motor mount serves to hold the motor, preventing the motor from weighing down the front motor mount. A tab and slot method is used to interconnect the front and rear motor mounts, shown in Figure 22a. The assembled motor mounts are attached to the base platform using tabs that keep the mounts from popping off, shown in Figure 22b. Our motor mounts provide permanent housing for the motor while still being removable from the base of the bot.



(a) (b)

Figure 21: *Front and Rear view of Motor of Motor Mounts.*



(a) (b)

Figure 22: *Individually assembled motor mount and motor mount attached on bot with motors.*

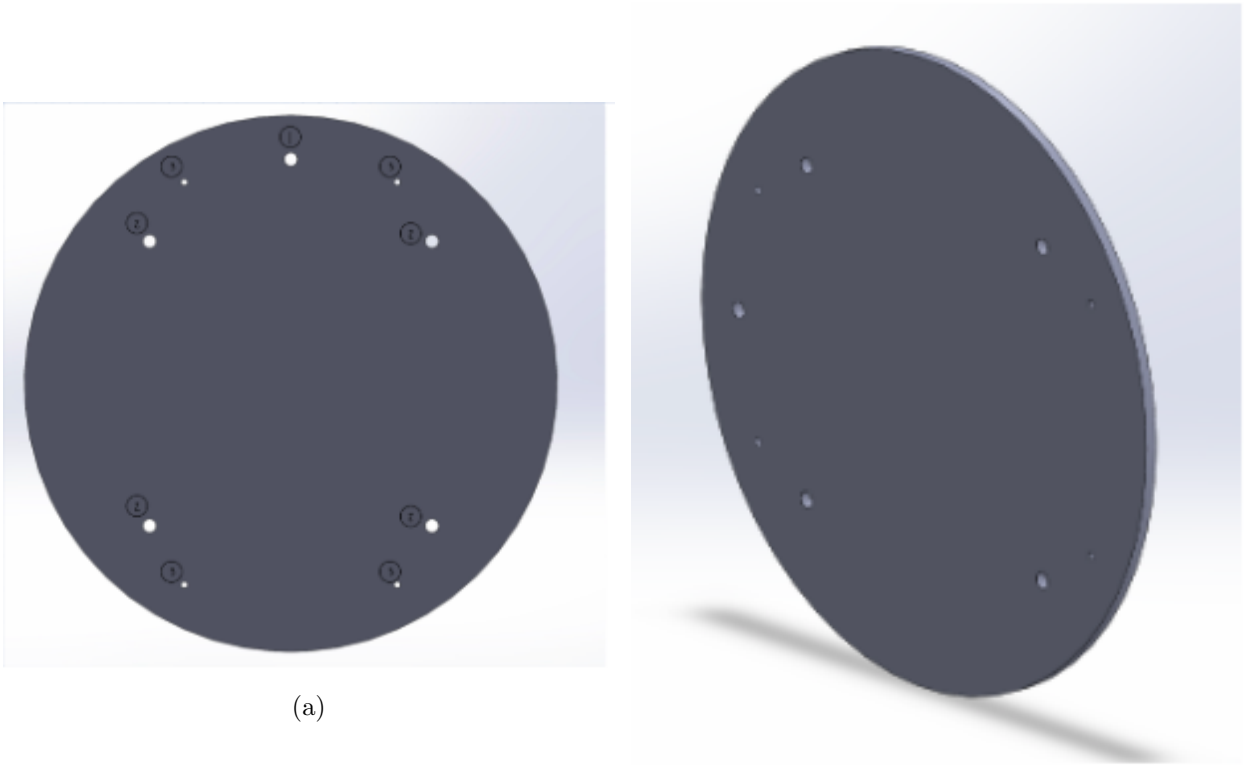
4.6.1 Difficulties and adjustments

Thankfully, we did not encounter many difficulties when designing the motor mounts since we had access to the motor dimensions. This allowed us to design accurate motor mount models that would fit snug on the motor. Furthermore, we were very happily surprised with the structural integrity of our mounts once they were mounted on the bot.

4.7 Second Platform

The second platform houses a variety of different components including: the rods that will interconnect the platforms and the rod that will allow the bumper to move forwards and backwards. Furthermore, it is meant to house the UNO, tape sensors, and H-Bridge. In particular, we mounted the side tape sensors on the platforms using MDF tabs which were taped down.

The first cut, shown in Figure 23a, is used to place the rod that will be used to move the bumper forwards and backwards. The second cut will be used to attach the base platform and the second platform together using rods and hex nuts. The third cut will be used to interconnect the second and third platforms together, again using rods and hex nuts. The diameter of the third cut is smaller than the diameter of the second cut since we decided to use thinner rods in order to keep the weight down.



(a)

(b)

Figure 23: *Top and Isometric view of Second Base Layer.*

4.7.1 Difficulties and adjustments

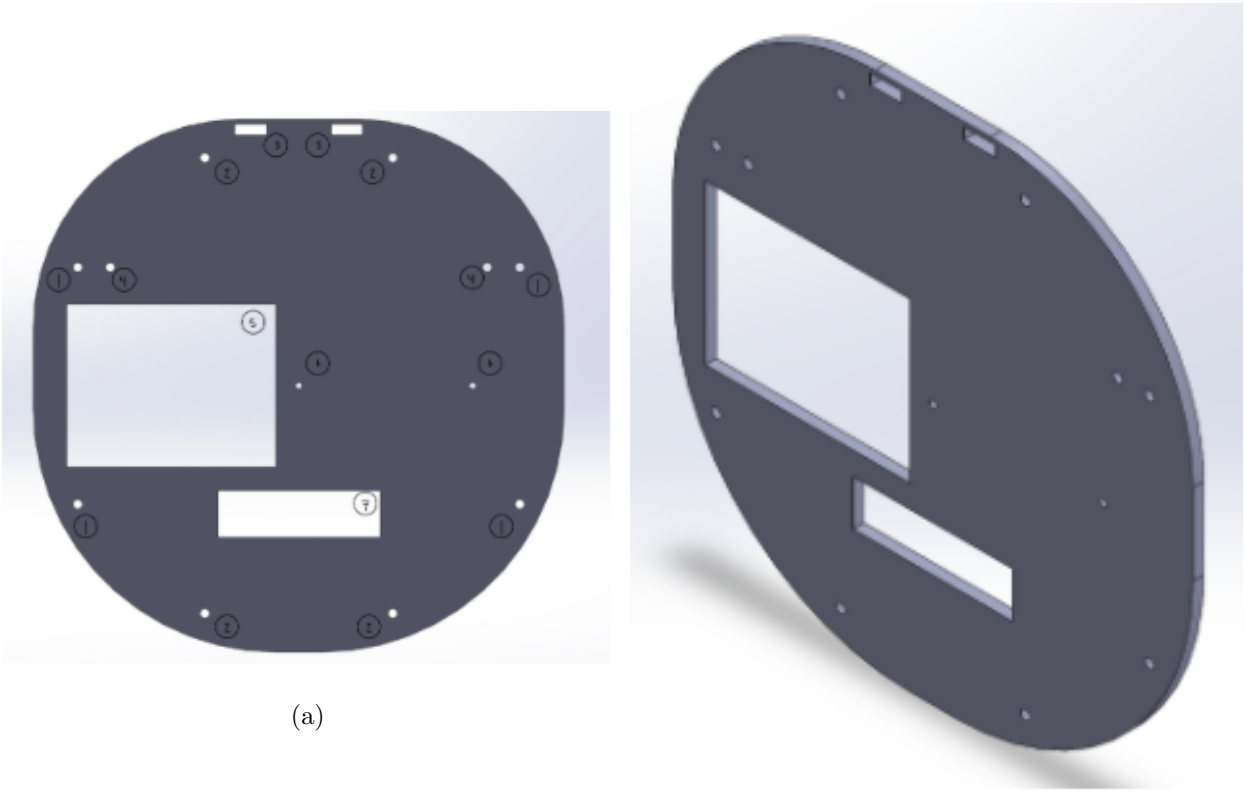
We didn't encounter many difficulties designing the second platform. One adjustment we made with our second design was slightly increasing the diameter of the holes to allow the rods to slide in easily. Otherwise, it would require the hole cutouts to be sanded off to allow the rods to be placed easily.

4.8 Third Platform

The third platform is crucial in our design as it allows for placement of the mount for the barrel, power distribution, and proximity sensors. Unlike the first two platforms, the third platform is non-circular. We decided to implement a square design that used the fillet feature on the corners. The reason being, we wanted to maximize the space available in order to fit the various other components.

The first cut, shown in Figure 24a, is for placement of the rods that will be used to interconnect

the third and second platform. The diameter of these cutouts were large enough to allow the rod to slide in but small enough to allow a hex nut to hold the platform. The second cut allows for placement of the rod that interconnects the third platform and the carousel. The third cut is a tab and slot method that is used to hold the barrel mount. The fourth cut allows for placement of a smaller rod, that will be used in combination with the first cut, to hold the proximity sensors in place. The fifth cut allows our power distribution board to be placed upside down and the sixth cut allows us to screw the distribution board in place. This design choice allows us to be close to the UNO and better manages our wiring. Finally, the seventh cut is used for wiring management since the trackwire, stepper, and stepper board are located above this platform.



(a)

(b)

Figure 24: *Top and Isometric view of Third Base Layer.*

4.8.1 Difficulties and adjustments

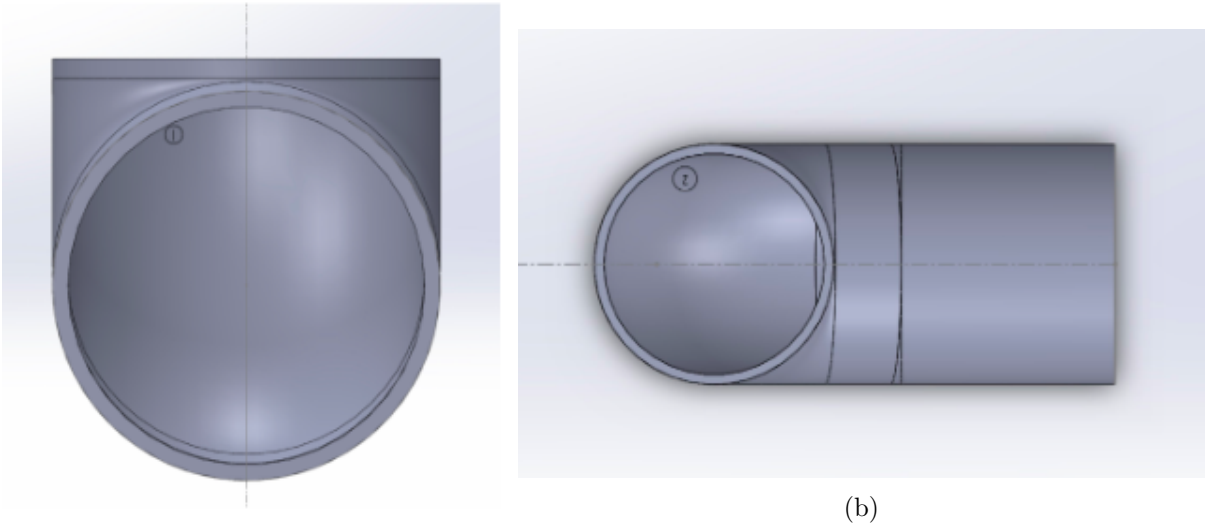
The main difficulty we encountered when implementing our third platform was managing space for the barrel, power distribution board, and stepper. Originally we had planned to place the stepper on the third platform but we decided it would be best if it was mounted directly below the carousel using screws. This provided enough space to place the power distribution board.

4.9 Barrel/Barrel Mount

Our barrel is mounted on the third platform and uses gravity to feed the ball inside the hole. The diameter of the barrel's hole is 50mm which provides 10mm of clearance for the 40 mm ping pongs. Figure 25a and Figure 25b show a front and top view of our barrel design. Our barrel was

thankfully 3D printed by our classmate Todd.

The barrel mounts sole purpose is to hold the barrel upright in order to interconnect with the carousel. The third cut, shown in Figure 27a, is also 50mm. This allows the barrel to fit snug in the mount, preventing the barrel from slipping out. We used a tab and slot method to mount it on the third platform. The ball is fed into the barrel through the carousel it is attached to, shown in Figure 28.



(a) (b)

Figure 25: *Front and top view of slide for shooting mechanism.*

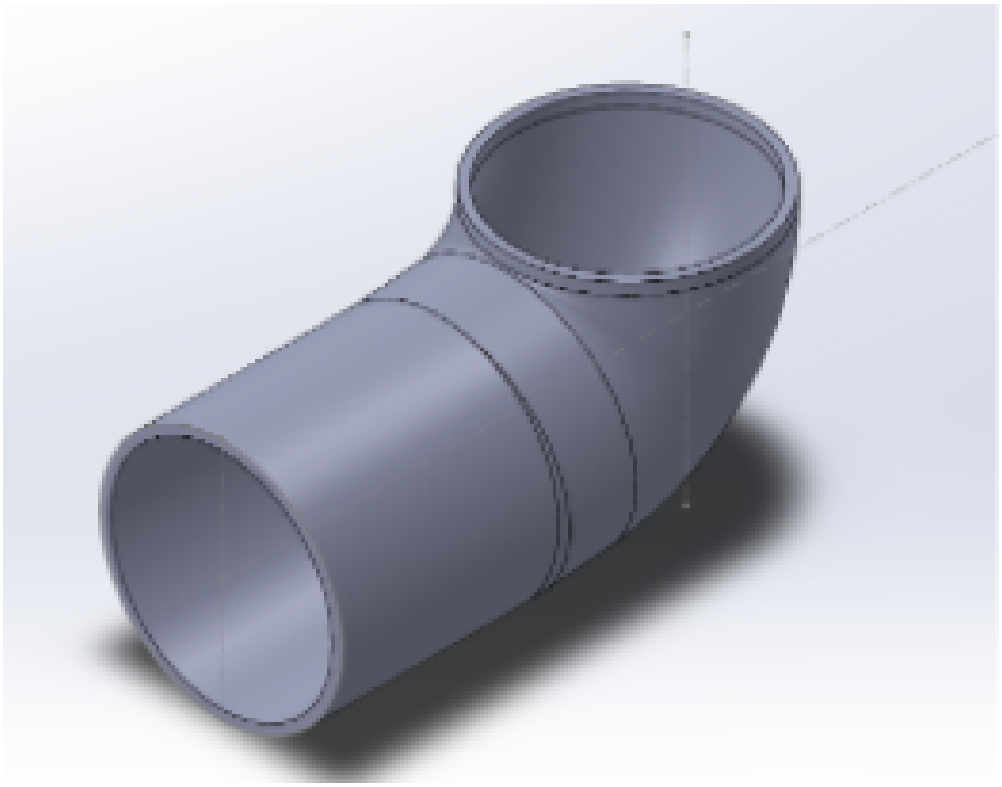
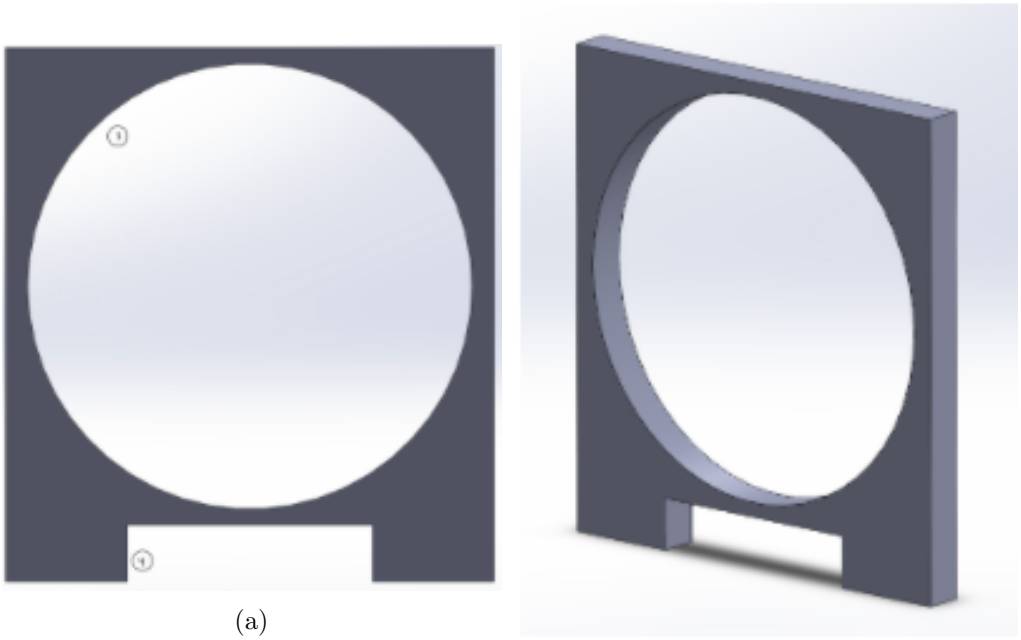


Figure 26: *Isometric view of Slide.*



(a)

(b)

Figure 27: *Top and isometric views of the mount used to attach the slide onto the third platform.*

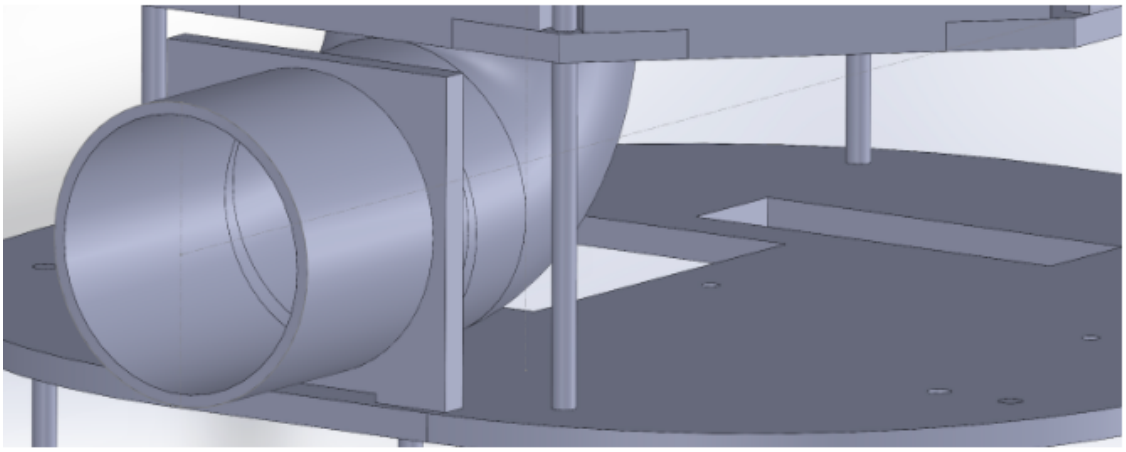


Figure 28: *Isometric view of Slide Connected to Base.*

4.9.1 Difficulties and adjustments

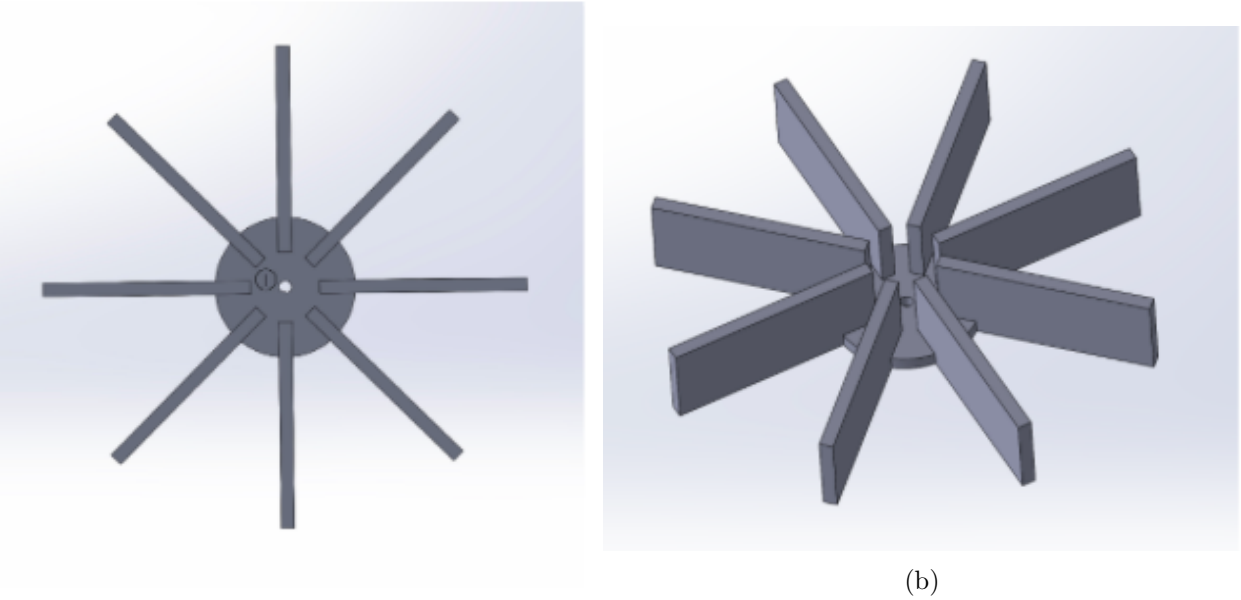
One of the adjustments that we made to the barrel was decreasing the amount it sticks out of the bot. Initially, the barrel stuck out beyond the base of the platform. As a result, the bot would get stuck when maneuvering around the tower. We corrected this issue by shaving down the barrel to be flush with the base of the bot.

4.10 Carousel Mechanism

Our carousel ball mechanism feeds balls into the barrel using a stepper motor which is connected to a gear head with eight tabs. The gear with eight tabs is shown below in Figure 29a and Figure 29b and is directly attached to the shaft of the stepper motor.

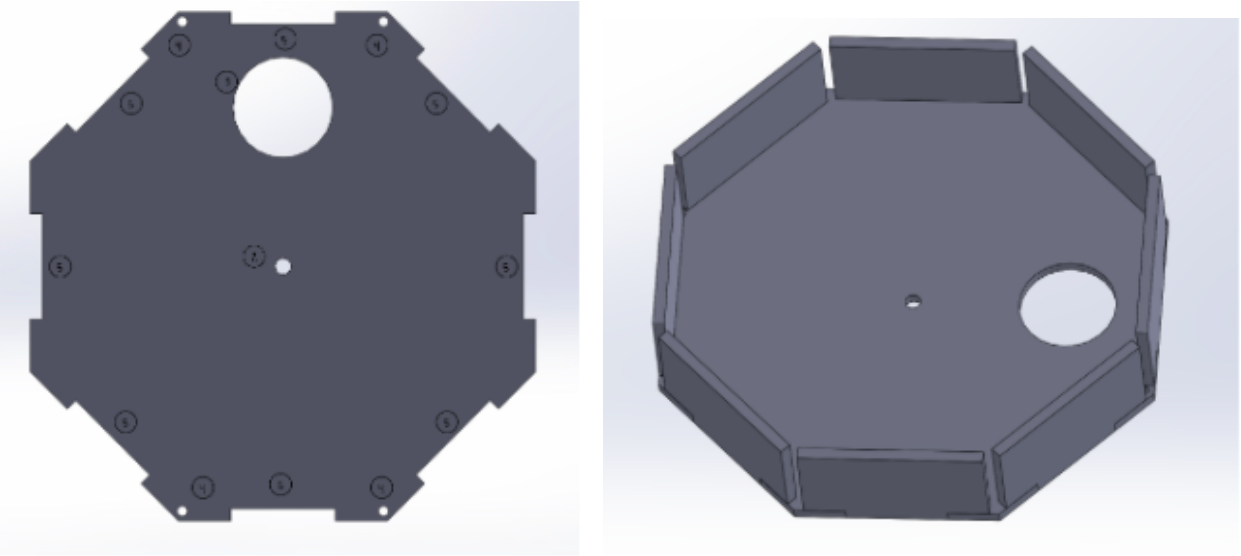
The base of the carousel is comprised of one 40mm diameter cutout which will allow the balls to fall into the barrel. The second cut, shown in Figure 30a and Figure 30b, allows the shaft of the stepper motor to be placed within the carousel. Using the tab and slot method, we mounted tabs around the carousel's platform to keep the balls within the platform.

A complete assembly of our carousel mechanism is shown below in Figure 31. Our dispensing mechanism can hold at maximum of 7 balls. This is due to the fact that initially there must be an empty ball located where the 40mm hole is located.



(a) (b)

Figure 29: *Top and Isometric view of Carousel Roulette Wheel for shooting mechanism.*



(a) (b)

Figure 30: *Top and Isometric view of Carousel Base for shooting mechanism.*

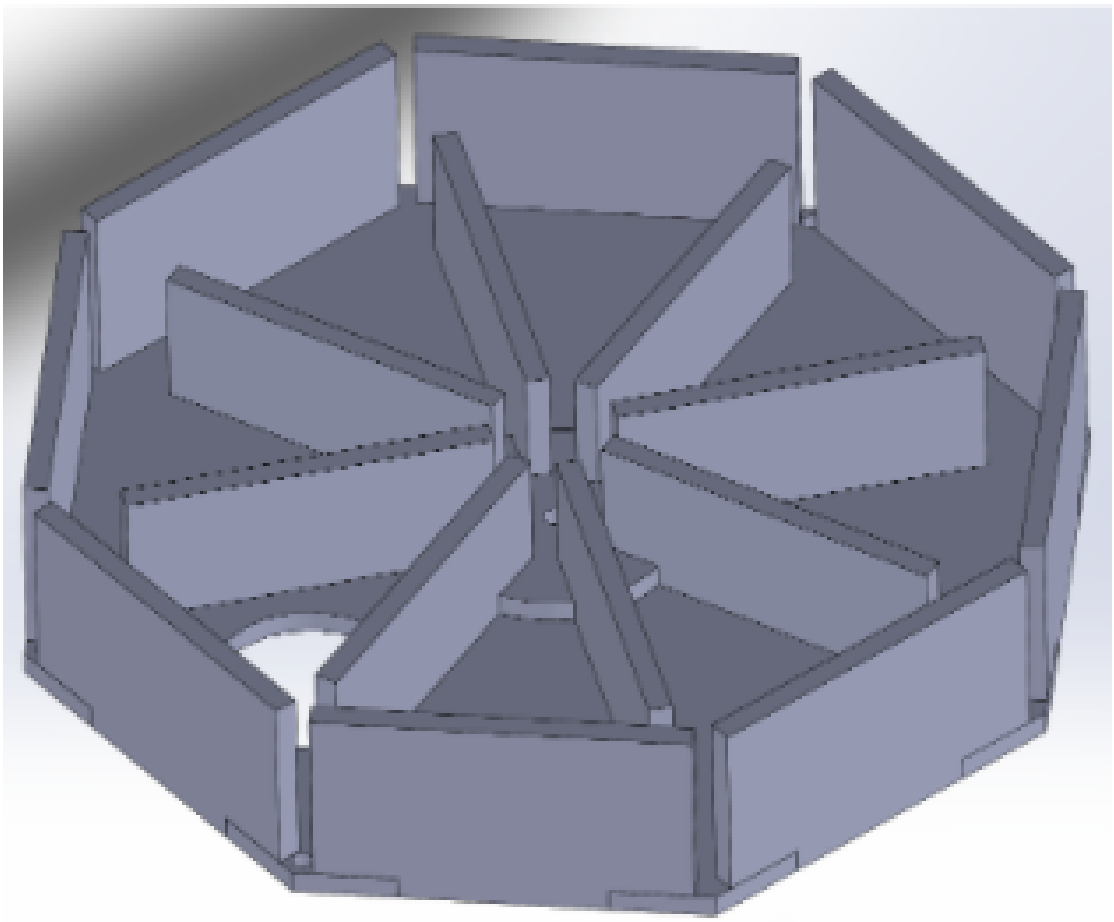


Figure 31: *Isometric view of Constructed Carousel Design.*

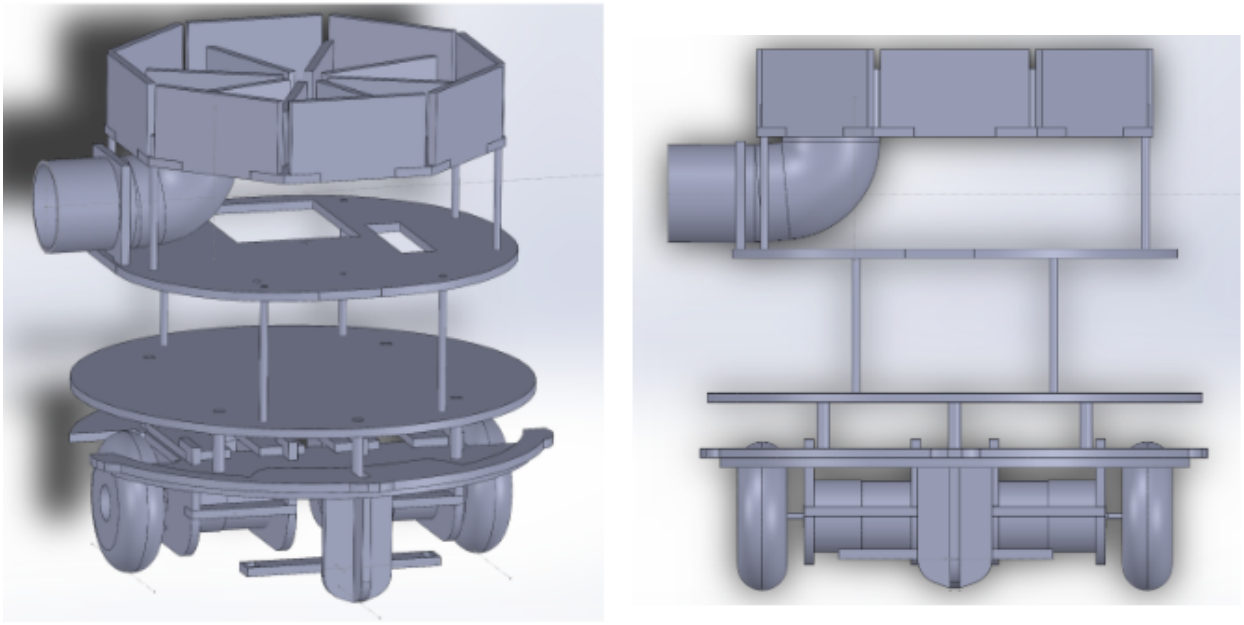
4.10.1 Difficulties and adjustments

Designing the carousel proved to be a real challenge as balls would often not fall through the 40mm hole. This is due to the fact that the ball would often miss the hole when passing over. We tried to correct this issue with software by slowing the stepper's speed. We found that the best way to correct this issue was by sanding the rim around the 40mm cutout and slowing down the stepper's speed. This combination allowed for a smooth operating carousel mechanism.

4.11 Fully Assembled Bot

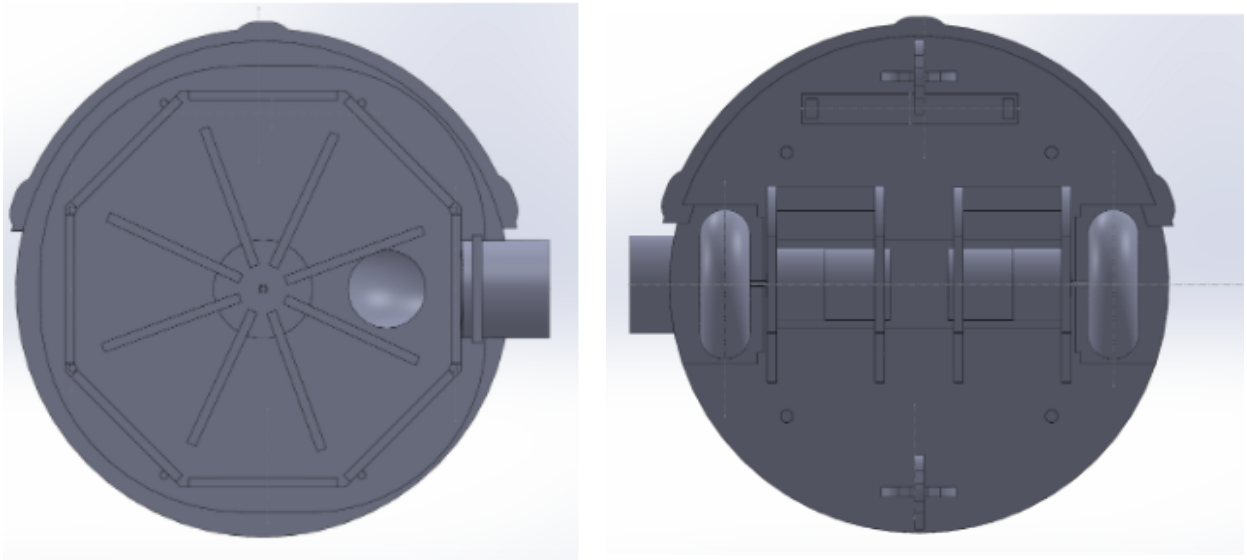
Our fully assembled bot is shown in Figure 32a. The height our bot is 10.91in, just below the height maximum of 11in. Furthermore, our bot's diameter is within the 11in by 11in area.

Overall, we were extremely pleased with our assembled bot. Implementing detachable motor mounts and platforms proved to be quite useful in our bot's assembly. It allows for quick access to the motors thanks to our tab and slot design and allowed us test components separately. Furthermore, we were also proud of the fact that we implemented a circular design that allowed us to closely navigate around the tower without obstruction. Figure 32b,Figure 33a, and Figure 33b showcase the Side, top and bottom views. Our finished bot design with mounted electrical components and sensors is shown below in Figure 34a and Figure 34b.



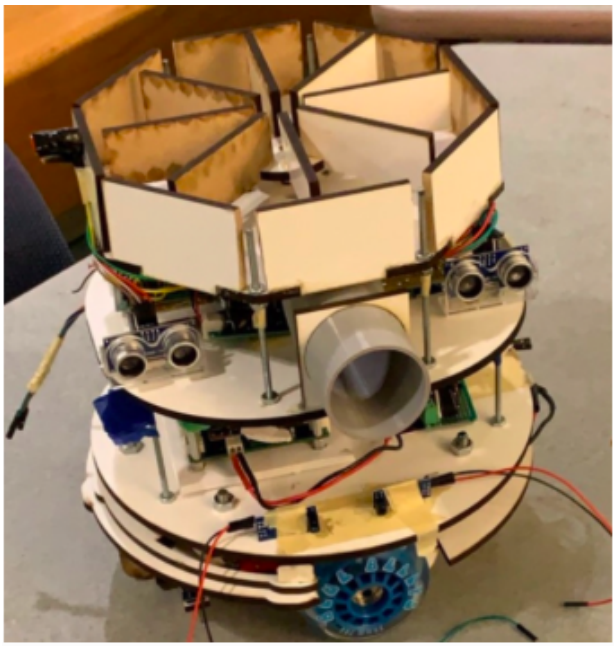
(a) (b)

Figure 32: *Isometric and Side view of Full Bot Design.*

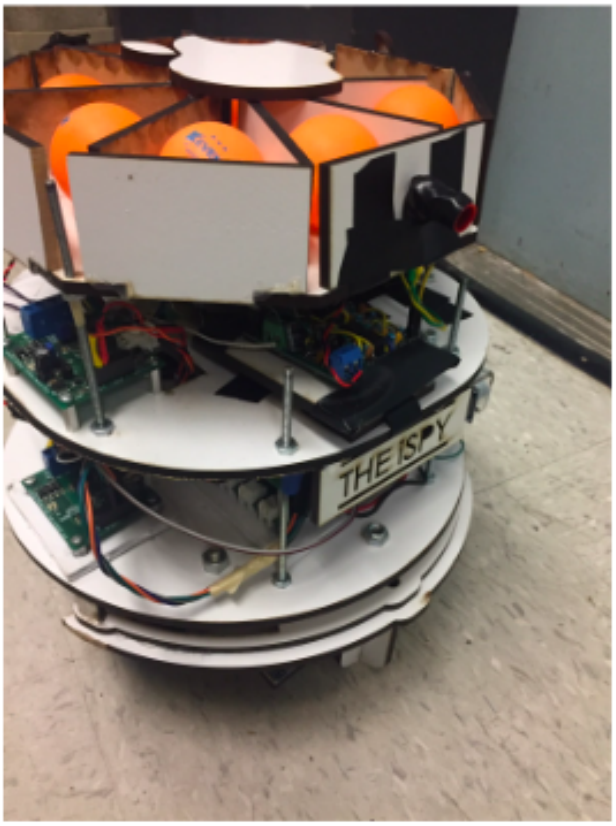


(a) (b)

Figure 33: *Top and Bottom view of Full Bot Design.*



(a) Min Spec Design



(b) Competition Ready Bot

Figure 34: *Photos of our final bot design at two different design stages.*

4.11.1 Difficulties and adjustments

One major difficulty we encountered when assembling our bot was dealing with wire management. When we first began assembling our bot we did not consider shortening the length of the wires. As a result, wiring became messy and it became difficult to determine the various connections to different components. To correct this issue, we cut the wires to length and used tape to bundle up corresponding sets of wires. We also used heat shrink to cover any open wires to prevent a possible short.

5 Software Implementation

The software implementation of the bot started with coding a test harness for each of the sensors and/or actuators we planned to use. Initially, the test harnesses used blocking code for experimenting how the hardware works and checking if the hardware works. Subsequently, the test harnesses were formatted to use the timers of the Events and Services framework provided. We organized the different sensors and actuators in created libraries to maintain an easy, readable and available list of commands and tools in the form of functions. The main control of the robot is running with the Events and Services framework as a finite state machine service. All of the readings and events from the sensors are kept a track of through the Events and Services framework as individual and different event checkers. All of the software implementation of the robot and state machine was written in the language C.

Software implementation was split between the creation of the main state machine in the Events and Services framework, and the testing and implementation of the sensors and libraries. The process and flow of programming started with the test harness implemented with blocking code, then the Events and Services framework. The libraries for the sensors were created and tested using the same test harnesses. The state machine was created in the framework as a service using the actuators and sensors libraries implemented as event checkers. Finally, after the implementation of the state machine and libraries, the program was only tweaked and debugged with minor adjustments for minimum specifications.

5.1 Test Harness

Similar to Lab 0, The Roach, we created and used a test harness to test if our actuators (DC motors and stepper motor) and sensors (beacon and trackwire detector, IR sensors, switches for bumps, and ping or proximity sensors) worked. We implemented a test harness that first, used blocking code and then, used the Event and Services framework.

The components were tested in the order of when the part arrived or was available. The ping sensors were tested first because they were already available to us and the degree of difficulty of implementing them would require more time for setup and debugging. The order of components that were tested using the blocking code implementation of the test harness: ping sensors, DC motors, switch sensors, IR sensors and the stepper motor. The beacon detector and trackwire detector were not run through the blocking code implementation of the test harness because they were confirmed to work initially through an oscilloscope and an LED soldered onto the board of the beacon detector and trackwire detector. The beacon detector and trackwire detector were tested within the Events and services framework implementation of the test harness.

5.1.1 Blocking Code

The blocking code used a helper function `delay()`. This function uses a while loop that increments a counter `i` up to the given number `ticks`. The program should run the `delay` function and increment the counter as fast as the CPU of the UNO32. The problems with getting an accurate time using blocking code will be addressed further below (sec. 5.1.3).

```
void delay(int ticks) {
    int i = 0;
    while(i < ticks) {
        i++;
    }
}
```

The `delay` function helped with printing the values of the test component through the terminal from the DS30 loader. Without adding blocking code for delays, the DS30 loader will be spammed with print statements. The `ticks` we used to experiment with the test harness harness and components was 300.

For this test harness, we individually tested each sensor and actuator component individually by using `#ifdef` to use specific sections of code or manually commenting in and out certain sections of code to use for the test harness. Individually testing each component, we were able to use the sensors and actuators in ideal condition without needing to worry about the current drawn for each part.

5.1.2 ES Framework – Event Checker

The Events and services implementation of the test harness only uses a single event checker file. The event checker holds all of the functions that generate events from the sensors. For this test harness, since we know that the DC motors and stepper motor work with blocking code, we did not test the motors and actuators. We only tested if the sensors are able to trigger the desired events. To run this test, we used the `EVENTCHECKER_TEST` macro.

The ES framework event checker test was to experiment if we got readings from the sensors correctly, how the events are triggered, and what thresholds are required for triggering events. From experimenting, we learned that the switch sensors used for the bumpers did not need to be be debounced. We took the first reading of the bumper changing states as an event. The thresholds that were changed frequently through experimentation were for the IR sensors, the ping sensors, the beacon detector and the trackwire detector.

5.1.3 Difficulties and adjustments

From testing with the blocking code, we could not accurately calculate the value of an `int tick` in relation to a unit of time. This is because the test harness used `printf()` statements, which is a blocking code itself that affects the overall speed of the program. With the print statements that were used in the test harness, we estimated that a single `tick` is equivalent to 1 microsecond. The number is still inaccurate as the number of print statements in the test harness varies per test run through the program.

Initially, the bumper events and ping sensor events were setup through their own services. To limit the number of services and have extra services for sub-level states if needed, we implemented the bumper events and ping sensor events with the event checker.

5.2 Libraries

We decided to create separate libraries to house every function of each component, sensor and actuator. The libraries were to organize and document the uses of every components. Each of the functions were created after experimentation of using the test harnesses. The libraries contained an `INIT()` function that connects the sensor output to an input pin into the UNO32 using the given library `IO_Ports.c/h`. The `INIT()` function also initializes the output pin that activate the drivers and motors. Since the libraries were created with reference to the code used in the test harnesses, specifically the Event checker implementation, the integration of the libraries, event checker and state machine was not difficult.

The libraries also provided full documentation of the description of the function, the parameters, return types and who wrote the code. The functions also provided a consistency in coding, as the library functions are called only in states or event checkers, and do most of the work. This helped clean the state machine. The main reason in creating libraries for the components is to keep the overall code clean and consistent. The libraries that were created for the parts were: `Beacon.c/h`, `Bumper.c/h`, `Motors.c/h`, `Ping.c/h`, `Tape.c/h`, and `Trackwire.c/h`. The libraries that were given and modified were `Stepper.c/h` and `timers.c/h`.

We kept a log of the I/O for the UNO32 pin listings and the component it interacts with to keep both physcial wiring and coding conistent.

```
PORTV
    PIN3 - Ping L Trigger
    PIN4 - Ping L Echo (Blue)
```

```
    PIN5 - Ping R Trigger
    PIN6 - Ping R Echo (Green)
    PIN7 - TrackWire
    PIN8 - FREE

PORTW
    PIN3 - Gun Tape Sensor Read 1(L)
    PIN4 - Gun Tape Sensor Read 2 (R)
    PIN5 - Floor Tape Sensor Read FL (Green)
    PIN6 - Floor Tape Sensor Read FR (Yellow)
    PIN7 - Floor Tape Sensor Read B (Purple)
    PIN8 - Beacon Detector

PORTX
    PIN3  - Gun sensor enable out
    PIN4  - FREE
    PIN5  - FREE
    PIN6  - FREE
    PIN7  - FREE
    PIN8  - FREE
    PIN9  - FREE
    PIN10 - FREE
    PIN11 - Gun sensor enable out
    PIN12 - FREE

PORTY
    PIN3  - Front Left Bumper Read
    PIN4  - FREE
    PIN5  - Front Right Bumper Read
    PIN6  - FREE
    PIN7  - FREE
    PIN8  - FREE
    PIN9  - Motor B (Left) Direction
    PIN10 - Motor B (Left) Enable
    PIN11 - Motor A (Right) Direction
    PIN12 - Motor A (Right) Enable

PORTZ
    PIN3  - Trackwire enable read
    PIN4  - FREE
    PIN5  - FREE
    PIN6  - FREE
    PIN7  - Stepper Enable
    PIN8  - FREE
    PIN9  - Stepper Steps
    PIN10 - FREE
    PIN11 - Trackwire enable out
    PIN12 - FREE
```

5.2.1 Difficulties and adjustments

A difficult part of keeping track of the libraries was during the implementation of the main state machine. When a new function is created in a library, a function name is modified or a function is removed, keeping track of the calls in the state machine and header files can get complicated. Since we used multiple libraries for each component, a larger problem was keeping track of the I/O pins used from the UNO32. The solution to this was simply keeping track of the pins in a separate document, as mentioned above. The creation of multiple libraries didn't introduce any new difficulties; it has mostly helped rather than hinder.

The main difficulties of libraries were in the modification of already existing libraries. With the `Stepper` library, we had experience from Lab 3, Motors. The problems with modifying this library was changing the preset pins that the library already defines. We had to check which free pins worked with the library. With the `timers` library, we needed to modify it to implement a microsecond timer to increase the accuracy and response of the ping sensors. We modified this library by changing the amount of timers and changing the frequency from the existing `#define` 's in the code. We also created a new function that gets the time in microseconds.

The last adjustment from the libraries was omitting the library `RC_Servo.c/h`. This library uses the timer `TIMER_3`, which conflicts with the `Ping` library, as it also uses the same timer. Since we planned on not using a servo, we decided not to add the library into the project.

5.3 Ping Sensors

The most difficult software implementation of the sensors was the proximity or ping sensors. The difficulties of the ping sensor were mainly from the lack of a microsecond timer and setting up an input capture. From ECE 167 (Sensing and Sensor Technologies), we created a similar piece of code that gets a microsecond count that modified `timers.c`. The function required the ISR to count microseconds at a different scale than the given `FreeRunningTimer`, which is in milliseconds. From ECE 167, we added the line:

```
unsigned int TIMERS_GetMicroSeconds(void) {
    return (microSecondCount + TMR5 / 5);
}
```

We modified the ISR to add the microsecond counter `microSecondCount`:

```
void __ISR(_TIMER_5_VECTOR, ipl3auto) Timer5IntHandler(void) {
    INTClearFlag(INT_T5);
    FreeRunningTimer++;
    microSecondCount+=1000;
    char CurTimer = 0;
    if (TimerActiveFlags != 0) {
        for (CurTimer = 0; CurTimer < NUM_TIMERS; CurTimer++) {
            if ((TimerActiveFlags & (1 << CurTimer)) != 0) {
                if (--Timer_Array[CurTimer] == 0) {
                    TimerEventFlags |= (1 << CurTimer);
                    TimerActiveFlags &= ~(1 << CurTimer);
                }
```

```
            }
        }
    }
}
```

Since we planned not to use the `RC_Servo` library, we used `TIMER_3` for the ping sensors. For implementing the input capture, we used a change notice pin and ISR, specifically we used pins 4 and 6 from port V, which correspond to change notice pins CN5 and CN7, respectively. Using two ping sensors, we had a flag that alternates which ping sensor is setting the trigger high and the other sensor reading the echo. We had the ping sensors alternate between triggering and reading echo so that the ping sensors do not interfere with the other's readings. The initialization of the `Ping` library:

```
char PING_Init(void) {
    // set timer for trigger
    OpenTimer4(T4_ON | T4_SOURCE_INT | T4_PS_1_64, 8);
    INTClearFlag(INT_T4);
    INTSetVectorPriority(INT_TIMER_4_VECTOR, 3);
    INTSetVectorSubPriority(INT_TIMER_4_VECTOR, 3);
    INTEnable(INT_T4, INT_ENABLED);
    TRISBbits.TRISB4 = 0; // V5
    TRISBbits.TRISB2 = 0; // V3

    // set input capture change notify for echo
    mCNOpen(CN_ON, CN7_ENABLE | CN5_ENABLE, 0); // CN7 = V6, CN5 = V4
    int temp = PORTB;
    ConfigIntCN(CHANGE_INT_ON | CHANGE_INT_PRI_1);
}
```

5.4 Final State Machine

For the final implementation of the state machine, we decided to recreate as a single finite state machine. The finite state machine still uses the sub-states from the initial design of the hierarchical state machine mentioned above, but the sub-states are now a series of states. In the state machine in Figure 35, the sub-states are color coded, where the tape avoidance states are green/orange, the bumper states are blue, the tower/trackwire traversal are yellow, the wall tape states are red, the shoot states are grey, and the escape states are purple.

The final state machine is linear as it performs each action in a process mostly determined by timers. Some of the states include an `ES_TIMEOUT` back to the DETECT_BEACON1 state in case it is trapped in performing the linear actions. The STALL state and similar states that stall have been added to reduce stalling in the motors to prevent motors from quickly changing direction.

The goal of the state machine is that it repeats the process of finding a beacon and depositing a ball into the correct hole. The robot detects and traverses to a beacon while avoiding tape obstacles. Finding a beacon, it traverses around it to detect the correct side with the trackwire. At the correct face of the tower, it sweeps forward and reverse to detect the tape under the correct hole in the wall. After fully depositing a ball, it traverses around the beacon while searching for other beacons in the process. This is to avoid detecting the beacon it is currently nearby.

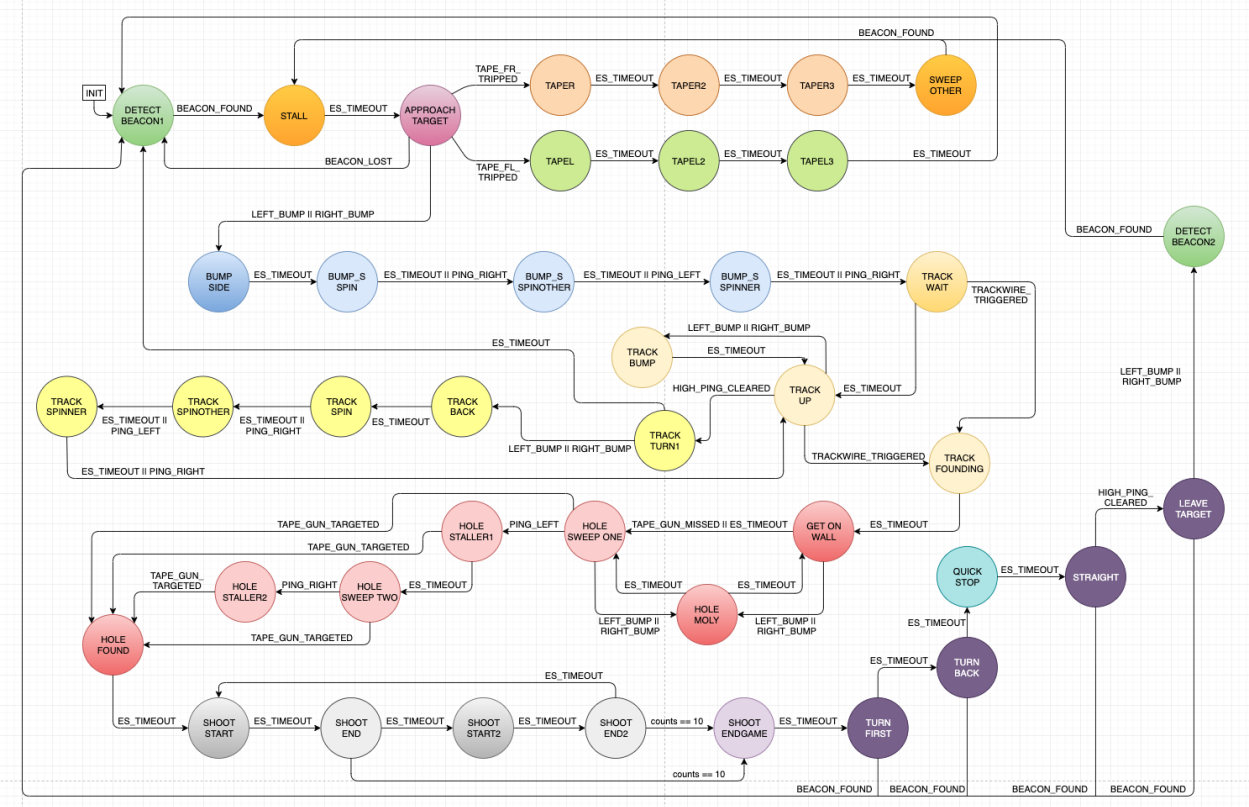


Figure 35: *Final Finite State Machine. [3]*

5.4.1 Beacon Detection and Tape Avoidance Process

`DETECT_BEACON1, STALL, APPROACH_TARGET, TAPEL1, TAPEL2, TAPEL3, TAPER, TAPER2, TAPER3, SWEEP_OTHER`

In Figure 36, we zoom in on the states for beacon detection and traversing. The bot only checks for tape obstacles on the floor when it has detected a beacon and is approaching the beacon. In DETECT_BEACON1, the bot tank turns in a counter-clockwise direction at a 70% speed of the motors. When it receives the event that is has found a beacon, it stops for 250ms and drives forwards at APPROACH_TARGET at 90% motor speed. At anytime in APPROACH_TARGET, it can detect an event triggered from if its left tape sensor or right sensor detects black tape. TAPEL and TAPER follow a similar process in which the bot reverses and turns and drives towards the opposite direction of the tape. Because the robot turns, it searches for the beacon again at the end of the tape avoidance process. SWEEP_OTHER is a state similar to DETECT_BEACON1, but spins in the clockwise direction opposite from the original beacon detection.

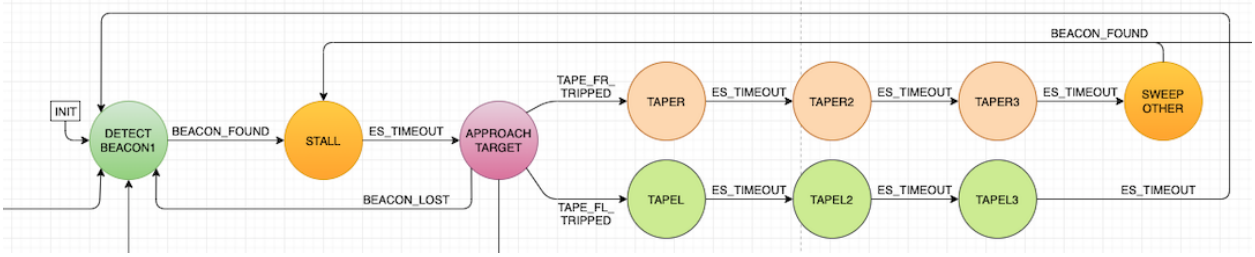


Figure 36: *Tape triggered States. [3]*

5.4.2 Beacon Found and Bumping Process

`BUMP_SIDE, BUMP_S_SPIN, BUMP_S_SPINOTHER, BUMP_S_SPINNER`

In Figure 37, we zoom in on the states for finding a beacon. In APPROACH_TARGET, when a bump event is triggered, the bot assumes that it has found a beacon tower. The bumper process is when it has found a tower, the bot attempts to parallel park along the side that it as bumped. In BUMP_SIDE, the bot reverses away from the tower so it has space to make turns. In BUMP_S_SPIN, the bot tank turns clockwise to align the right ping sensor parallel to the wall. In BUMP_S_SPINOTHER, the bot tank turns counter-clockwise to align the left ping sensor parallel to the wall. In BUMP_S_SPINNER, the bot tank turns clockwise to realign the right ping sensor parallel to the wall again. The process is also done with timers in the case that the bot hits a corner and does not detect the wall, so the bot does not spin forever attempting to align a ping sensor with nothing.

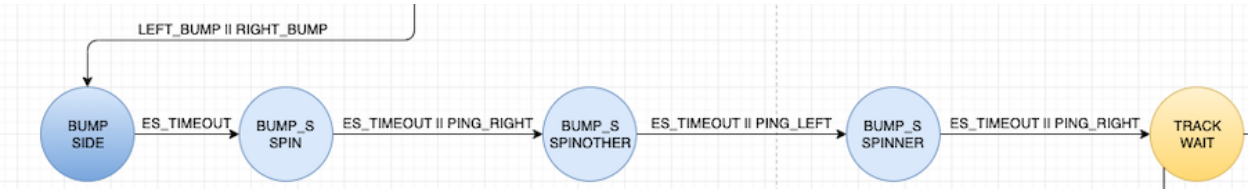


Figure 37: *Bumper States. [3]*

5.4.3 Tower Navigation and Trackwire Process

`TRACK_WAIT, TRACK_FOUNDING, TRACK_UP, TRACK_BUMP, TRACK_TURN1, TRACK_BACK`
`TRACK_SPIN, TRACK_SPINOTHER, TRACK_SPINNER`

In Figure 38, we zoom in on the states for navigating around the tower to detect the correct trackwire side. After the bumper process for the bot to make itself parallel, in TRACK_WAIT, the bot stops and checks if it found the correct side. If not, it goes into TRACK_UP. In this state, the bot will check if it receives the event that the trackwire side was triggered. When navigating around the tower, the bot may lose its parallel position to the tower. If the bot is facing towards the tower and bumps, TRACK_BUMP re-parallels by turning away from the wall slightly so that it does not bump again when moving forward. If the bot is facing away from the tower, the ping sensors will not detect the tower and trigger an event to TRACK_TURN1, where the bot hard turns by pivoting back towards the tower. If the bumper event is triggered during the hard turn, then the bot attempts to re-parallel itself similar to the bumper process mentioned above.

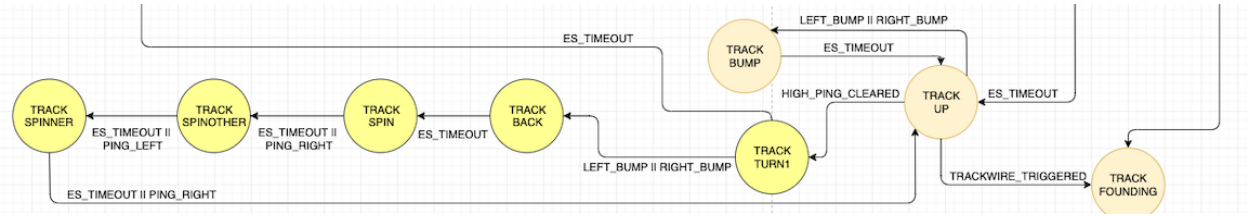


Figure 38: *Finding the trackwire States. [3]*

5.4.4 Sweeping and Finding Hole Process

`GET_ON_WALL, HOLE_MOLY, HOLE_SWEEP_ONE, HOLE_SWEEP_TWO, HOLE_STALLER1, HOLE_STALLER2, HOLE_FOUND`

In Figure 39, we zoom in on the states for finding the correct hole with the tape. GET_ON_WALL is to make sure that we initially see the white MDF before attempting to find the black tape, in the case that the correct tape is triggered by empty space, since empty space is around the similar threshold value as black tape. HOLE_MOLY works similar to TRACK_BUMP, when the bot gets too close to a wall and loses its parallel position. HOLE_SWEEP_ONE, the bot drives forwards waiting for the side tape event to be triggered. If a ping event is triggered, then the bot has travelled off the wall and must reverse into HOLE_SWEEP_TWO, which is similar to the first one but in reverse. The two STALLER states are for a timer based movement when the ping sensor event is triggered. The states move the bot a little further forward or reverse to ensure that the tape sensor does full detect tape if it is further out.

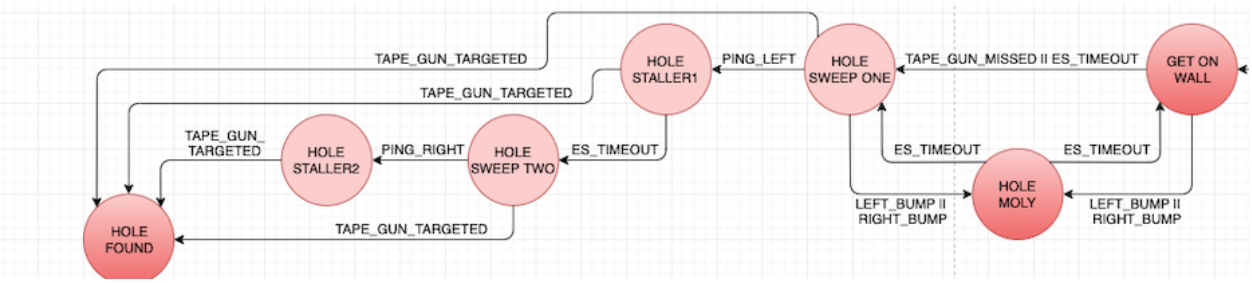


Figure 39: *Finding the correct hole States. [3]*

5.4.5 Shooting a ball Process

`SHOOT_START, SHOOT_END, SHOOT_START2, SHOOT_END2, SHOOT_ENDGAME`

In Figure 40, we zoom in on the states for shooting a ball. The states for ball launching is similar to a PWM, where the enable for the stepper is set high in the START states and low in the END states. It is set high for 20ms and low for 200ms, so that the stepper moves a small amount. The shoot states cycle until a global counter reaches 10 counts. The counter is incremented at any time during a START state and checked during an END state. 10 counts allows the bot to accurately shoot 3 balls. After shooting 3 balls, SHOOT_ENDGAME stops all of the motors and prepares the bot for the next beacon.

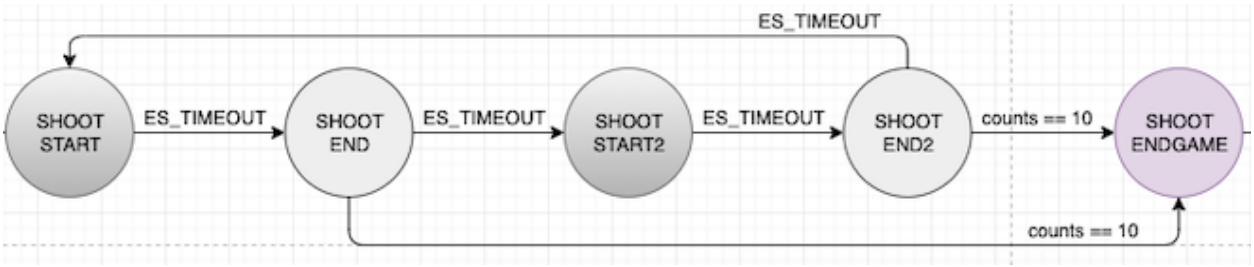


Figure 40: *Shooting States. [3]*

5.4.6 Escaping and Finding Next Beacon Process

`TURN_FIRST, TURN_BACK, QUICK_STOP, STRAIGHT, LEAVE_TARGET, DETECT_BEACON2`

In Figure 41, we zoom in on the states for finding the next beacon. TURN_FIRST and TURN_BACK are for a complete 180°tank turn and back away from the current beacon. STRAIGHT drives the bot straight and away from the beacon after its full turns. After a ping sensor event is triggered when driving straight away, in LEAVE_TARGET, the bot navigates around the beacon but not facing the beacon detector towards the current beacon. The bot then sweeps, tank turning away from the beacon in DETECT_BEACON2. During the process of the 180°turn, driving straight, navigation and sweeping, the bot is constantly checking for a beacon change event to detect the next beacon without detecting the current one.

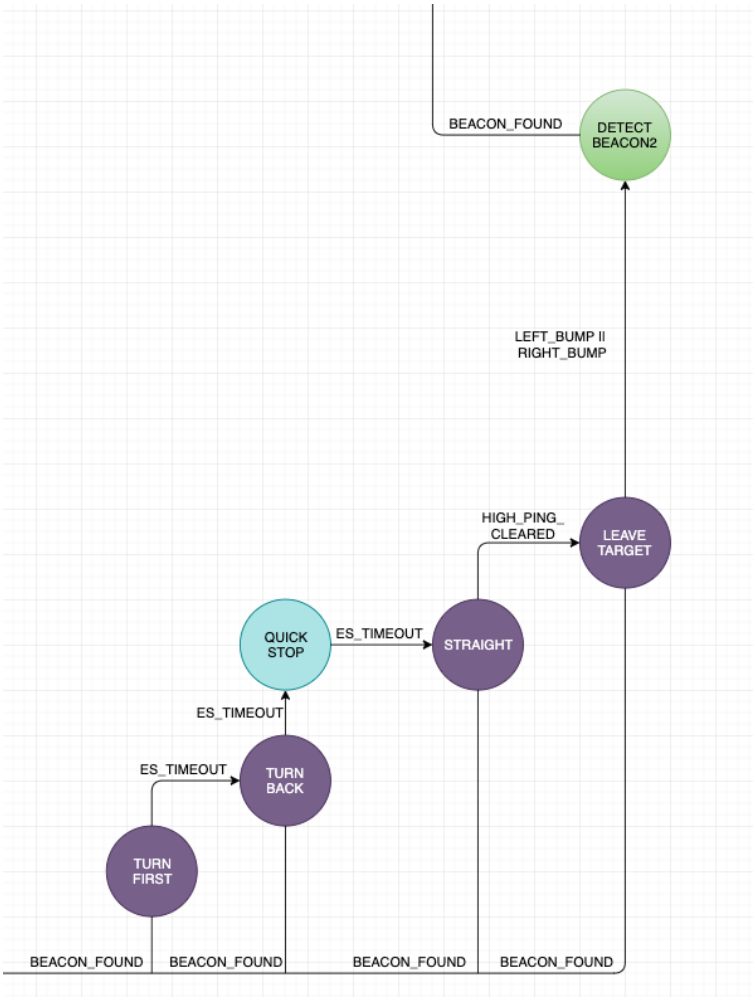


Figure 41: *Finding the next beacon States. [3]*

5.4.7 Difficulties and adjustments

The first hurdle during the implementation of the state machine was using a hierarchical state machine. When using the HSM, some of the timers for sub-levels worked correctly, while some were not triggered at all. We also found it difficult to pass events from a higher level to one of its sub-levels through an easy and efficient method. A lot of the problems when using an HSM were fixed or didn't exist when we transferred to a finite state machine implementation. The largest negative of using an FSM is that the FSM code is longer and more cumbersome to navigate around.

Another adjustment to the state machine is the addition of extra filler states or stall states for continuation of a process. In the original state machine, a single state was used for the animation or movement of the bot as sub-states, but in an FSM, all of the states are added in a linear form with a similar name or tag to ensure that it is a part of the series of sub-states.

6 Billing List

Item	Quantity	Model	Brand	DataSheet	Link	Price
Wheels	2	N/A	Blue Bellies	N/A	BLUEBELLIES	$6.00
DC Motors	2	B072R57C56	Greartisan	N/A	MOTORLINK	$30.00
Bump Sensors	6	V-155-1C25	TOUHIA	N/A	BUMPERS	$7.00
Servos	4	SG90	Deego	N/A	SERVOS	$10.00
Tape Sensors	10	TCRT5000	OSOYOO	N/A	TAPE_SENSOR	$12.00
Ping Sensors	5	HC-SR04	Smraza	N/A	PINGS	$10.00
Stepper Motor	1	NEMA 17	StepperOnline	N/A	STEPPER	$15.00
MDF	4		BELS	N/A	N/A	$20.00
3M Screws	8	N/A	ACE	N/A	N/A	$2.00
Wheel Inserts	2	N/A	John	N/A	N/A	$2.00
Screw Terminals	10	N/A	BELS	N/A	N/A	$2.00
Hex Nuts/Bolts	20	N/A	ACE	N/A	N/A	$10.00
Boost Converter	2	DSN6009	XL6009	N/A	BOOST	$7.00
Threaded Rods	4	N/A	ACE	N/A	N/A	$10.00
TOTAL						**$143.00**

7 Videos

Tape Navigation
Tower Navigation
Parallel Tower navigation
Full Test with 2 Towers
Final Minimum Specification Checkoff

8 Conclusion

Using a combination of electrical engineering skills, mechanical design, and embedded programming, we were able to successfully build a robot that performed minimum specifications checkoff within a minute and five seconds. Our design relied heavily on ping sensors to traverse the beacon and utilized constructed circuit boards to detect the actual tower and determine which hole to shoot into. We found that for our system to work, iterative design was necessary, considering that there was so many aspects we had to change when we ran into things we didn't expect during testing. We often had to print out new bases to add space to remount some of our electrical components so that they are more stable or provide a cleaner layout for wire management.

Due to our experience of having to reorganize wires, or inspect certain electrical components during testing, we found that it was very useful that we had a multilayered bot attached with screws. This design allowed us to easily disconnect layers of our bot so that we can either focus on the electrical components on the secondary layers, or we had to disconnect the top and inspect the base layer which was entirely responsible for the robot's locomotion. Although there were many issues we ran into during the weeks spent working on this robot, we found that no particular issue really set us back too far, and we have always managed to adjust and stay on track. Over the past five weeks, we believe that we received good team-building experience working together to

construct such a complex system. Building this robot ultimately taught us key engineering lessons that we will use throughout the industry.

9 Acknowledgements

We would like to acknowledge professor Michael Wehner, the ECE 118 TA's, Zach Potter and Tony Li, and the group tutors for their guidance and assistance during the provided lab sections. We would like to specifically thank our mentor for this project, Omar Reyes Cisneros, as he provided useful advice and some materials for the project. We collaborated with other groups in the class, as we all helped each other to get checked off in time.

References

[1] Amazon. *OSOYOO TCRT500 Sensor Modules*. URL: https://www.amazon.com/OSOYOO-TCRT5000-Infrared-Reflective-Photoelectric/dp/B07C69N65P/ref=redir_mobile_desktop?_encoding=UTF8&psc=1&ref=ppx_pop_mob_b_asin_title#detail-bullets (visited on 11/16/2019).

[2] Digi-Key. *Scheme-It: Free Online Schematic Drawing Tool*. URL: https://www.digikey.com/schemeit (visited on 03/28/2019).

[3] *Draw io*. URL: https://www.draw.io (visited on 04/03/2018).

[4] Texas Instruments. *More Filter Design on a Budget*. URL: http://www.ti.com/lit/an/sloa096/sloa096.pdf (visited on 11/10/2019).