Slugificant Seven

Barron Wong - Tianshu Ji - Jesus Bobadilla

December 12, 2018

Contents

Introduction

This project sets to combine all of the software, mechanical, and electrical techniques learned in previous labs to address an open ended question in designing an autonomous bot. The bot will need to perform several tasks based on asynchronous events that are triggered by readings from its various sensors; these various tasks include determining its orientation, traversing a course using tape sensors, avoiding obstacles, and accurately shooting ping pong balls at an enemy target. Most parts will be designed from scratch with a few off of the shelf parts consisting of a budget lower than $150.

1 Preliminary Design

1.1 Mechanical Design Drafts

To get started on the project we first had to come up with two different designs that will be able to complete the objective of the project and present them to the class to get feedback on them. Our first design was called M1N-5P3C and can be seen in the figure below.

Our first design would rely on four tape sensors, two front bumpers, and a 2kHz IR beacon detector to navigate the field and get to the initial firing zone(IFZ). For shooting, we had two pitching motors to launch the balls, a solenoid to load the balls to the pitching wheels, and a servo to adjust the angle of the launch. We also had a track wire sensor and a ping senor mounted in the front in case we decided to move closer for shooting.

Our second design was called 5P3C-TACTICAL and can be seen in the figure below.
The second design was very similar to the first design, with the exception that we would have rising platform run by a stepper motor. After receiving feedback for our design, we decided to scrap the solenoid and instead use a servo arm to load the balls to the pitching motors. Since our top priority was to complete the minimal specifications for the project, we went with our min spec design.

1.2 UNO 32 Pin Layout

Once we had a design to go off of and selected the components we would use for our robot, we created a block diagram for the UNO 32 pin layout to show what pins each component will be connected to. The pin layout changed throughout the course of the project due to adding additional components and removing others. The final layout we used for our robot can be seen in the figure below.

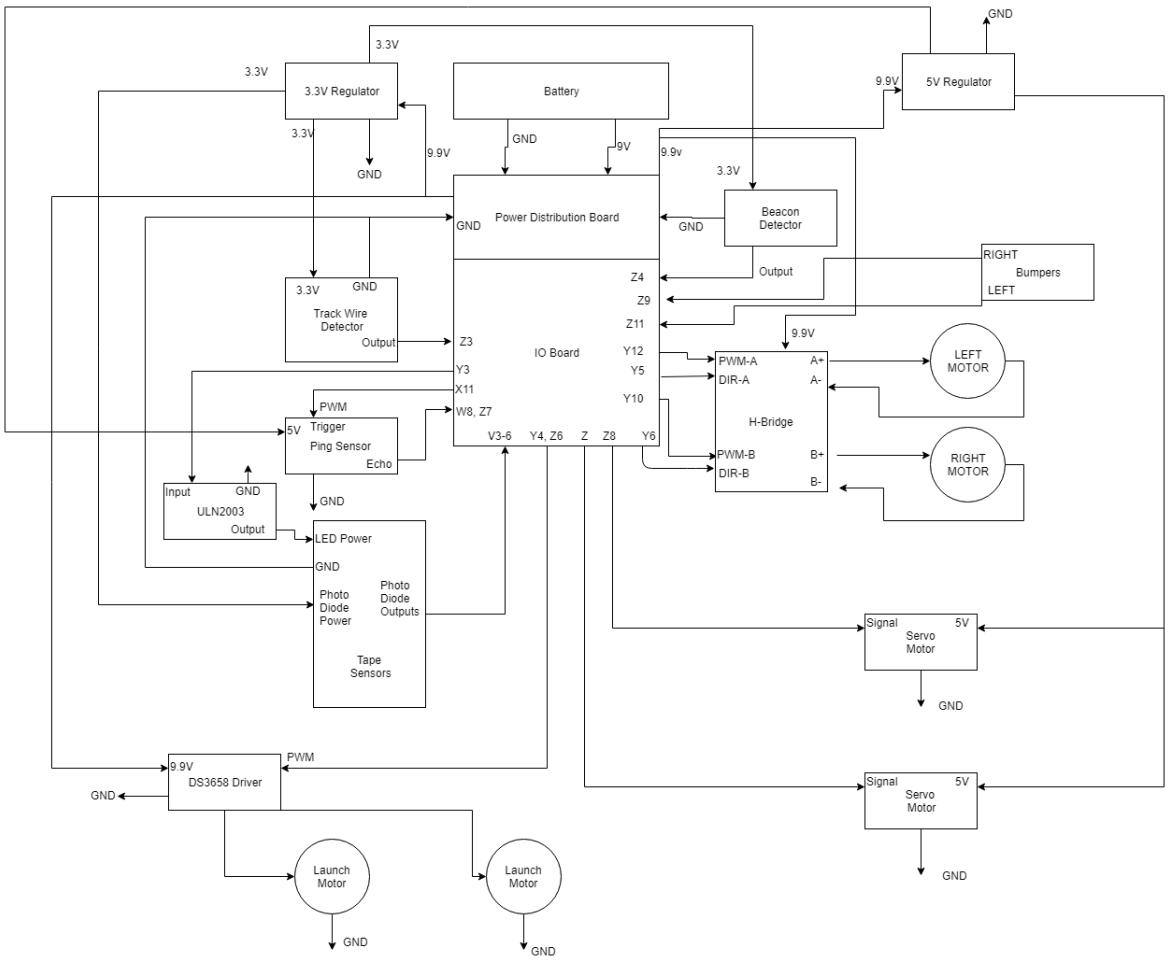


Figure 1: UNO 32 Pin Layout

1.3 Team Roles

The last thing before we started working on our robot was assign primary and secondary roles for mechanical, electrical, and software. We decided to take Professor Elkaim's advice and do what we are worst at; below are the assigned roles.

- Mechanical
  - Primary: Barron Wong
  - Secondary: Tianshu Ji
- Electrical
  - Primary: Tianshu Ji
  - Secondary: Jesus Bobadilla

- Software
  - Primary: Jesus Bobadilla
  - Secondary: Barron Wong

2 Mechanical Design

The mechanical design at first was based on the M1N-5P3C bot presented in our preliminary design. The mechanical pieces were designed and created in such a way that enabled multiple parts of the project to be developed in parallel. For example, once a motorized platform was created, the mechanical lead could start working on the top layer, while the software lead worked on the state machine.

2.1 Motorized Platform

The motorized platform[2] will serve as the bottom layer. The bottom layer houses the tape sensors, motors, and the bumpers as well as all the electronics that go with it. As shown in the preliminary design, the base is an octagon that has a width of 10 inches. This shape and size were chosen so that the bot would not have any trouble moving through obstacles while having a safety margin if other components needed to be added.

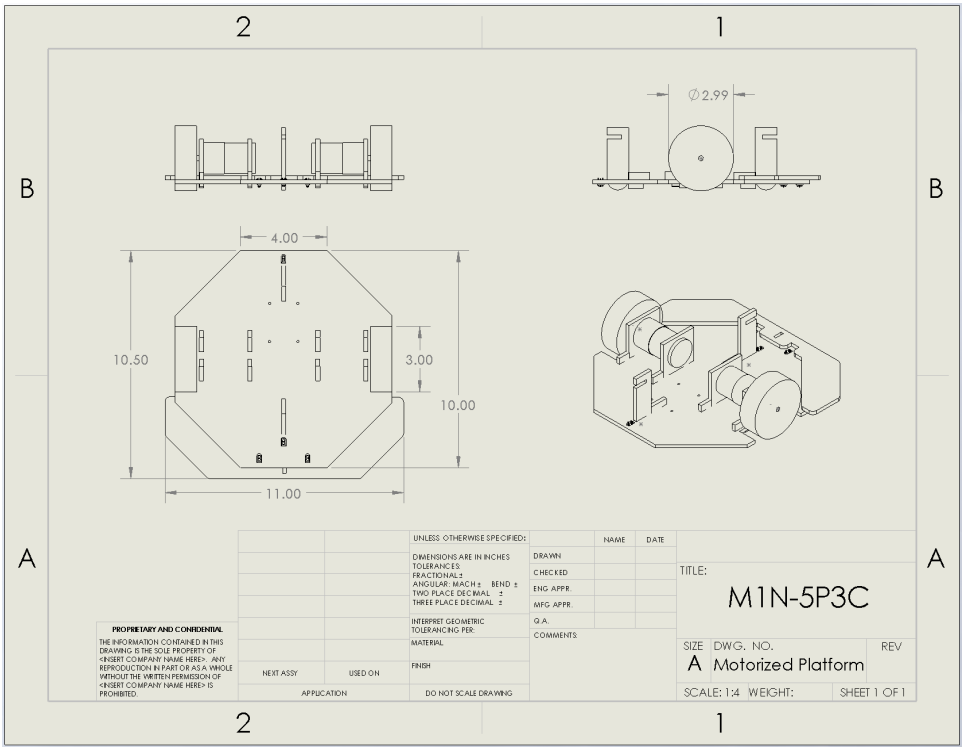


Figure 2: Motorized Platform - Drawing

2.1.1 Motor Mounts

A CAD model of the motors were created, in order to design the mounting brackets for the motors. The mounts secure the motor to the bottom layer using a friction fit. This works by creating a hole in the bottom layer where the mounts can drop into. Each mount has two legs that lock into place by sliding them to the front. Wedges are then used to lock the mounts in to keep them from sliding out[3]. The motor themselves are secured to the mounts using machined screw fasteners; this design enabled an easy way from removing the motors from the bottom layer.

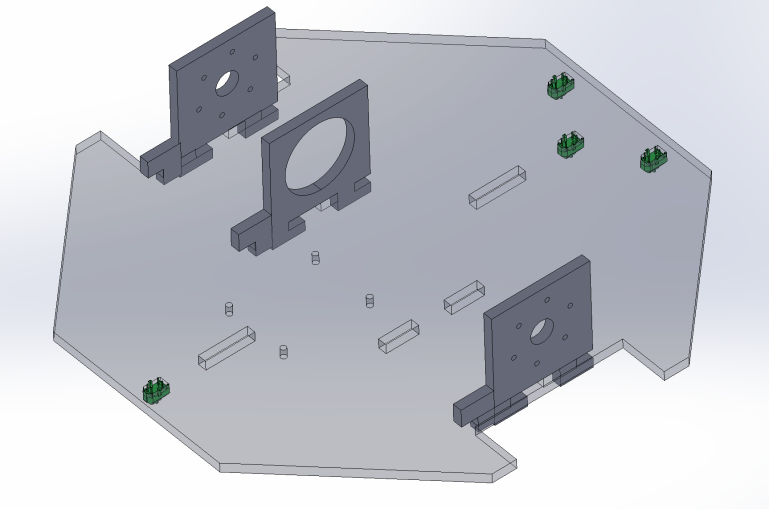


Figure 3: Motor Mount - CAD

2.1.2 Skids

Since our wheels are placed in the middle, the bot drags a portion of its weight on the floor. To minimize the amount of friction the bot would experience while driving, skids were added[4]. The skids are designed to be slightly higher than the wheels, so that they would not prevent the wheels from having the necessary amount of traction needed for movement; at first this tolerance was set to 0.1 inches higher than our wheels. When attempting to run tests on the half fields, the bot would get stuck in some places where the field itself was warped. This led to increasing the distance from the floor to our skids with the use of a file.

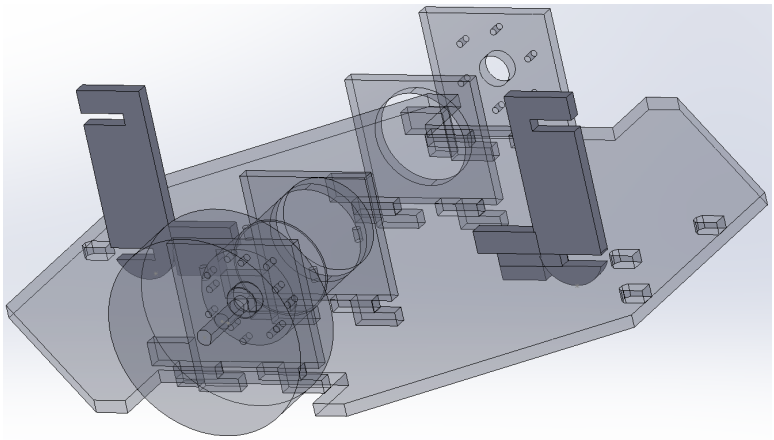


Figure 4: Skids - CAD

2.1.3 Bumpers

For obstacle detection, bumpers were implemented. Two micro-switches were used in the front to detect any head on collisions with obstacles; since the bot would always be moving forward only a single front bumper was created[5]

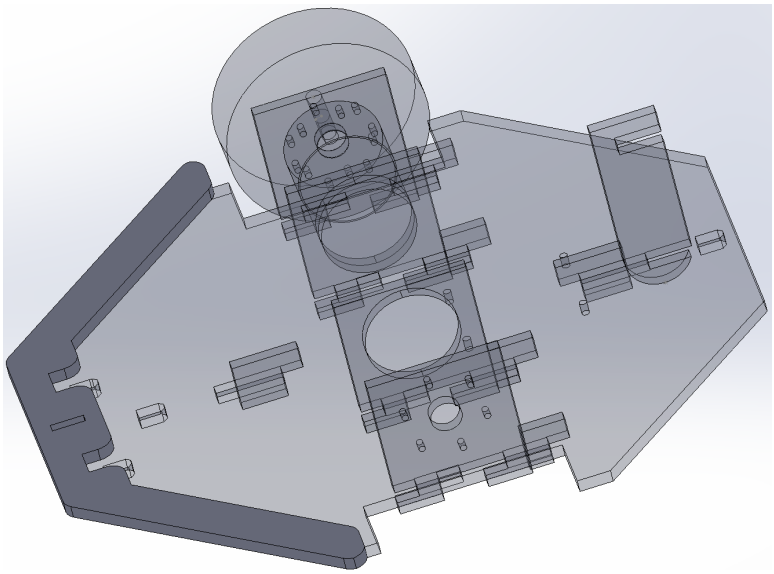


Figure 5: BumperV1 - CAD

The micro-switches were placed after the bumper was created. This was done by roughly setting the bumper and marking mounting holes to be drilled for the switches.

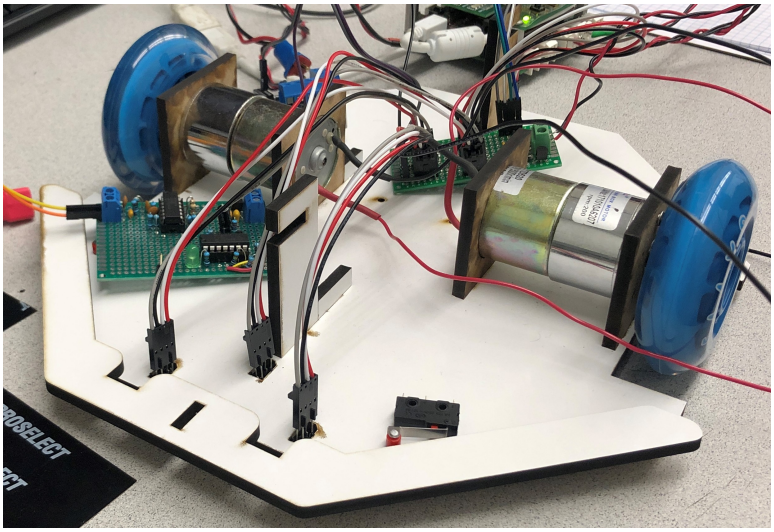


Figure 6: Bumper - Switch Mounting

Using the first iteration of our bumper resulted in the bot getting stuck in certain areas where an obstacle would barely trigger the outer edges of the bumper. To resolve this, the bumper was extended[7], this resulted in sooner obstacle detection that could be resolved more easily, given the tight spaces between obstacles.

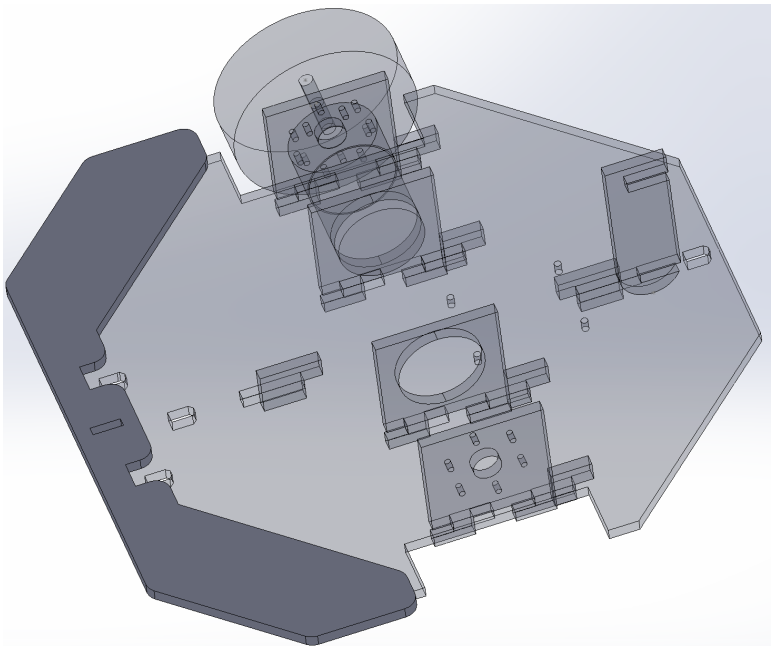


Figure 7: BumperV2 - CAD

The revision of the bumper concludes the bottom layer, the completed motorized platform can be seen in the picture below, with the components and its wiring hooked up.

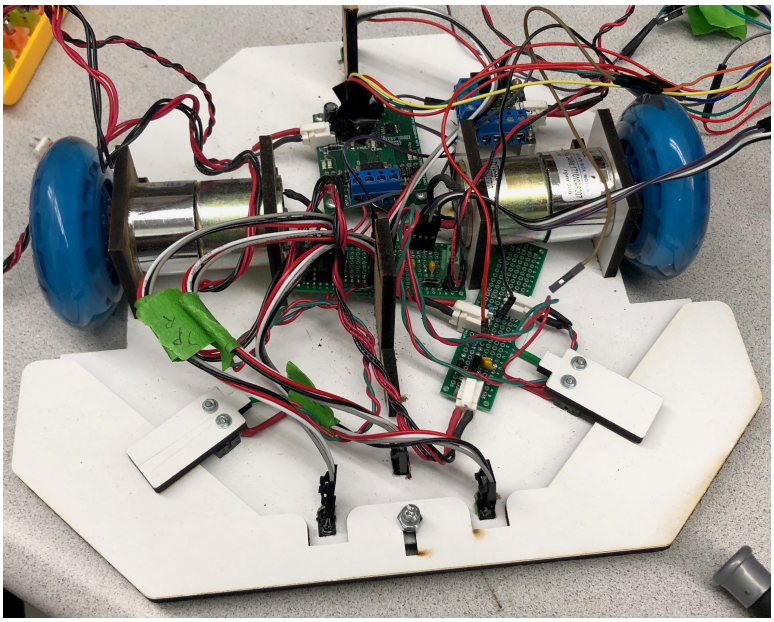


Figure 8: Motorized Platform - Complete

2.2 Top Layer

The top layer is designed to be the central location for all of the electrical components. This is where the UNO Stack and DS3658 driver is mounted. The top layer also is the mounting points for the beacon emitter and shooting platform.

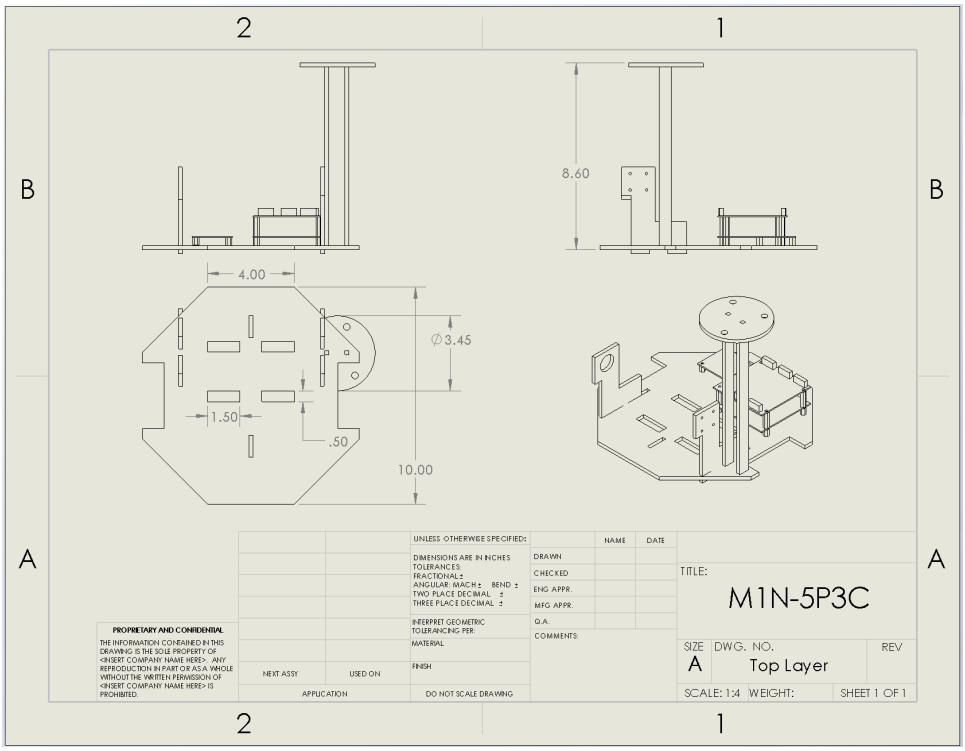


Figure 9: Top Layer - Drawing

2.2.1 Mounting

Looking back at the motorized platform, one may have noticed that the skids seemed to have an additional slot on the top for mounting[10]. The skids are used to fasten the top layer to the bottom layer with a friction fit. The size of the motor mounts and skid slots are designed to be at the same height so that all of the weight of top layer will be supported with no additional pieces. The top platform slides in and gets locked in place with wedges in the same way the motor mounts lock into the motorized platform; this leads to a ridged and light weight chassis.

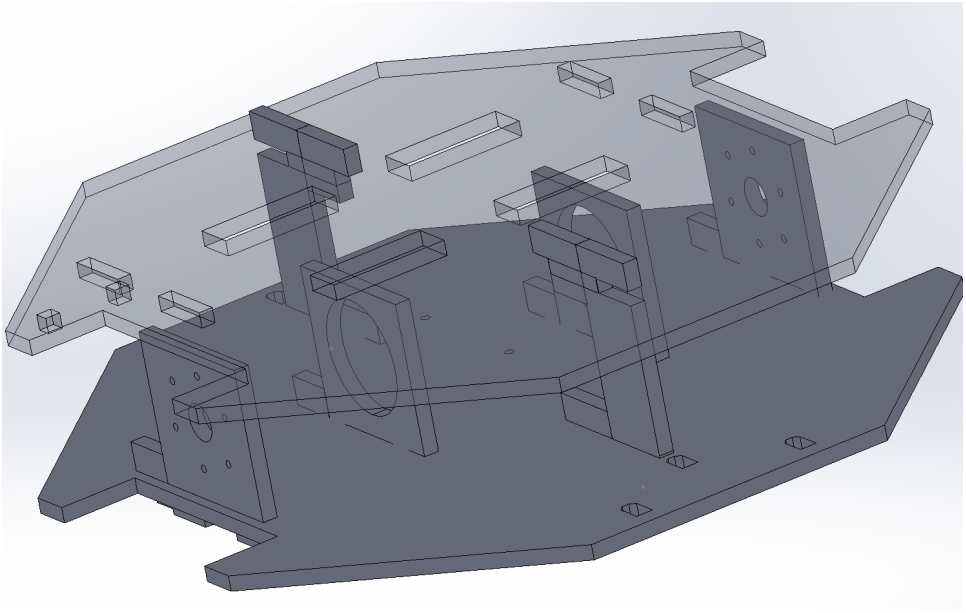


Figure 10: Top Layer Mounting - CAD

2.2.2 Beacon Mount

The beacon mount sits on the top layer, it is secured with a tab and slot design. The tab and slot design proved to be inferior to the slide and lock method used for the motor mounts and top layer. Hot glue was necessary to keep the mount in its place. This design decision was chosen out of pure laziness and was enough to support the beacon with its aluminum can; however the stand became wobbly and fragile after the tournament. The height of the beacon platform was set so that it would be exactly 10.99 inches when the motorized platform and top layer are joined together[11].

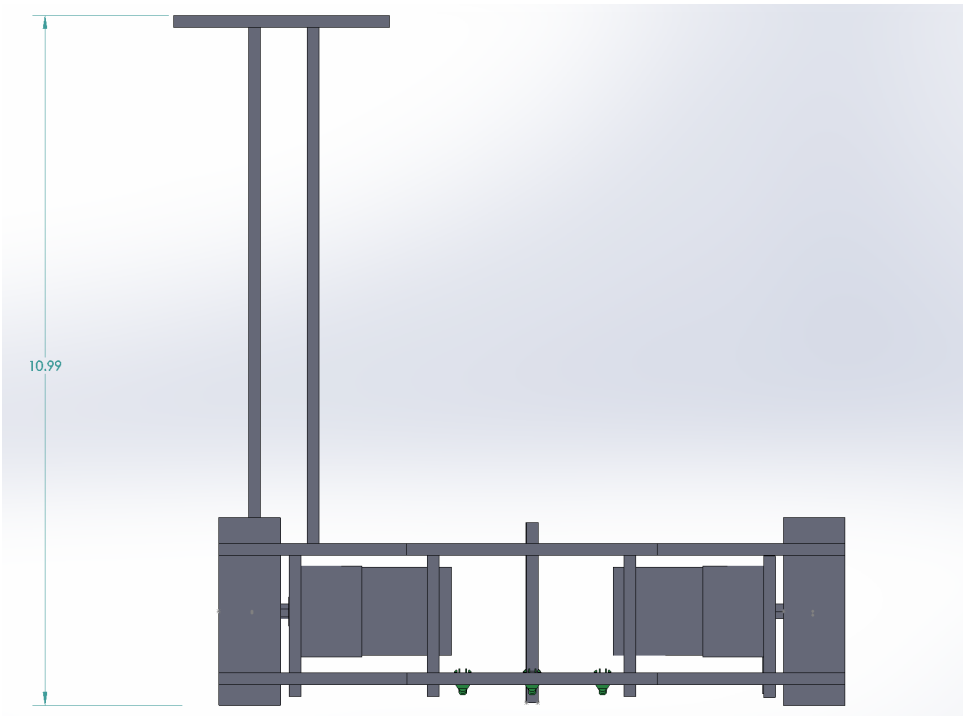


Figure 11: Beacon Mount - CAD

Being able to design the exact height of the beacon platform in CAD ensured that the bot would fit within the cube of compliance; while meeting the specifications outlined in the project manual.

2.2.3 Wire Routing

Since the top layer is the mounting place for the UNO Stack, it became necessary to have sections in our chassis for wire routing. Four rectangles were made on the top layer for just this specific purpose. This led having all of our wires contained within the bot and made wiring easy; these can be seen in Top Layer Drawing [9].

2.2.4 Firing Assembly Mounts

From the preliminary design M1N-5P3C had a stationary mounted shooting mechanism, while 5P3C-TACTICAL had a rising platform that could adjust its angle with a servo. The idea of an angle adjusting platform was adopted into M1N-5P3C. In order to mount the firing platform to the top layer two distinct mounts were created. On one side a servo will need to be adapted and the other will house a bearing to reduce the load on the servo when angling the platform. The mounts were created with the slide and lock method, so that entire firing platform can be dismounted from the top layer easily.

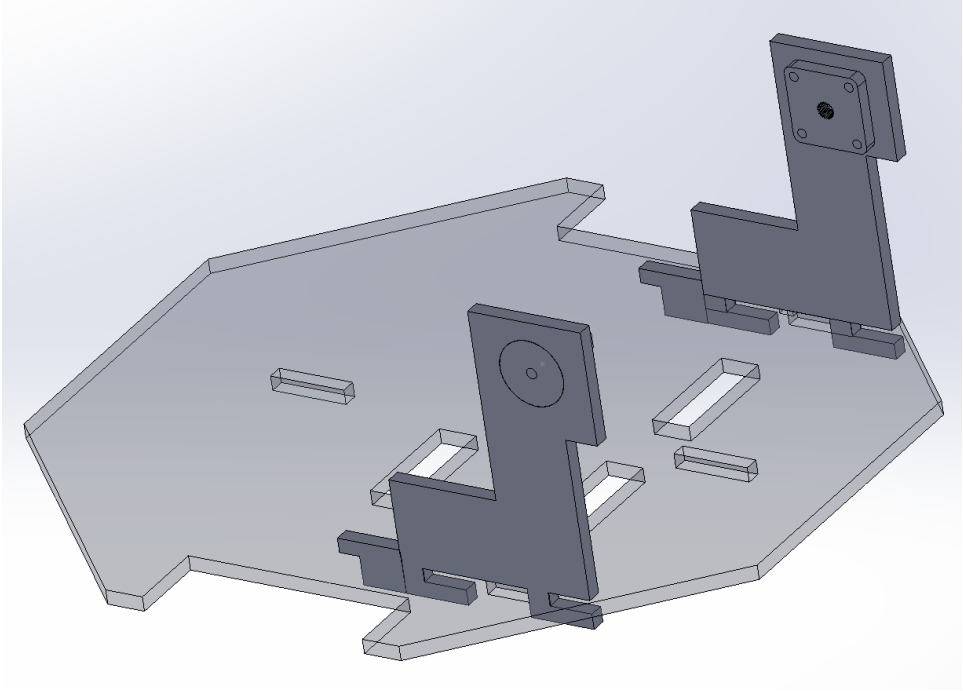


Figure 12: Firing Assembly Mount - CAD

The adapter for the servo was first designed as a ring gear that fit the servo's output shaft. This design was made out of acrylic and since out servo could produce 20kg/cm of torque, the ring gear's teeth were destroyed over time; this issue was resolved using a metal clamp.

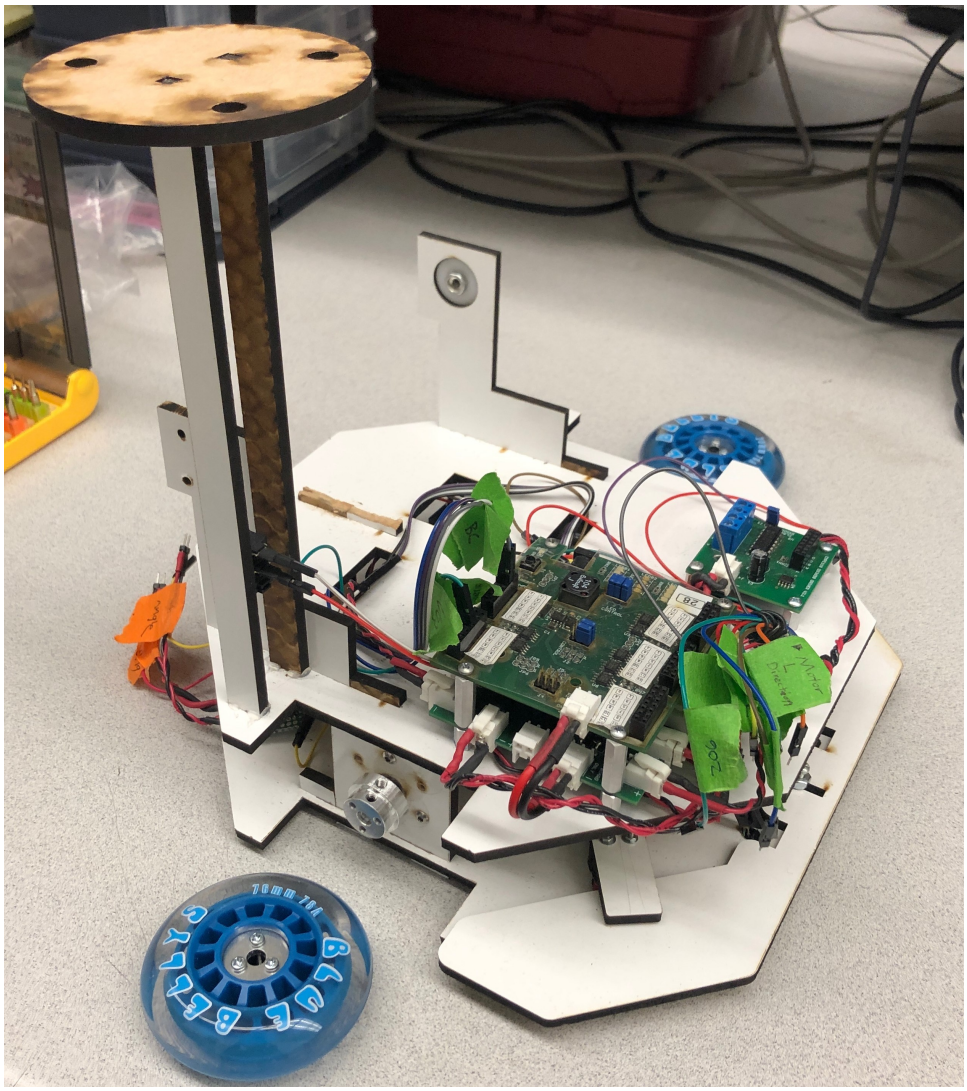


Figure 13: Top Layer - Complete

The figure above shows the completed top layer along with how the wires are routed though the holes created.

2.3 Firing Assembly

The firing assembly consists of multiple parts that allow M1N-5P3C to hold, load, aim, and shoot ping pong balls. As mentioned above, the firing mechanism is a combination of both designs presented in the preliminary review.

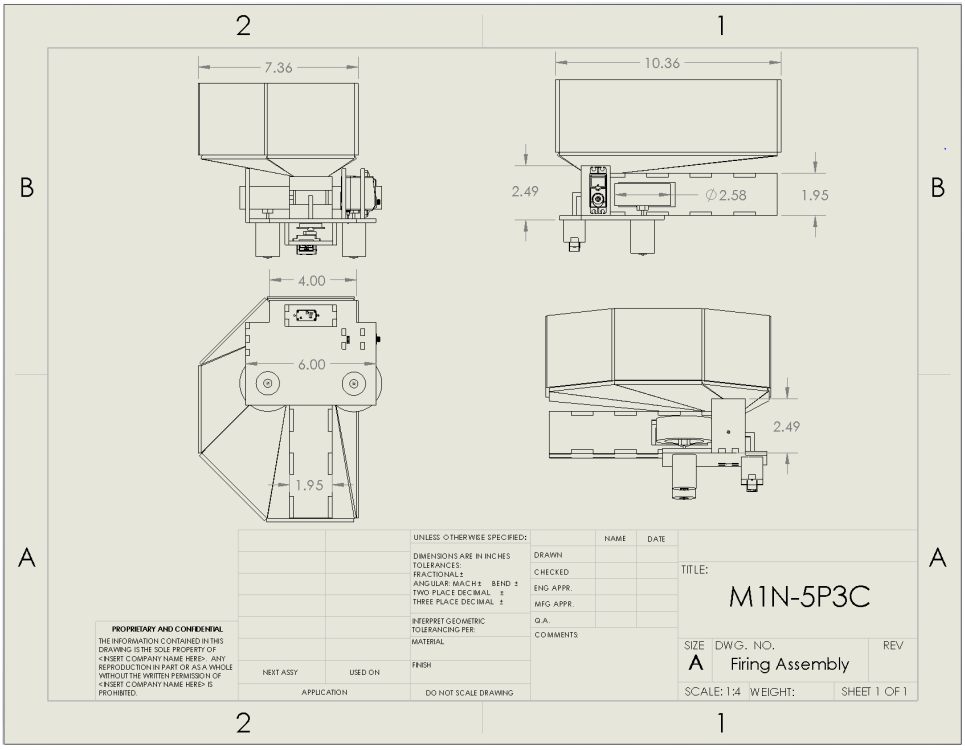


Figure 14: Top Layer - Complete

2.3.1 Firing Platform

The firing platform is the central mounting location for the firing assembly parts. The base of the platform consists of mounting brackets for the angling servo and its bearing

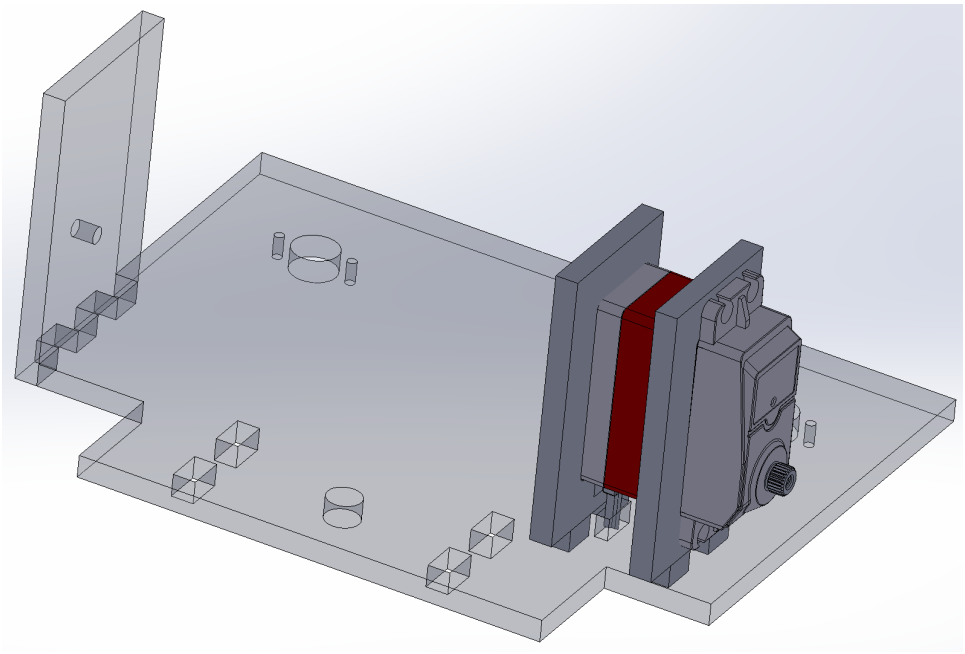


Figure 15: Top Layer - Complete

As seen in the CAD figure above, the platform consists of tab and slots for securing the platform mounts. This style of joining was chosen since the entire platform itself is removable from the main body and dismantling single components were not necessary. The angling servo was placed in its mounting brackets using a friction fit[16], so that it could be easily replaced if needed. Additional holes were placed in the firing platform for the firing mechanism.

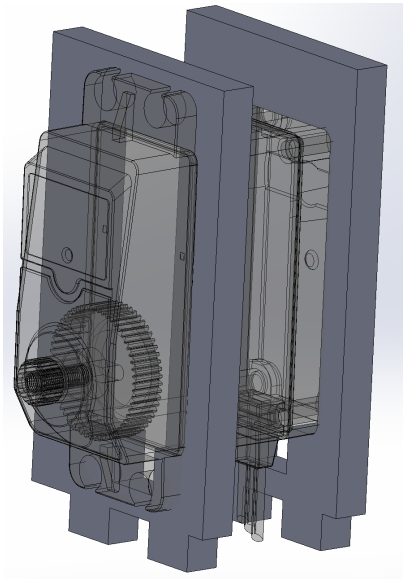


Figure 16: Servo Mounting Bracket - CAD

2.3.2 Firing Mechanism

The firing mechanism is designed using a dual pitching motor system. Since the vertical angle could be controlled using the servo on or firing platform, a decision was made to making the pitching motor horizontal; this gives horizontal control of the ball's trajectory by adjusting the speed of either the left or right pitching motor.

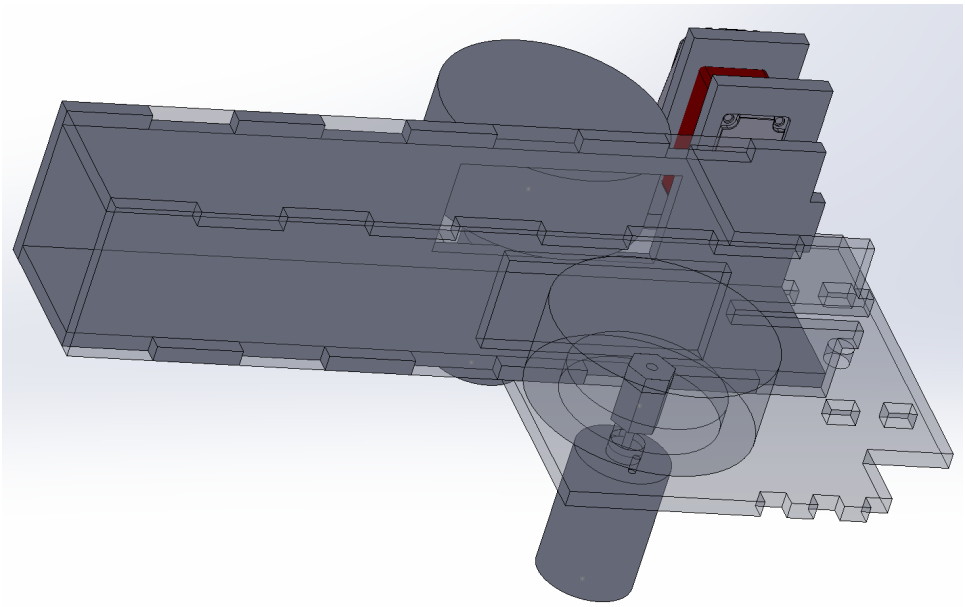


Figure 17: Firing Mechanism - CAD

Both pitching motors are 10,000 RPM DC motors. The high angular velocity made it possible for the balls to launch at a high speed. The pitching wheels consist of two performance RC car wheels; these were chosen for the stickiness of the tire compound. A square barrel was chosen to reduce the amount of surface area the ping pong ball where friction would slow it down. The spacing between the two wheels was set by fully compressing the tires and setting the distance to be the size of a 40mm ping pong ball; luckily this also made consistent shooting with 38mm wheels, since the rubber expanded when rotating at a high angular velocity

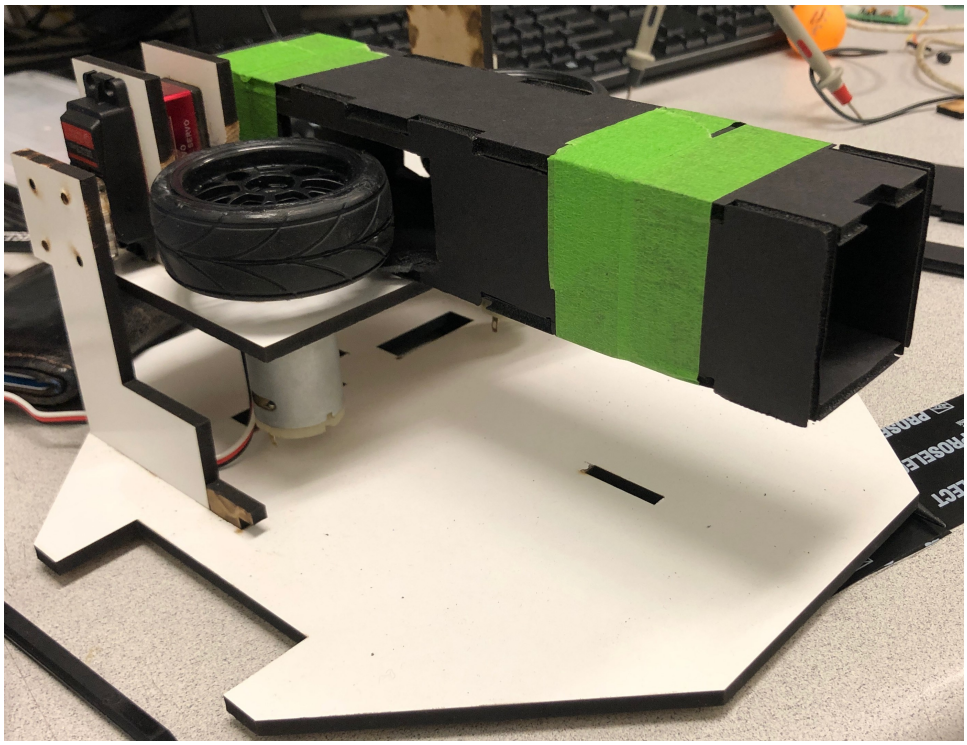


Figure 18: Firing Mechanism - Prototype

2.3.3 Ammo Hopper

From the preliminary designs, M1N-5P3C was designed to hold as much ammo as possible. The bot with no hopper has a height of only 7 inches; this leaves 36% of our vertical space for holding ammunition.

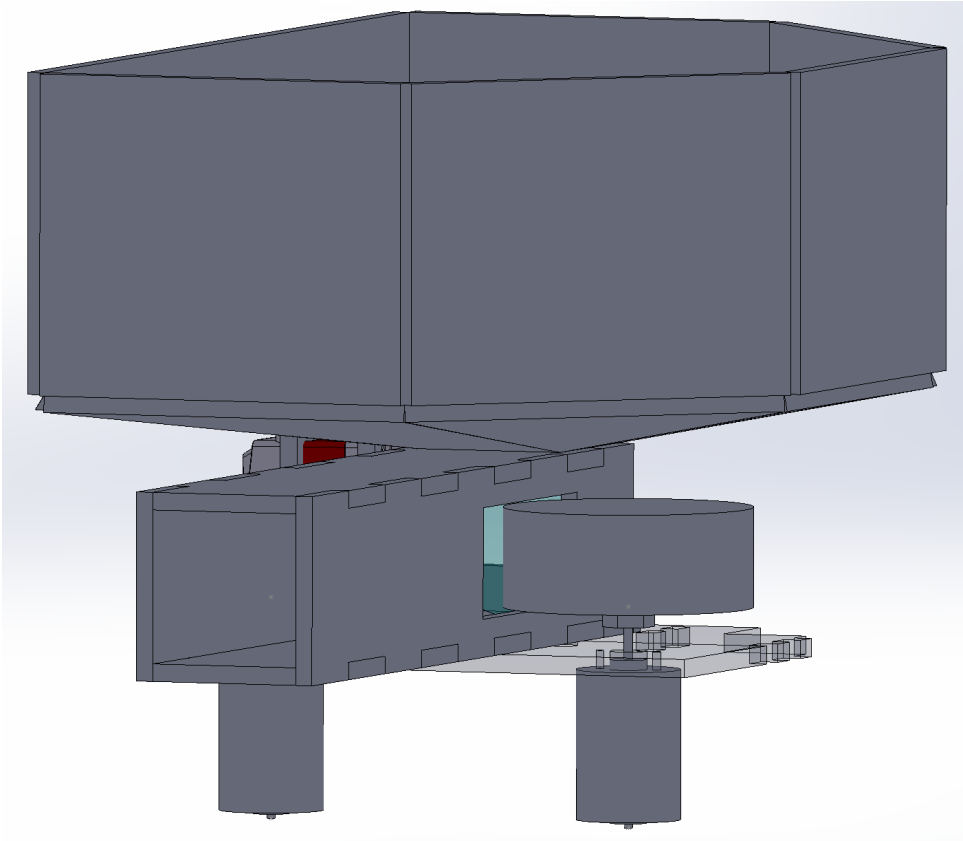


Figure 19: Ammo Hopper - CAD

Since our firing platform angles vertically, a portion of the hopper space had to be reduced so that the beacon could stay stationary at 11 inches; even with the volume reduction, the hopper was still able to hold over 30 balls.

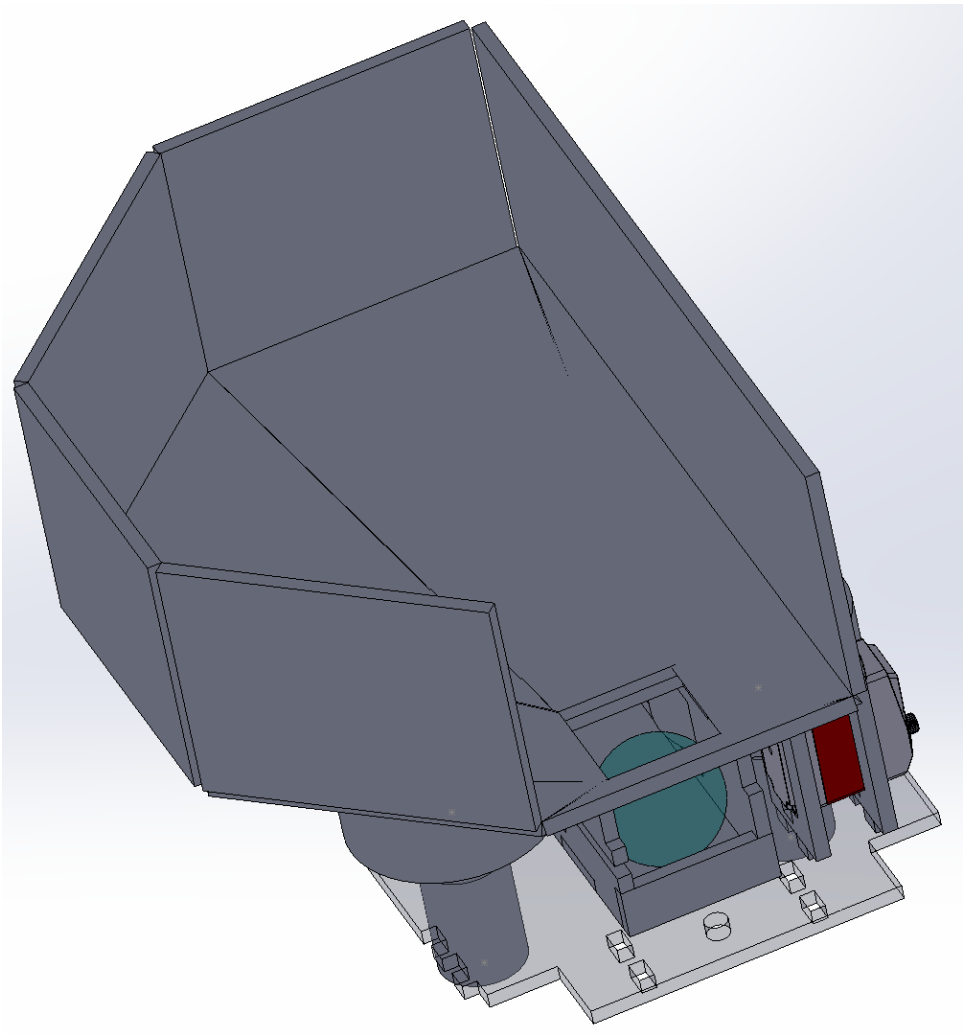


Figure 20: Ammo Hopper - CAD

From the figure above, one can see that balls are dropped into the back of the barrel using just gravity; the bottom of the hopper is sloped so that it sinks at this location. Issues of balls getting jammed were a concern when first designing this method. After implementing this design, balls rarely jammed and when they did, the vibration from the pitching motors would set them free in a reasonable time.

2.3.4 Loading and Shooting

The loading and shooting mechanism consists of a loading arm and a servo. The design is simple and only requires two ranges of motion.

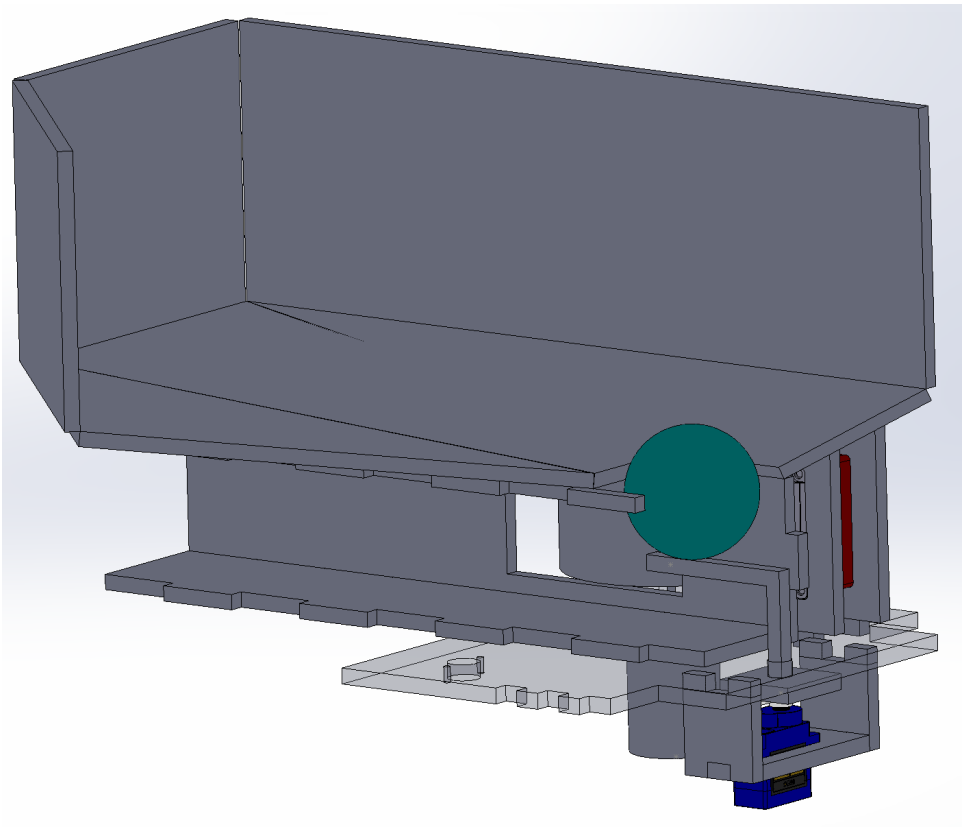


Figure 21: Load and Shooting - Initialization

The figure above shows how the system is first initialized. In the beginning no balls are in the barrel and the servo has the loading arm in the SHOOT position; this position naming will make sense on the last step. When the arm is pointing down at the barrel, it blocks any balls from entering the chamber; this helps with unintended ping pongs being fired.

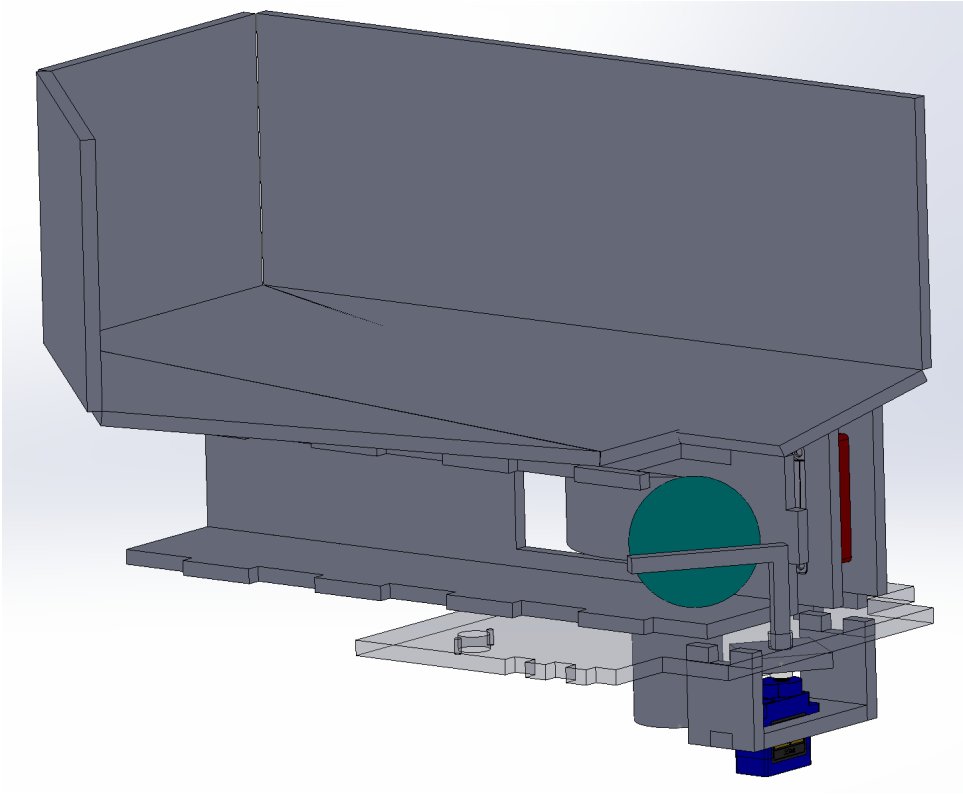


Figure 22: Load and Shooting - LOAD

To load a ball in the chamber, the servo moves the loading arm 90° counter clockwise into its LOAD position. With the arm no longer restricting the path into the barrel a single ping pong ball is gravity fed into the chamber. While in action, the barrel is always slightly angled upwards, which keeps the ping pong ball at the back of the barrel; the ping pong ball below also stops the balls from above from entering the chamber.

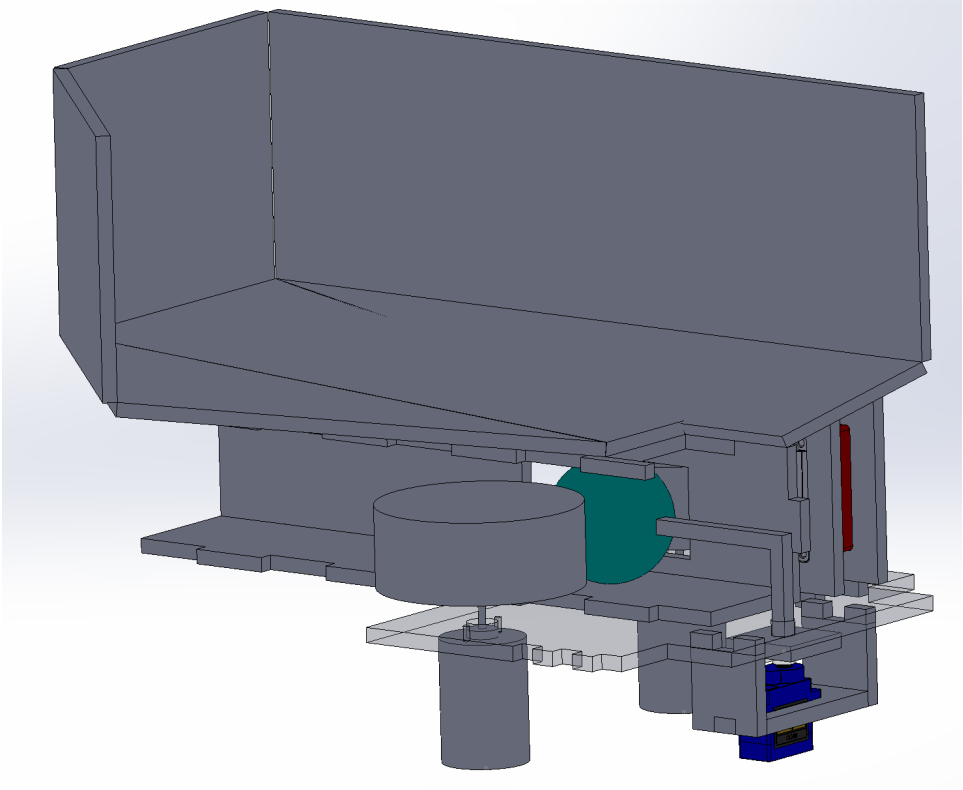


Figure 23: Load and Shooting - SHOOT

The cycle is complete once the servo moves the loading arm back into the SHOOT position. The ball is wedged between the wall and the loading arm. Since the ball is round and the loading arm is angled, it forces the ball towards the pitching wheels; once the pitching wheels contact the ball, the ball is launched out of the barrel. Back at the SHOOT position the loading arm once again stops any other unintended balls from entering the chamber. This cycle repeats itself in a periodically when an enemy is spotted.

2.4 Final CAD Design

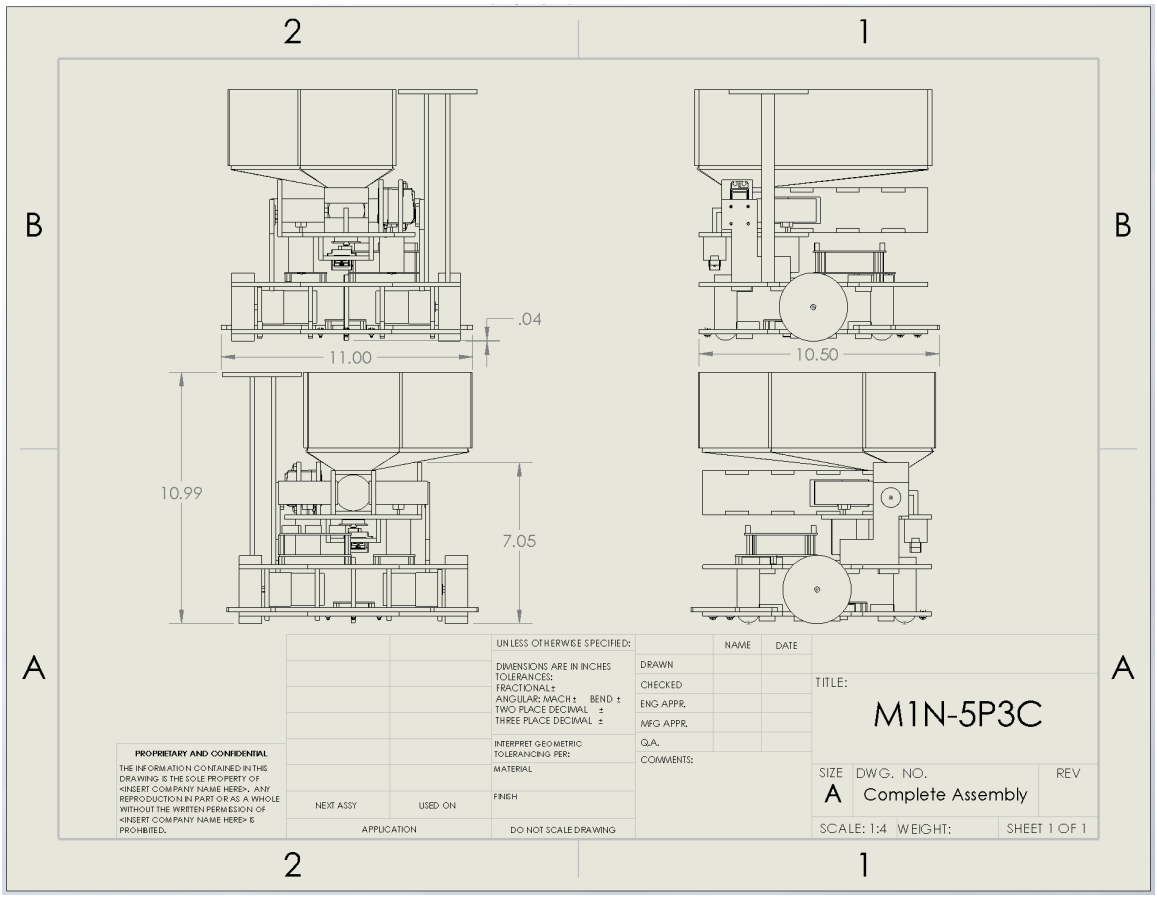


Figure 24: Final CAD Design - Drawing

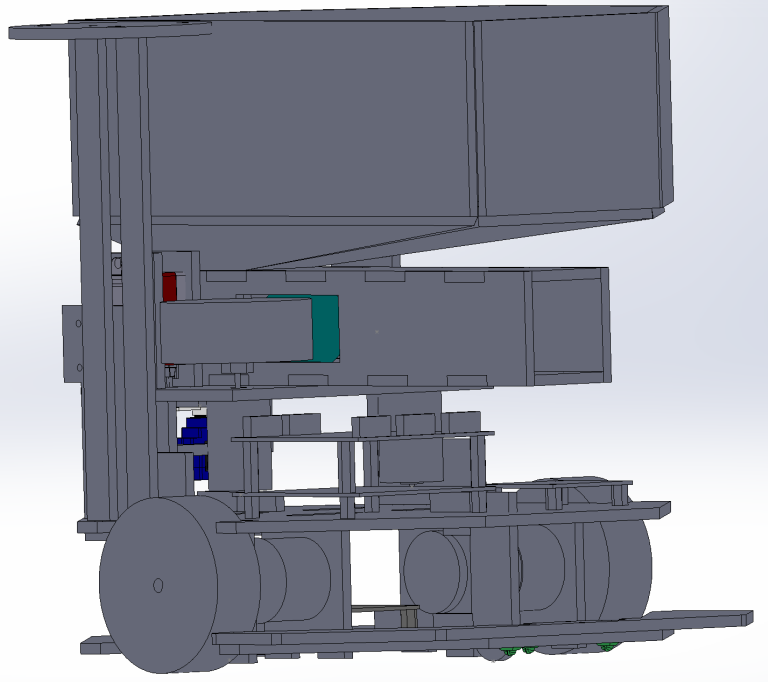


Figure 25: Final CAD Design - Front Right

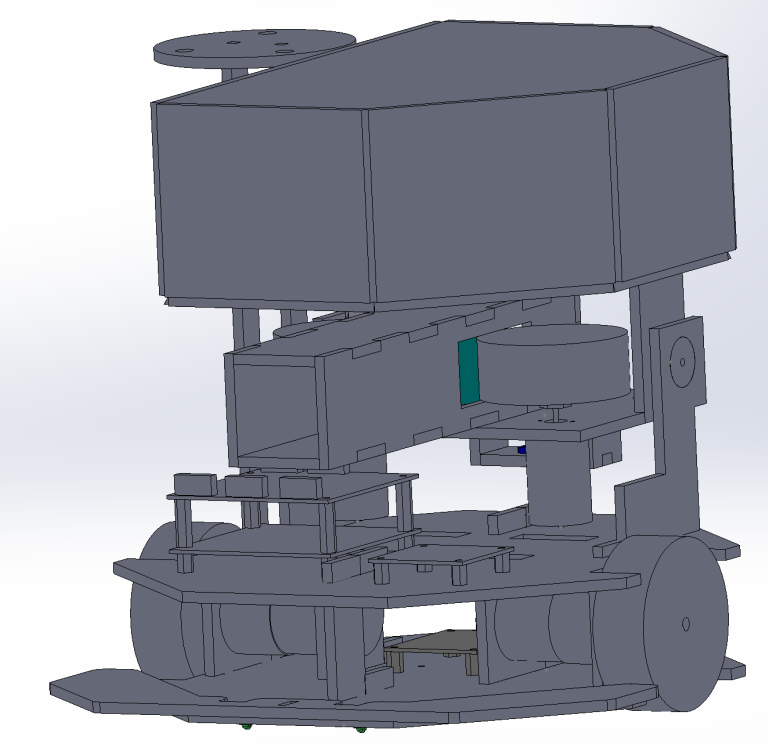


Figure 26: Final CAD Design - Front Left

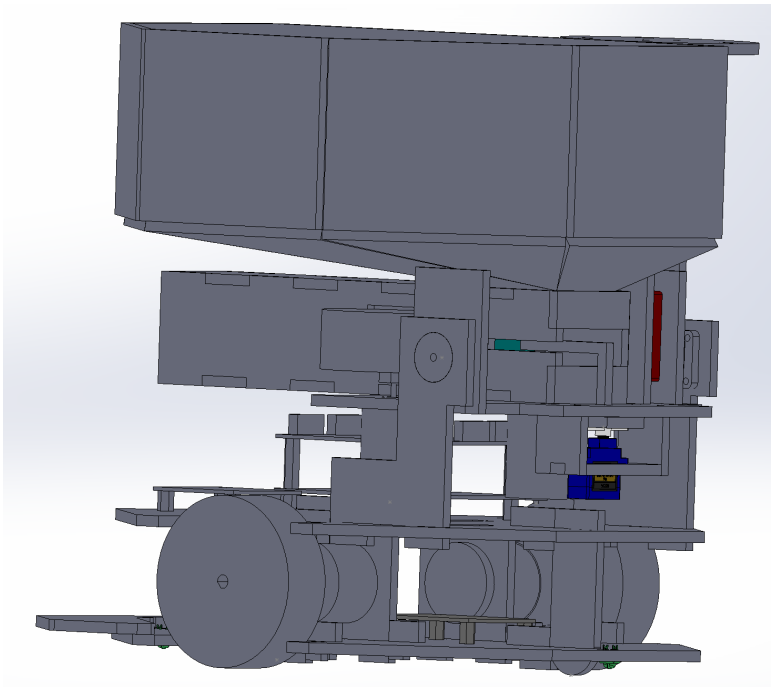


Figure 27: Final CAD Design - Rear Left

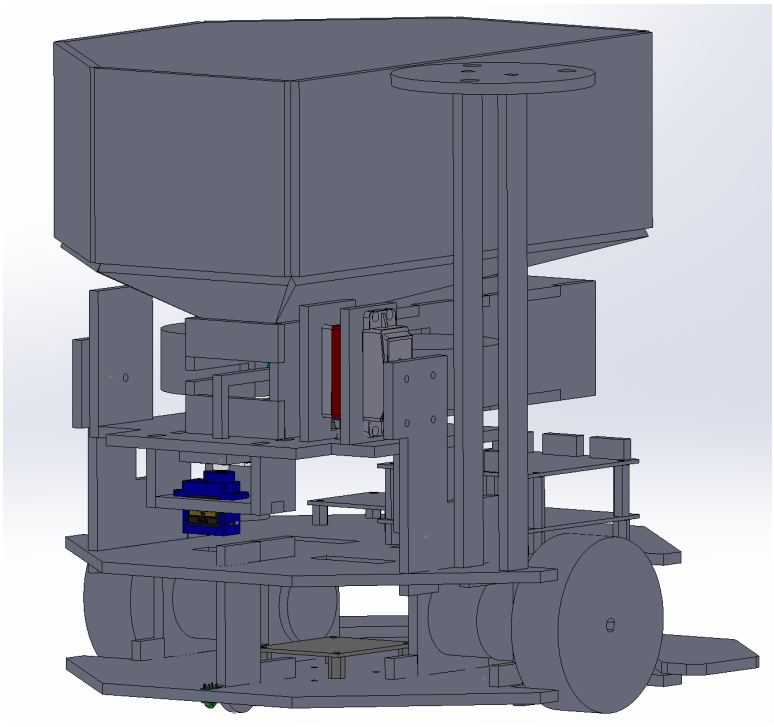


Figure 28: Final CAD Design - Rear Right

After completing all of the modeling in SolidWorks, all of the remaining parts were cut and assembled together. There were still a few items that need to be sorted out once the chassis was fully assembled. A few I/O boards used for the tape sensors, beacon detector, servos, and bumpers still needed to be placed physically.

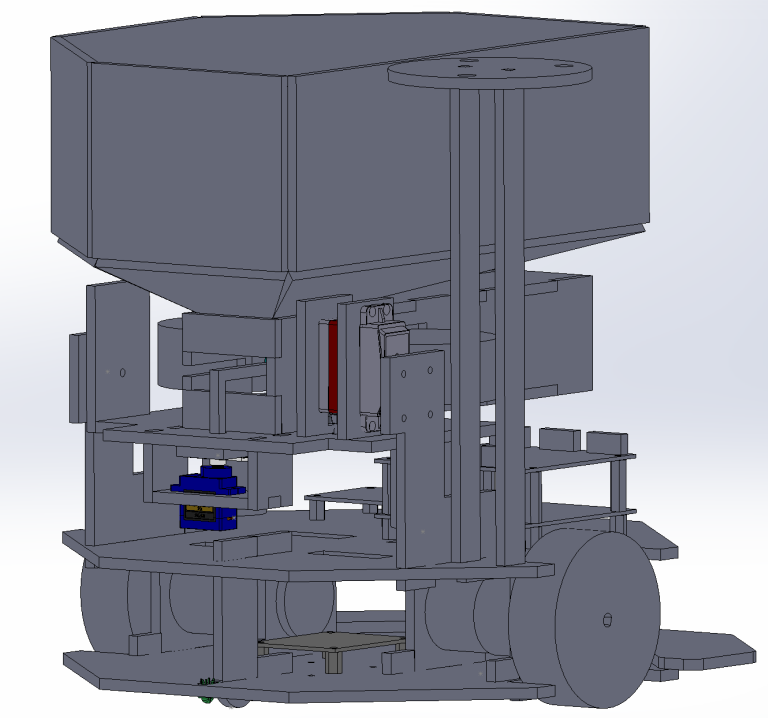


Figure 29: Final Design - Assembled

Fortunately all of the electronics fit in tight spots on the motorized platform. The circuit for the beacon detector was mounted on the hopper; since the beacon detector would be moving with the firing mechanism, it made the wiring tighter since it stay stationary relative to the firing platform.

2.5 Revisions\Modifications

A few modifications and revisions were made to M1N-5P3C as more testing was done on a completed field. These changes were for the most part performed as temporary band aids that happened to become permanent.

2.5.1 OBSTRUCTION!!!!

As M1N-5P3C approached beer checkoff a couple of red flags were made apparent. Since the hopper angled upwards with the firing platform, it was pointed out that the hopper was partially blocking the beacon.

Although this was only from one angle that would never be block during competition, the changes had to be implemented for a beer check off attempt. The modifications to the hopper were sloppily done, partially due to the fact that the team was well into being awake for over 24 hours and were rushing to make last minute changes.

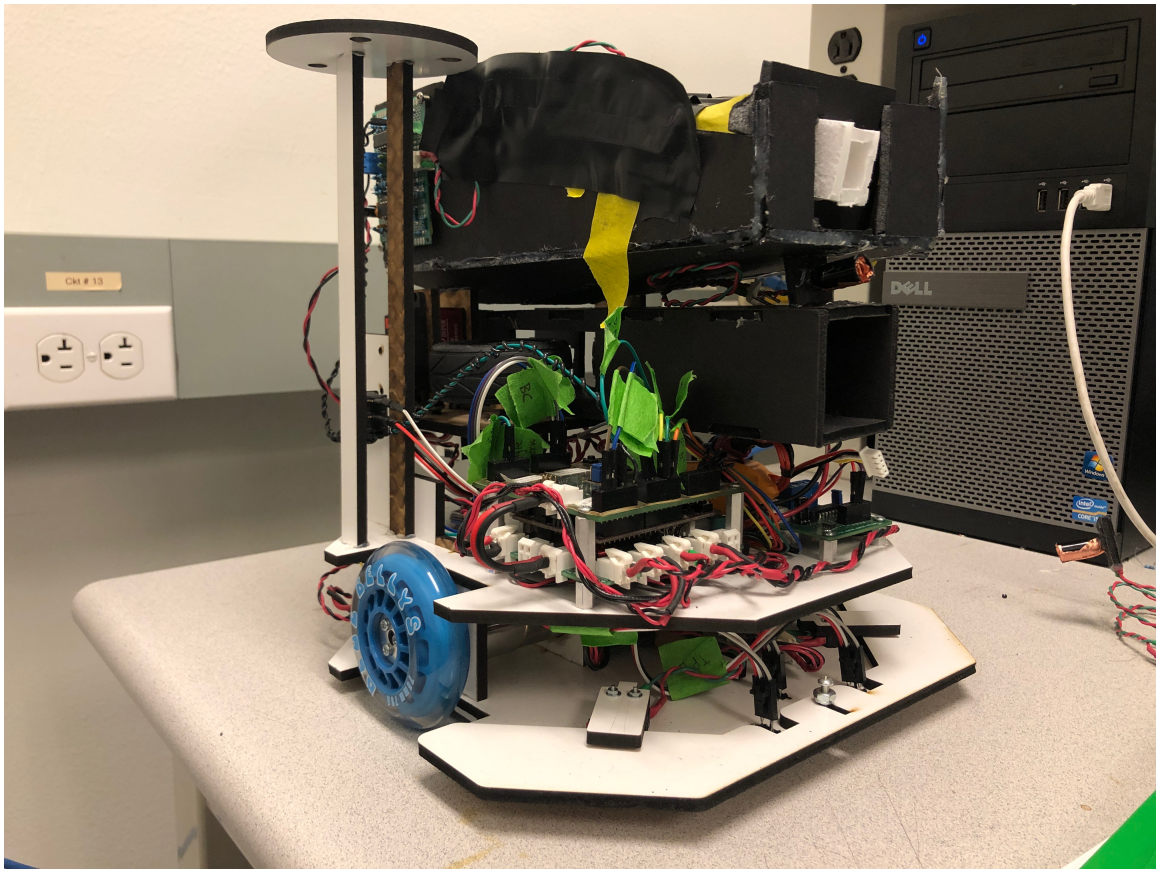


Figure 30: Revisions\Modifications - Hopper Chop

2.5.2 Beacon Hat

During the beer checkoff event, our bot was provided a beacon hat for the very first time. As soon as the beacon hat was placed on M1N-5P3C, problems became apparent. When approaching the IFZ, the reflection of the beacon hat bounced off of the white objects, which triggered a false positive. This lead M1N-5P3C to aim at the IR reflections instead of its intended target. To fix this issue a long narrow barrel was implemented to reduce the horizontal and vertical angles in which the IR could enter the detector. Creating the new barrel also caused some unforeseen issues; since the barrels were just pieces of white foamcore glued together, some of the IR from the beacon hat bled through and into the detector; this was remedied by wrapping the barrel with black tape[32]

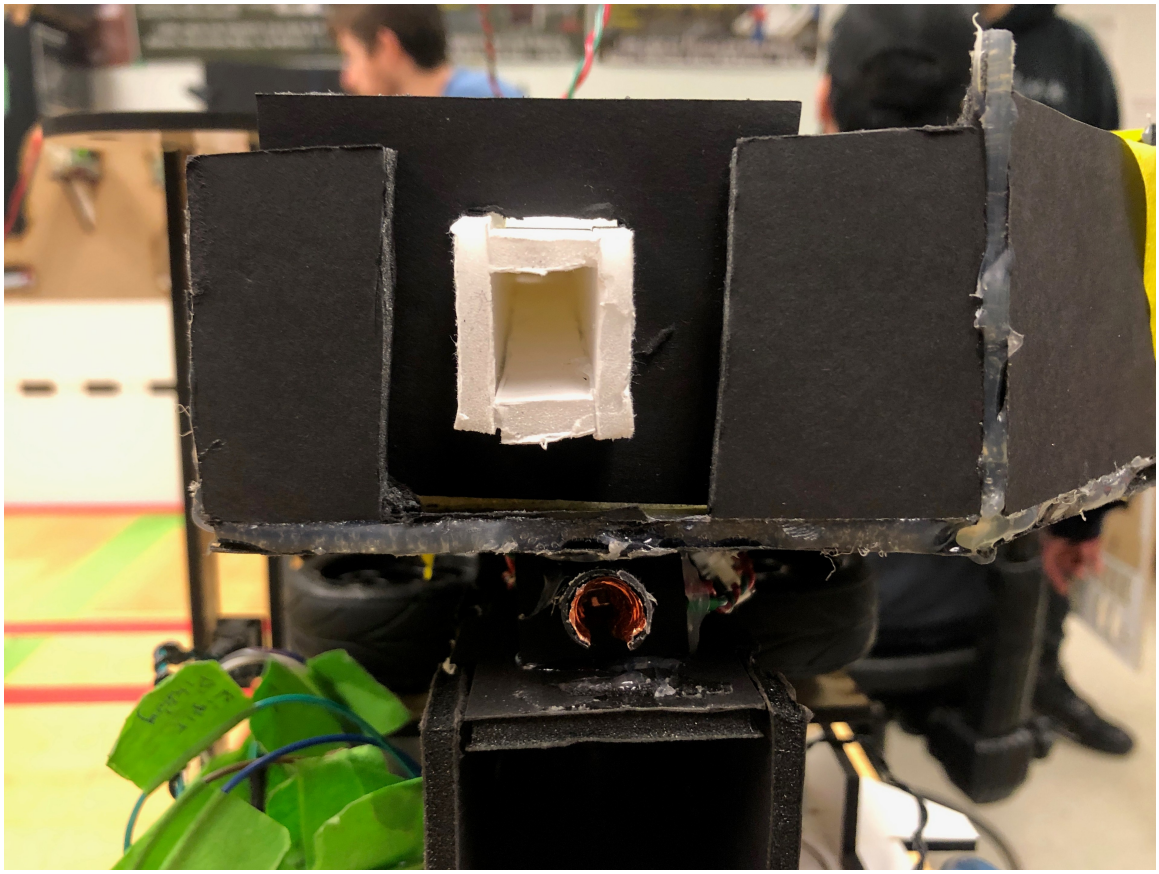


Figure 31: Revisions\Modifications - Beacon Barrel

2.5.3 Beacon Tracking

During a min spec check off, the enemy bot is guaranteed to be stationary, however in competition this is not the case. With a single beacon detector, it is impossible to know which the direction the enemy moves to and if an enemy is constantly moving, targeting could be a major issue. To combat this problem, two beacon detector were implemented. The main beacon detector is used for aligning the barrel to its opponent, while the second one detects a broad range on a single side to see if the enemy is there. This lets M1N-5P3C know whether to turn left of right based on whether a beacon is detected or not on the secondary detector.

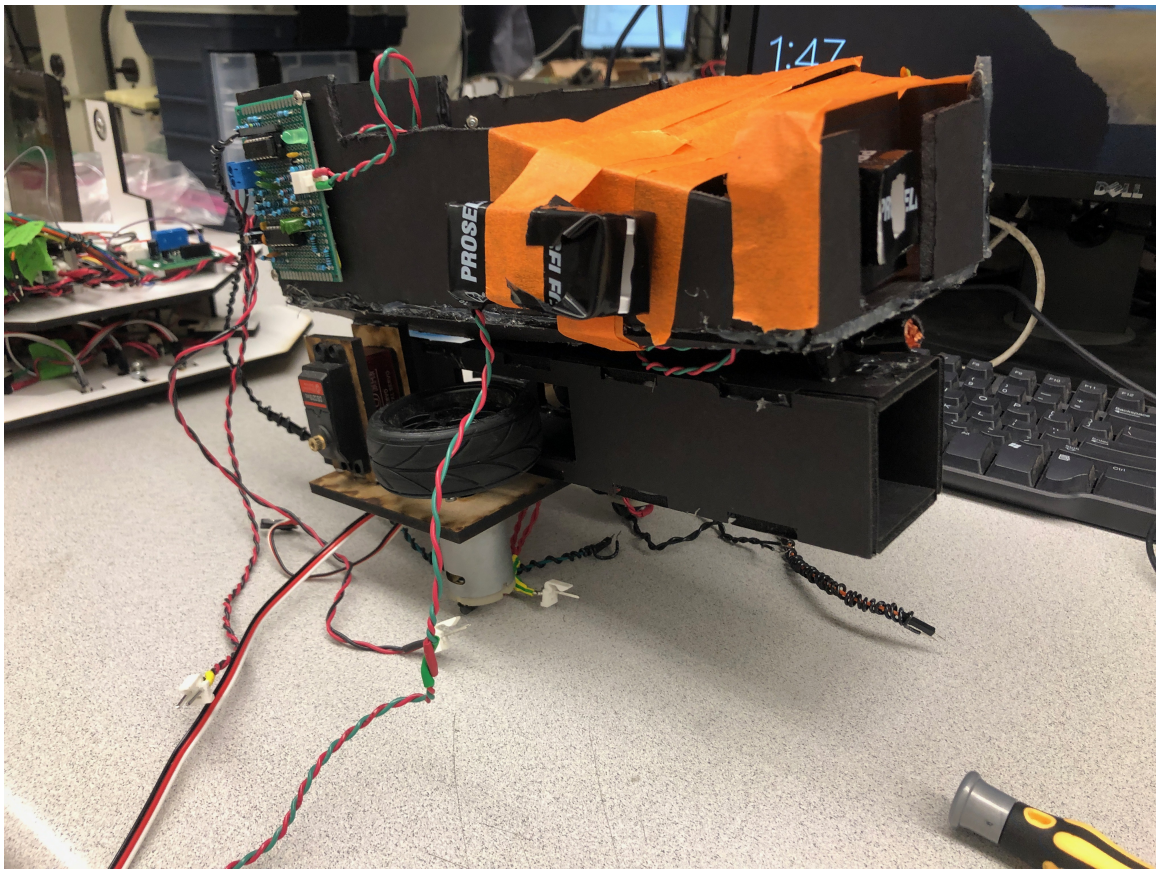
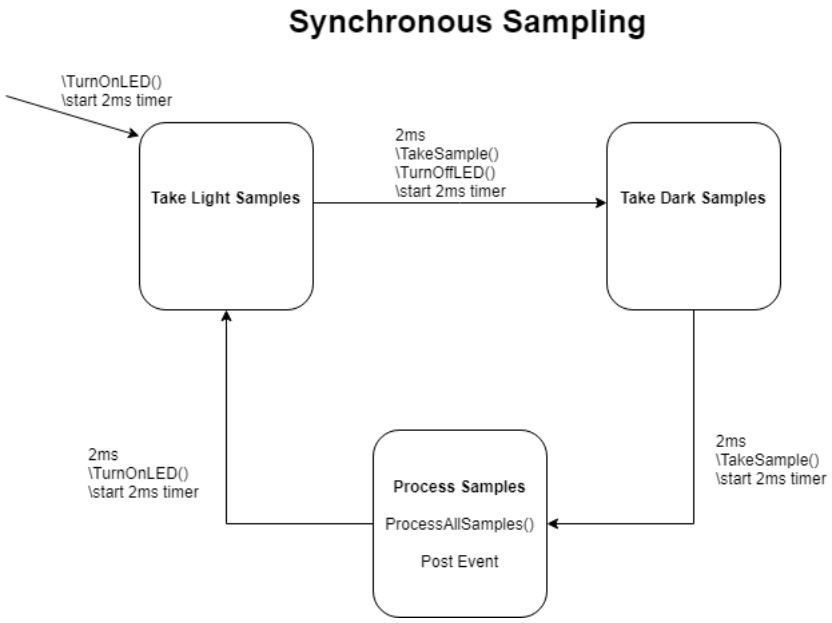


Figure 32: Revisions\Modifications - Beacon Tracking

3 Software Design

3.1 Synchronous Sampling

The software design started with writing a service to perform synchronous sampling for our tape sensors. The synchronous sampling makes our tape sensor more robust in different lighting scenarios so we are less likely to have a false tape event. The state machine used for synchronous sampling can be seen in the figure below.



Before transitioning to the initial state, we turn on all LED's and initialize the timer for 2ms that will be used to transition between state. After a timeout event occurs, we take samples for all tape sensors and store it in an array for light samples, turn off all LED, and initialize the 2ms timer before we transition to Take Dark Samples state. After a timeout event occurs, we take samples for all tape sensors and store it in an array for dark samples, and initialize the 2ms timer before we transition to Process Samples state. To keep track of the state of all four tape sensors, we used two static unsigned 8-bit variable to store them: one for the current tape states and another for the previous tape states. The bits representing the each tape sensor are as follows: bottom center(BC), top center(TC), top right(TR), and top left(TL) with TL being the least significant bit and BC the most significant bit. In Process Samples state, we take the dark samples and subtract them from the light samples. If the difference was less than the on tape threshold then the bit for that sensor is set high. Else if the difference is greater than the off tape threshold, the bit is set low for that sensor. In the case the difference is between the two thresholds, the bit for that sensor is set to its previous tape state. This is done for all tape sensors and stored in current tape sensor state. Once processing the samples is complete, we compare the current tape sensor states with the previous tape sensor state. If the current tape sensor state is greater than the previous tape sensor state, then we posted an On Tape event. Else if the current tape sensor state is less than the previous tape sensor state, then we posted an Off Tape event. After a timeout event occurs, we turn on all LED's and initialize the 2ms timer before we transition back to Take Light state. After implementing the synchronous sampling, we connected all the tape sensor to their proper AD pins and ran the simple service test to ensure that it works as expected. We printed the ad value for each tape sensor when it was on tape and off tape in order to get the proper threshold for on

tape and off tape events. After thoroughly testing each tape sensor we got the expected events signifying that our synchronous sampling works. The code for synchronous sampling can be seen in appendix A.

3.2 DC motors

We wrote the software that will drive the DC motor to move our robot. Using the roach motor function as reference, the function would take a number between -100 and 100 and anything out of that range would return an error. A negative number would result in reverse direction which is set using a digital pin and setting it low for the left motor and high for the right motor. We then used the PWM function to set the duty cycle based on the input of the function. The same thing was done for the other motor and we had a third function that could set different seeds for the left and right motors. We tested the motors by connecting them to an H-bridge and to the appropriate pins on the UNO 32. We wrote a test harness that would set the motor speed at different speeds ensure everything was working as expected and it was. The code for driving the motors and the test harness can be seen in appendix B.

The software for the pitching motors is very similar to the main DC motors driving our robot, with the exception that it only moved in one direction and we added motor offsets for each pitching motor. The reason for this was because when shooting, the balls tended to curve to the left. To fix this, we increased the right pitching motor speed so that it would curve less. After testing different offset values, we found that an offset of 0% for the left motor and an offset of 15% for the right motor worked best for our robot.

3.3 Servo

Using the RC servo header and source files provided to us, we programmed a servo for loading and shooting balls and a servo for adjusting the angle for our shooting mechanism. For the loading and launching servo, we created two functions: one for loading and the other for shooting. The loading function would just set the pulse time to the min pulse, which would allow a ball from the hopper to fall into the cannon and be set for shooting. Now we had to push the ball to the pitching wheels so it can be launched. We did this by setting the pulse time to max pulse, which moved the servo back its initial position.

For the angling servo, we created a function that would take in an angle and would adjust the servo according to the angle. This was done by multiplying the angle by 11.11, which is the angle to servo conversion factor, and subtracting it from the max pulse width.

3.4 Beacon Detector

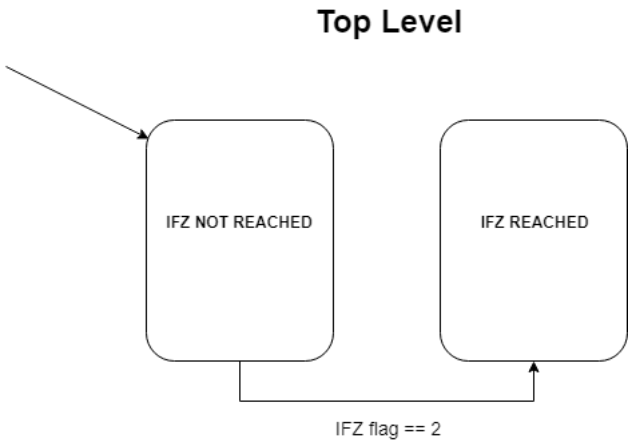
We wrote event checkers for our beacon detector. We read the digital pin input and if it's low then we incremented a beacon sum variable otherwise we decremented it. If the beacon sum variable is greater than the beacon detected threshold then post a beacon detected event. Else if the beacon sum variable is less than the no beacon detected threshold then post a no beacon detected event. We used the event checker test to verify that the appropriate events were being posted when triggered. The reason we check a beacon event this was is to add a delay posting the event so that our IR sensor for our beacon detector could line up with the target beacon thus giving us better aim. Getting the right beacon detected threshold required a lot of testing with different field configurations. The code for the beacon event checker can be seen in appendix C.

3.5 Debouncing Service

We had to implement a service for denouncing bumper and posting Bumped and Unbumped events. The service would call the bumper event checker every 5ms. To implement the bump detection we used two 8-bit numbers to store the history of each individual bumper. If a bumper was hit we left shifted the bits and added one to it. Otherwise, we just left shifted the bits. If the memory bit pattern matched the pattern 0111, then it is considered a valid bump and the appropriate bit in the debounced bumper state variable holding the actual event parameter was set high. Otherwise, if the memory bit pattern matched the pattern 1110, then it was considered a valid unbump and the appropriate bit in debounced bumper state holding the actual event parameter was set to zero. We then compared debounced bumper states to the previous bumper state value and if debounced bumper states was greater than previous bumper state, we posted a Bumped event. Else if debounced bumper states was less than previous bumper state, we posted an Unbumped event. We tested the bumpers using the simple service test to verify that the appropriate events were being posted.

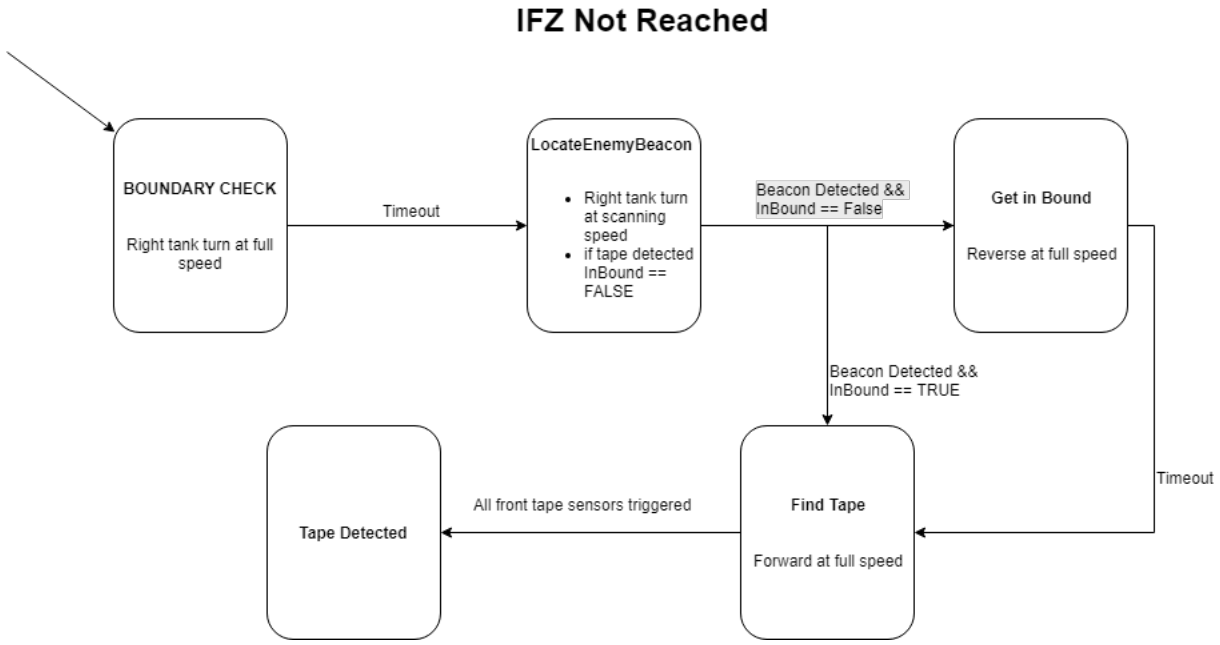
3.6 HSM

With all the sensors and motors functional we were ready to start implementing tape following and get to the IFZ. We used a HSM to complete the project task and have four levels. The top level state machine can be seen in the figure below.



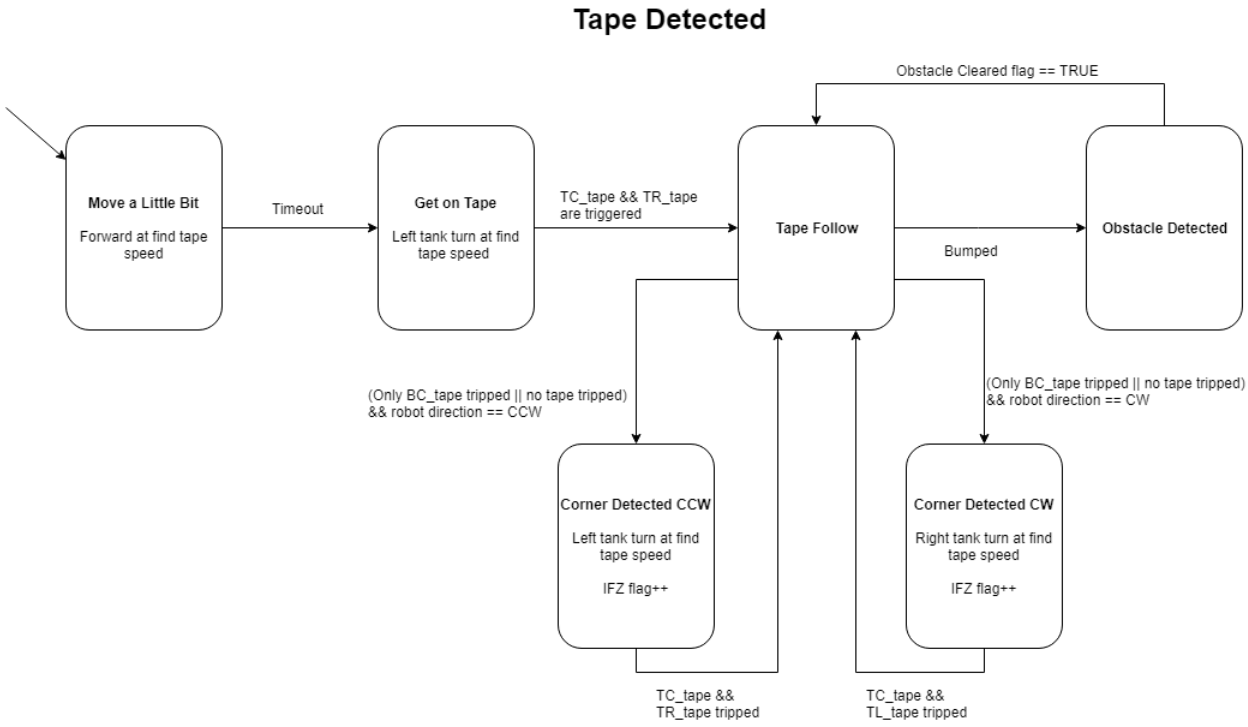
3.6.1 IFZ Not Reached

Within IFZ not reached we will implement getting on tape, tape following, and obstacle avoidance maneuvers that would help get us to the IFZ. Once we have turned on two corners, which we keep track of using a global variable, it signifies that we have reached the IFZ. The second level state machine for IFZ not reached can be seen in the figure below.

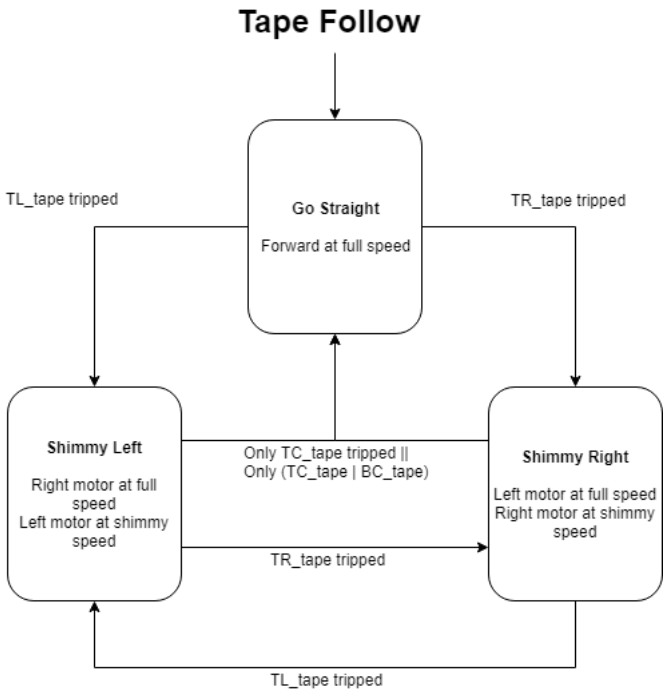


Since we will be placed in a random orientation and half our robot could be placed outside the tape boundary, we need to first check if we are inside the tape boundary. This is done by doing a 180°spin and setting the inbound flag false if tape is detected. Once we know whether we are inbound or not, we use the enemy beacon to orient ourselves so that we face the tape. Once a beacon is detected and we are not inbound, we will reverse for half a second and drive forward until all front tape sensors detect tape Otherwise,

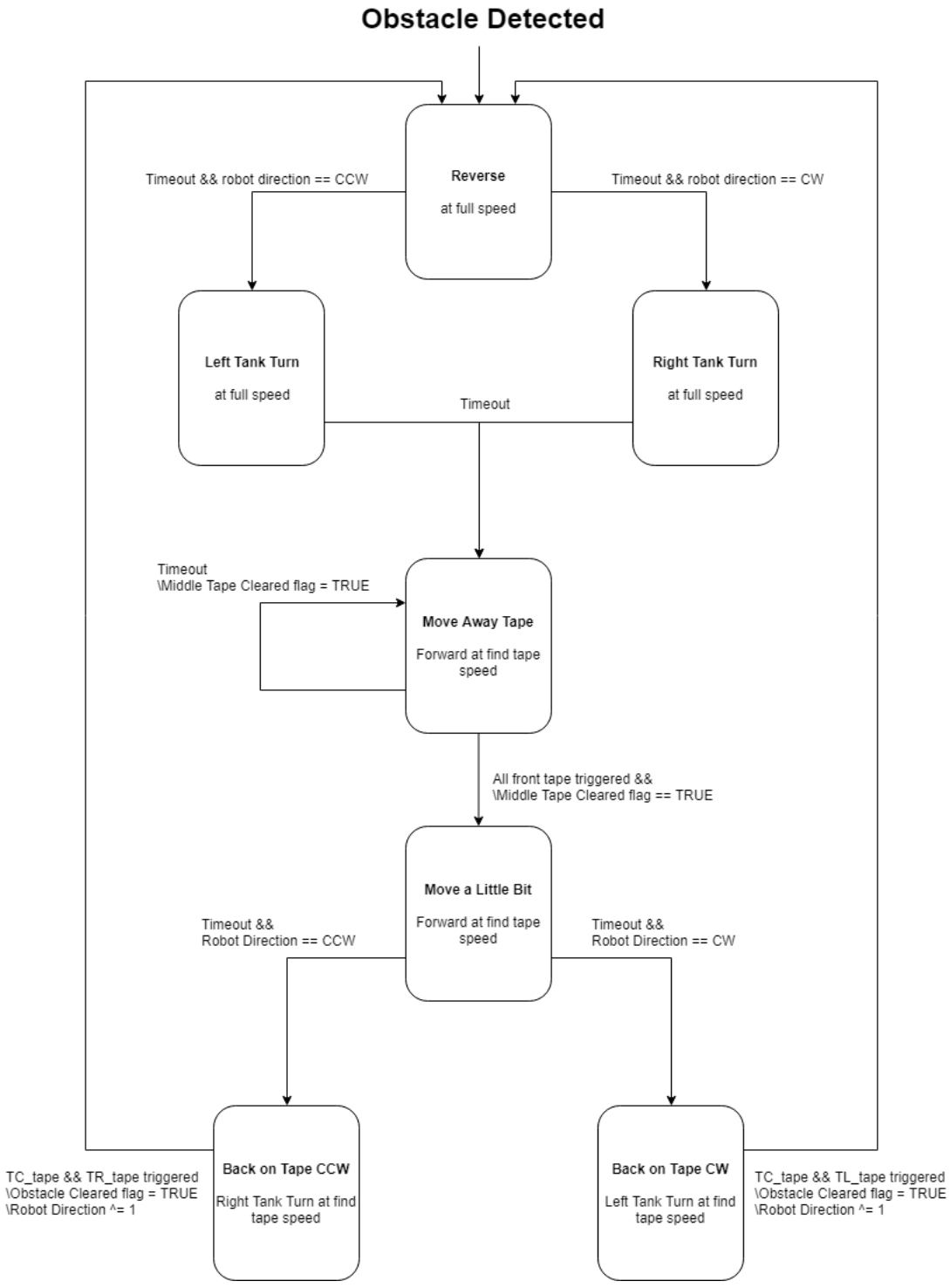
we just drive forward until all front tape sensors detect tape. Once we get into tape detected, we will be at the third level of the HSM and the state machine for tape detected is shown in the figure below.



Once we detected the tape, the robot keeps moving for 200ms, which will place the middle of the robot approximately on the tape. The robot will then do a left tank turn to get on the tape and start tape following. The tape following state machine can be seen in the figure below.



To ensure that our robot goes straight on the tape, we have it adjust itself by either moving a little to the left when TL tape sensor is tripped or a little to the right when TR is tripped. Only when the TC and BC tape sensors are tripped will the robot move straight. If we detect an obstacle while tape following, then we have to maneuver around the obstacle. The state machine for obstacle detected can be seen in the figure below.



The first thing we do when we detect an obstacle with our bumpers is reverse for 250ms to get away from the obstacle. The time is enough to cover our worse case scenario where the obstacle barley hits the edge of our robot. We want our robot to go to the other side of the field and start tape following there. To do this, our robot performs either an approximate left or right 90°tank turn depending on whether our robot

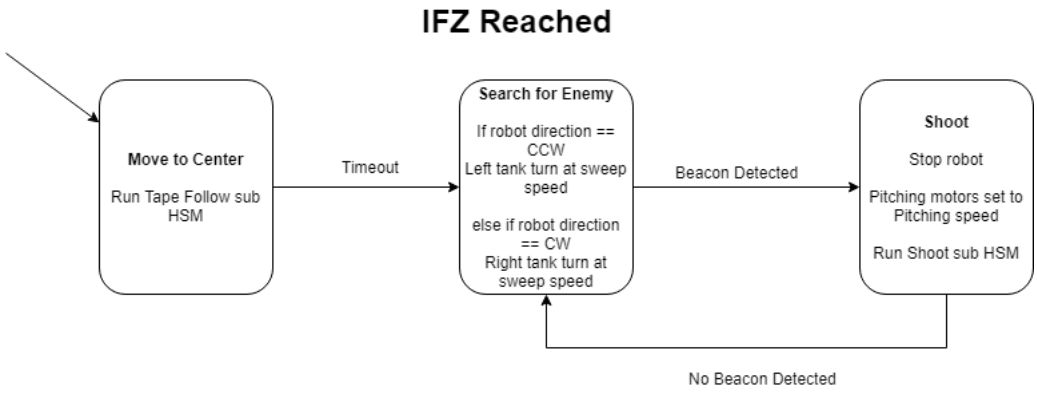
is going clockwise(CW) or counter-clockwise(CCW) on the field. We know we get to the other side of the field when all front tape sensors are tripped and 2.5s have passed. The reason for the 2.5s timer is to ignore any middle tape we will encounter. Once we have gotten back on the tape we set the obstacle cleared flag true and toggle the robot direction. After this we begin to tape follow again.

While tape following, if we detect a corner then our robot will do a left tank turn or right tank turn depending whether the robot direction is CCW or CW. We increment an IZF flag after doing a corner turn. Once our robot has turned on two corners, it signifies that our robot has reached the IFZ.

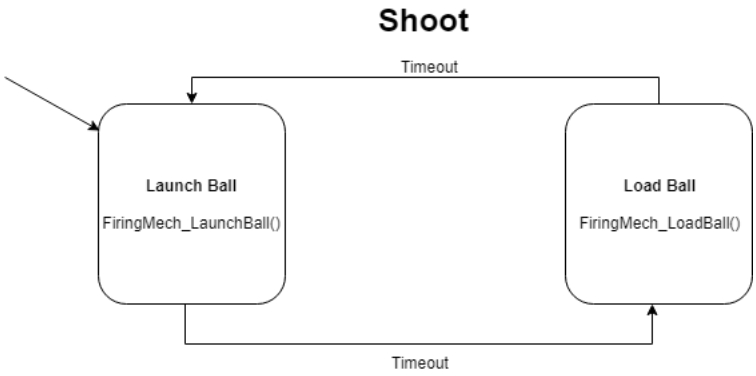
A problem we encountered when testing was sometimes our robot would think it detected a corner when tape following even though it shouldn't have. This was a problem since we relied on counting corners to determine when we have reached the IFZ. After thorough testing using tattletale and printing ad values of all tape sensors, we discovered that the white printing on the tape was enough to sometimes trigger an on tape event, which is what caused the false corner detection. After increasing the on tape threshold we didn't have false corner detection anymore.

3.6.2 IFZ Reached

Since we do not plan on moving back to the front of the field, we will just search for beacon and shoot until we get the required two hit for min spec. The state machine for IFZ reached can be seen in the figure below.



The first thing we do when we get to the IFZ is move towards the center using a 1150ms timer. Once there the robot rotates at the slowest speed possible until it detects a beacon. Once the beacon is detected, we start the pitching motors and began loading and launching balls using a servo once the pitching wheels have reached their assigned speed. The shooting state machine can be seen in the figure below.



In shoot state machine all we do is add a 0.5s delay between launching and loading the ball to the pitching motors. We do this in order to give the servo motors time perform the load and launch actions. The robot will continue to shoot until there is no beacon detected and will begin to search for the beacon again.

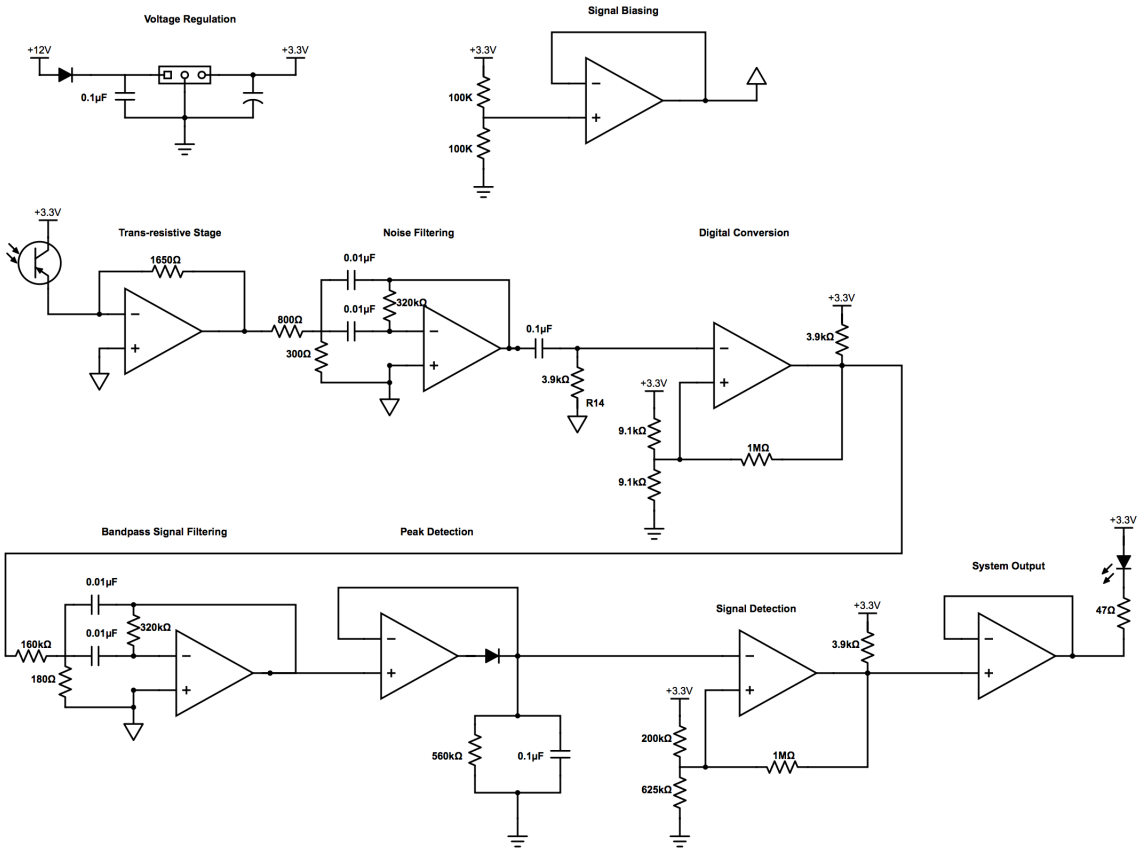
4 Electrical Design

The electrical design includes sensor designs and motors calculations. The sensors used on the robots includes photo transistors at the front with beacon detector circuits and tape sensors in the bottom.

4.1 Beacon Detector Design

The Beacon Detector Design on this project was based on the requirements of the project. The obstacle in the middle of the field had two "mini" Beacons with 1.5k and 2.5k Hz, and there was a 2.0k Hz Beacon located on the top of the enemy target. Thus, the Beacon Detector needed to block the signal with frequency of 1.5k Hz and 2.5k Hz and output the signal around 2.0k Hz whenever it detected any signal. The signal from the Beacon was generated by a set of LEDs. The Beacon Detector used an photo transistor for detecting signal.

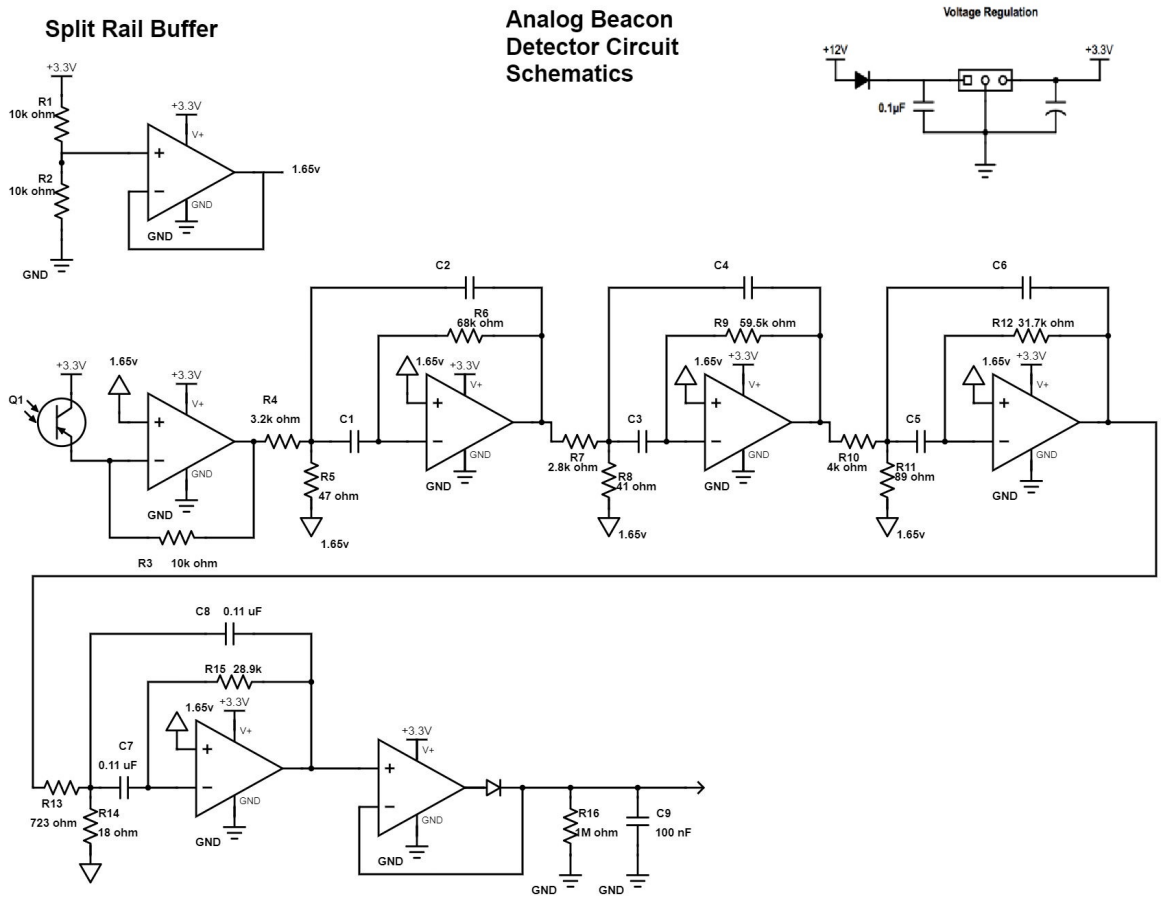
There are two Beacon Detectors on the robot. One Beacon Detector is a digital Beacon Detector and the other one is an analog Beacon Detector. For the minimum specification, the robot only used the digital Beacon detector. The schematics of the digital Beacon Detector is shown below.



The digital Beacon Detector contained eight stages in total. Since the voltage output from the power distribution board was 9.9v, a 3.3v regulator was needed to convert the voltage from 9.9v to 3.3v after the power and ground connectors. The Beacon Detector stages started with a transresistive amplifier which amplifies the signal received by the photo-transistor. After the transresistive amplifier, the signal went into a band-pass filter which blocked the noise signal except the signal from 1.5k to 2.5k Hz. The comparator after the band-pass filter was used to convert input signal with the peak to peaks voltage less than 3.3v to 3.3v. The second band-pass filter on the schematics was used to attenuate the signal of 1.5k Hz and 2.5k Hz. The center frequency of the band-pass filter was 2.0k Hz, thus, the signal of 2.0k Hz could be larger than 1.5k Hz or 2.5Hz. Thus, the output of the peak detector was different for different input signal frequency, since they had different charge time for different voltage level. With the comparator and LED buffer in the end, the output of the Beacon Detected could change between 0 volts with no 2.0k Hz signal detected and 3.3 volts with 2.0k Hz signal detected. The Beacon Detector was able to detect the Beacon for more than 18 feeds on the robot.

In order to increase the accurate of the aiming system and make the robot follows the enemy target while it was moving, there was an analog Beacon Detector on the robot. This Analog Beacon Detector was only

designed for the competition, so, it was not used during the minimum specification check off. The schematics of the Analog Beacon Detector is shown below.



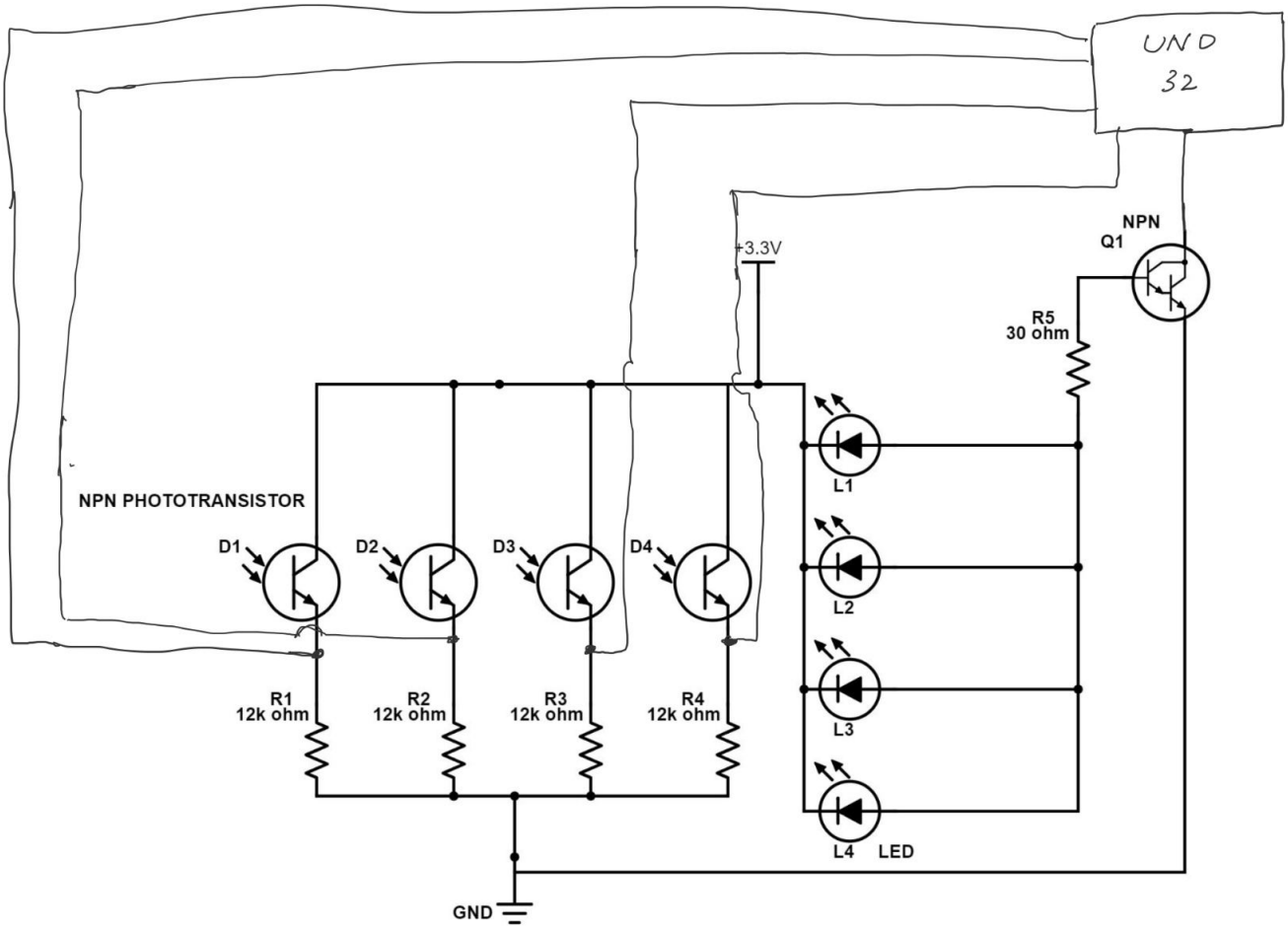
The Analog Beacon Detector contains seven stages. In order to get more attenuation for the 1.5k Hz and 2.5k Hz signal, this analog Beacon Detector used an 8th order band-pass filter. The 8-the order band-pass filter has four stages, and each stage has the Q-factor of 20. Thus, the signal with 1.5k Hz or 2.5k Hz were almost blocked in any distance after the filter. By the filter above, the output of the peak detector only contained the signal with 2.0k Hz. Since there was no comparator existing before the filter, the signal strength was different at the different positions. Thus, the robot could be controlled to aim at the enemy target based on the analog signal reading.

4.2 Tape Sensor Circuit Design

The Tape Sensors used in this project were Reflective Optical Sensor with Transistor Output sensors. The sensor contained with one LED and one transistor. The transistor is able to receive the reflected lights which was from the LED. Since the output voltage value of the transistor was different on different surfaces, the

UNO32 board was able to get readings from the output of the transistor for determining the color of the reflecting surface.

The robot used 4 tape sensors in total. To make the tape sensors be easier to organize, all 4 tape sensors' circuit were soldered on single one purf-board. The schematics for the tape sensors can be seen in the figure below.



The LED and transistor for each tape sensor were connected separately, because the tape sensor needed to be turned on and off by the software for Synchronous Sampling (see software part for details). All the transistors were connected to the 3.3 volts power source and ground in parallel and connected with 12k ohm resistor in series.

4.3 5 Volts and 3.3 Volts Power Board

5 Conclusion

Through cooperation, hard-work, and time commitments, we were successfully able to complete the project task within five weeks and stay under the $150, with our BOM being $74.46. This was the most engaging school project we have ever done to date. This project was unique in that we actually applied almost everything we learned in class to help solve an open ended question in designing an autonomous bot that performs a specified task.

6 Appendix