Slugnificent Seven Project Report

UCSC Spring 2019, CMPE 118 - Introduction to Mechatronics

Team 3:George Leece, Ali Elagamawi, Joaquin Banuelos

June 7, 2019

Table of Contents

Introduction

Rules

The general idea of the final project was to make an 11 x 11 x 11 droid that would be capable of starting in a designated area on the field, navigate to the other side of the field without colliding with obstacles or able to resolve collisions within 5 seconds and then able to shoot at an enemy on the opposite field. In order to pass minimum specifications, the droid must be able to complete the above and be able to locate the beacon of the enemy and hit it at least twice or once in the head which is the can resting on top of the beacon. Specific rules are listed below, gathered from the document provided by the class.

Field Specifications

All of the field specifications are illustrated through Figure 1.

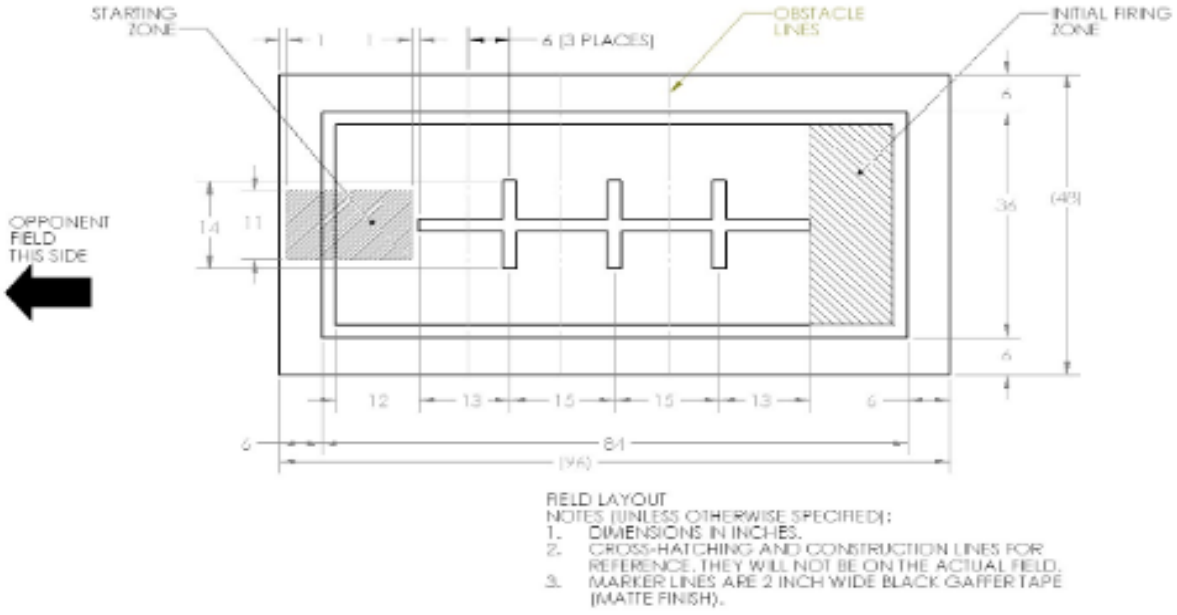


Figure 1: Field Specifications

Competition Goal

The task is to build a small autonomous robot, or droid, that can effectively and robustly navigate a standardized field, locate the enemy and advance through the field. The bot may advance forwards from the IFZ by hiding behind obstacles for a specified amount of time, and then shooting their opponent with ping pong ball ammunition. The match is won by shooting the opposing bot's can on their Beacon which is a headshot, or through points for hitting the body of the robot. Two shots on the body equals a win. This will be accomplished in teams of three, over five weeks, during which teams will design, implement, test, and complete a working bot. The minimum specification task will be completed when three trials are ran and two are completed successfully.

Initial Firing Zone (IFZ)

The initial firing zone is designated by the rearmost horizontal strip of black tape and extending 6in beyond this strip to either side. Your robot must be fully inside their initial firing zone before firing upon the opposing robot. The bot can leave the initial firing zone and advance forwards, by hiding behind obstacles for a second then maneuvering to shoot at the opponent robot. However, the bot can remain in the initial firing zone the entirety of the round and fire from the IFZ, but the IFZ is the farthest point from the enemy robot. There is no component placed in the IFZ to indicate the bot has reached the IFZ. All teams need to deduce and code a way for the bot to know when it has reached the IFZ.

Advancing Down the Field

In order to advance to a closer fighting position, your bot must first hide behind the next closest obstacle for a minimum of one second, and then can advance to the next obstacle or fire at the enemy. Hiding is defined by stopping behind an obstacle, with the robot fully behind at a complete standstill, before moving on. The bot is not allowed to stay behind an obstacle for too long or will be disqualified. Collisions or contact, with the obstacle must be resolved within 5 seconds. Failure to do so means the robot is disqualified.

Obstacle Line

Each obstacle line is 6 inches below the corresponding horizontal black tape, such that if your bot is tracking the line in the center it will pass within 1/2 of an inch of the obstacle, assuming that the robot can track the line, and it's the 11in wide dimension.

Robot Size

Each droid must start the match within 11”x11”x11” volume. Parts may move after the round begins, but must remain intact throughout the match as in the bot may not drop or lose any pieces or components, excluding the ammunition. Robot sizing is checked by the Cube of Compliance. If the robot does not meet the Cube of Compliance, it needs to be altered to fit the Cube of Compliance prior to the competition or check off.

Requirements

The robot is required to stay within the field, marked by 2 inches of black tape boundaries. In-bounds is defined as keeping, at least half the robot being within the black tape. Robots exiting the playing field by having more than half the robot pass the black tape boundary are disqualified. The bot is required to detect collisions and resolve collisions. The bot is required to break contact within 5 seconds of the collision or will be disqualified. The bot must be able to reach the IFZ and destroy the opposing bot within the 2 minute time limit.

Tower Specifications/Cactus Specifications

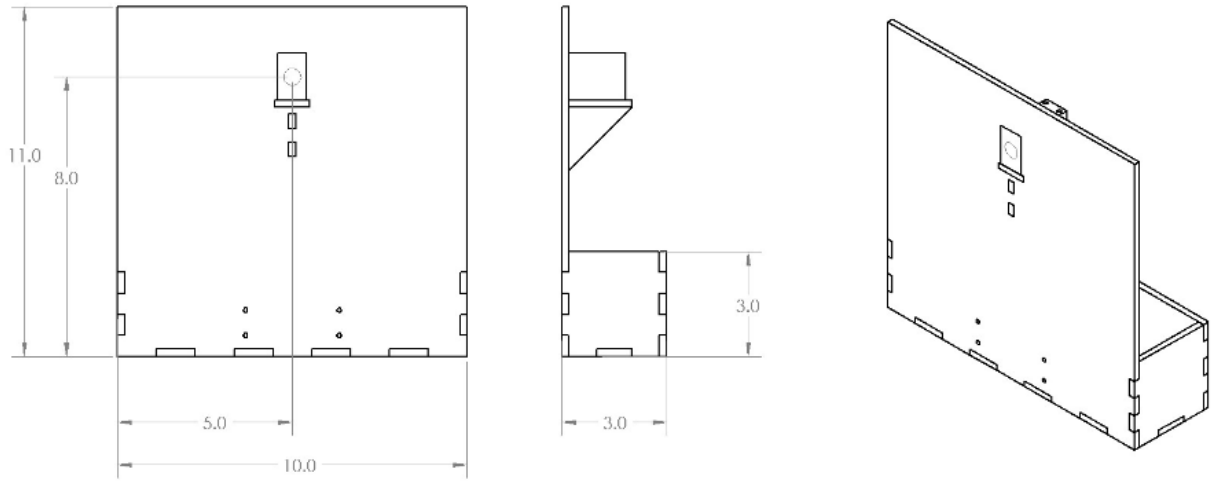


Figure 2 Tower Specifications

Overview

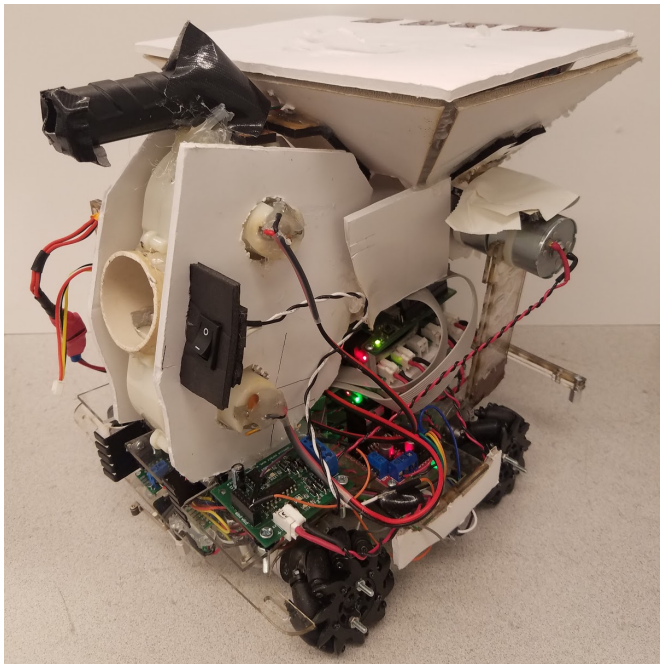


Figure 3: Plankton Bot

The final design of the Team 3 Mecanum Wheels droid, Plankton, consists of a motor base with four Motors each supporting one Macnum wheel, an angled ping pong ball launcher, similar to a Hot Wheels launcher or a pitching machine, and a hopper ammunition reserve. All components used for this bot were either milled or PCB. The bot was able to complete the minimum specification check off in under a minute, at 57 seconds and 37 seconds. The bot had several unique features such as the 4 wheel drive of mecanum wheels, the vertical shooting mechanism, and an on/off switch Top Level State machine.

Droid Specifications

The droid must be a standalone and any that fits in a 11” x 11” x 11” cube at the beginning of the round. The machine must contain all ping pong ball ammo at the beginning of the round, nor is loading allowed. It must be able to detect bumps at a

height of 3 inches above the ground. The bot must be able to resolve collisions and break contact within 5 seconds to avoid detection. The bot must also remain in bounds defined by the perimeter of 2in black tape on the field. In bounds is considered that at least half of the bot is on the inside of the perimeter.
Droids are programmed in C, with the standard MPLAB-X IDE. Thedroid behavior will be constructed using the ES_Framework from Lab 0. The droid may be reprogrammed between rounds if desired, but may not be altered once the field configuration is established.
Each droid will be equipped with a remote power switch, using the remote switch header on the UNO stack. At the beginning of the round, the bot will be switched on, and may not be interfered with it until the round ends.

Materials

Each team will be provided with one UNO stack, one H-Bridge, one stepper board, if needed, one DS3658 Board, one battery and one ULN2003. There is also wire, regulators, and solder freely available in the lab.
Each team should not exceed a budget of $150 total for other parts of the robot, and must contain an up-to-date bill of materials. If a nice 5000 component is spotted on your board it will be confiscated and $150 returned.

Example Field

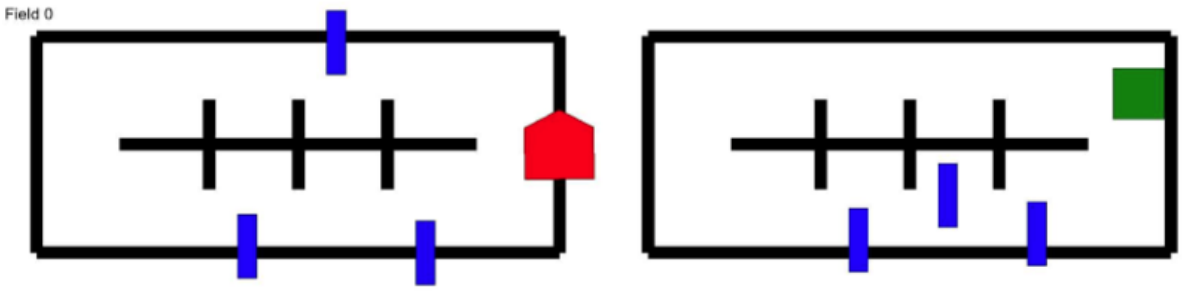


Figure 4: Competition Spec Robot
We were also provided a random field generator that lets us see edges cases more easily.

Design

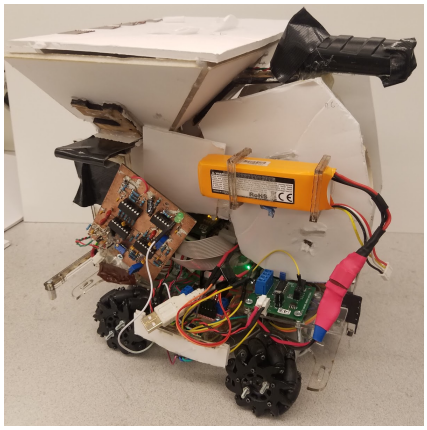


Figure 5: Plankton Front Left

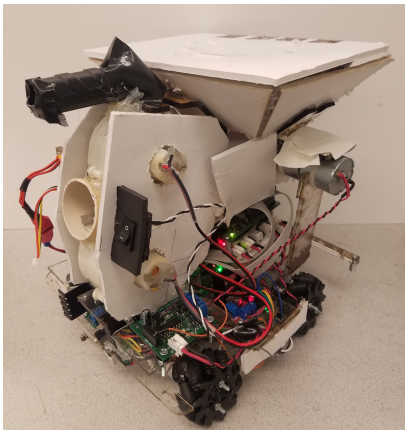


Figure 6: Plankton Front Right

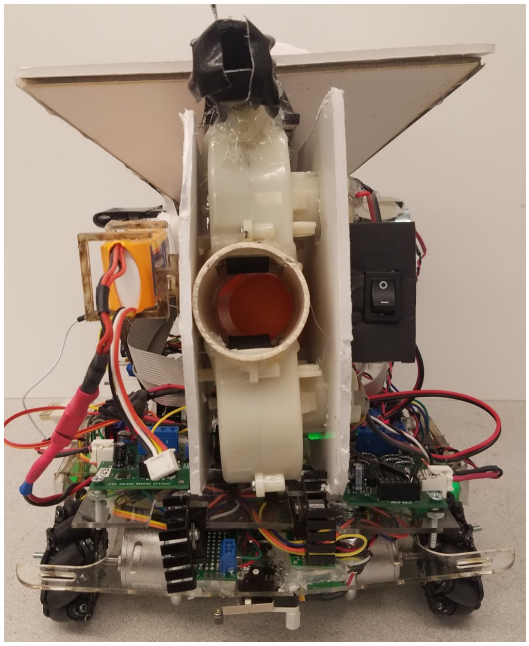


Figure 7: Plankton Front

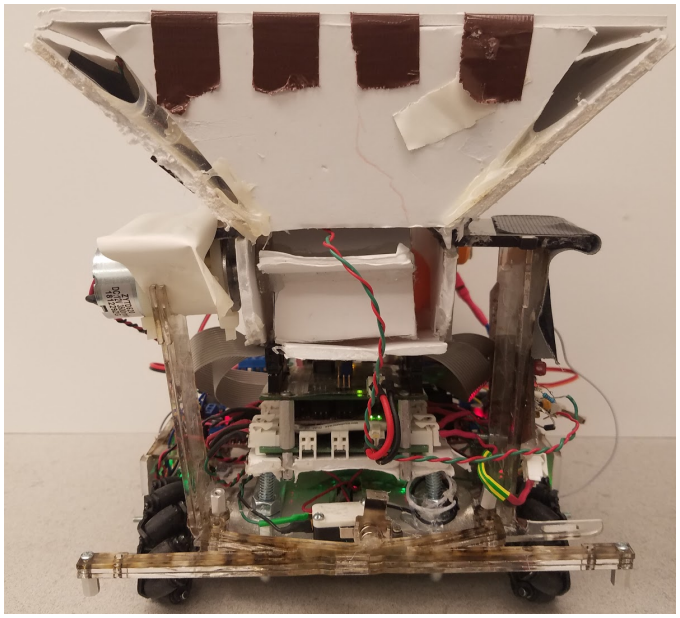


Figure 8: Plankton Rear

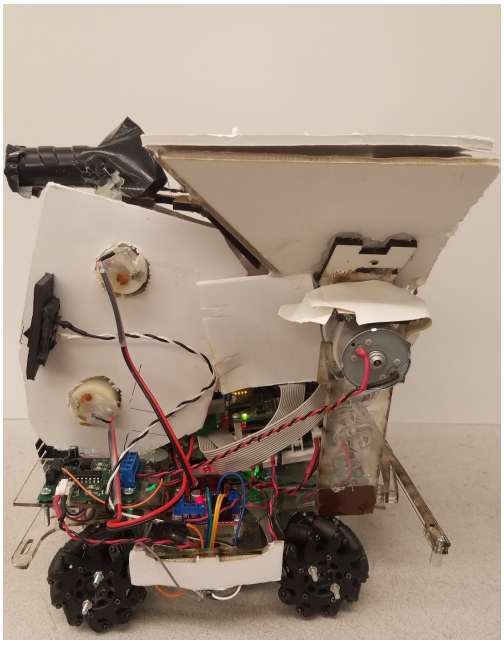


Figure 9: Plankton Right

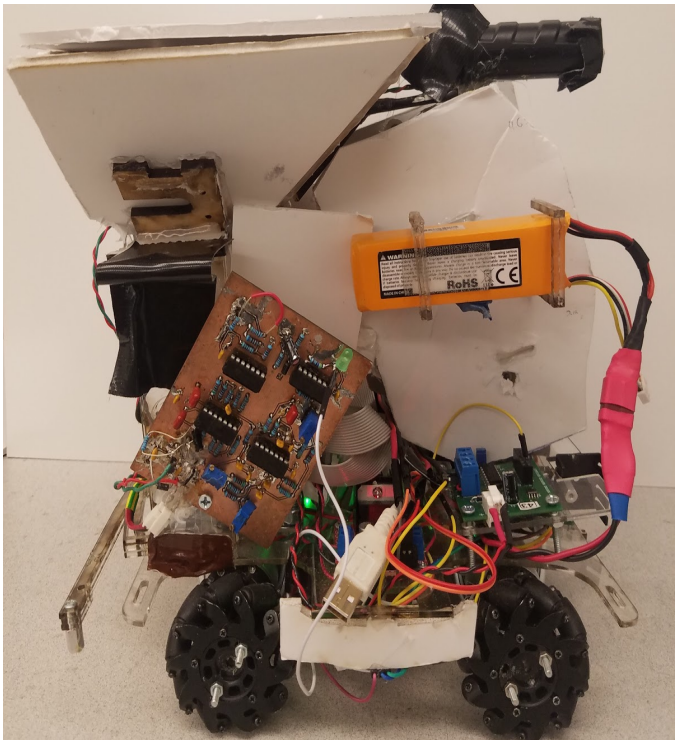


Figure 10: Plankton Left

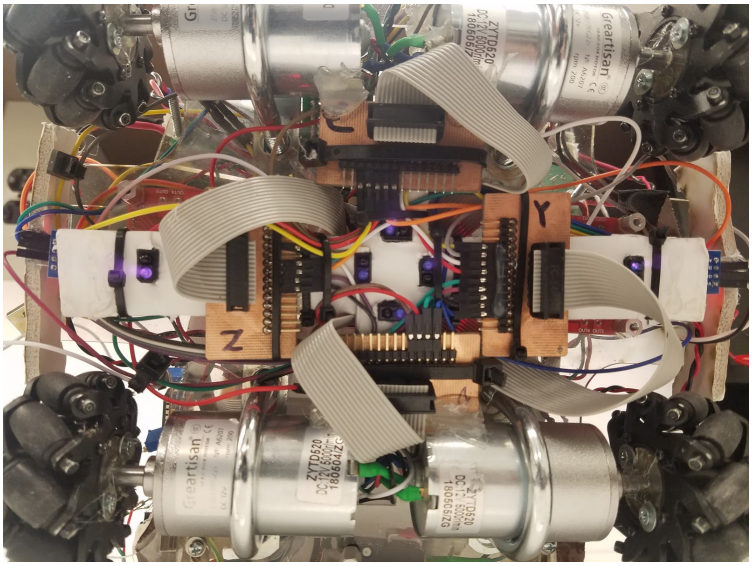


Figure 11: Underside of Plankton

Component Choice:

An overview of all possible components were analyzed when brainstorming the concept and implementation of the bot. Initially, a design review of all possible components were compared with the potential pros and cons for performance of the bot. A combination of these components led to the final design of the bot, which were the beacon detector, 6 tape sensors, 4 mecanum wheels, and a vertical shooting mechanism with a Hopper design as its loading mechanism.

Mecanum wheels were chosen for their ability to traverse across the field in a multitude of directions without changing the orientation of the bot. Even after being advised against utilizing the unusual drive, the drive design was accomplished within the first week to ensure the capabilities of the mecanum wheels. Along with the success with the mecanum wheels came the base design. Originally MDF was utilized as the prototype, which proved wise as it allowed much customization and remodeling through drills, saws, and sanding in order to create a more proper final acrylic final.

The driving motors that we chose were the Greartisan DC 12V 200RPM Gear Motor High Torque Electric Micro Speed Reduction Geared Motor Eccentric Output Shaft 37mm Diameter Gearbox. The reasoning behind the choice of these motors were that we had to have a high RPM to decrease the effects of having non similar amperage pulls on the different wheels within the 4 wheels system that we created. The second reason was due to its high torque,which decrease the chances of coasting (extra unanticipated strafing due to the omni directional design, which can lead to undesired movement). Finally, was the price; as we are working under a 150$ budget, we knew that this would require most of that budget due to requiring four motors for the four different wheels, so the price was a big factor that we had to take into account.

The tape sensors were purchased in order to provide more time with other parts of the design and they also provided good feedback with an LED indicating power and an LED indicating the digital output. The tape sensors also provided an option for digital or analog output allowing a multitude of options of utilization. In the end, with the layout selected for the tape sensors, the digital output was utilized. The final layout of the tape sensors consisted of four in a square orientation in the center of the bot, with two on the “X” extremities. The X configuration consisted of the Far Right, Right, Left, and Far Left.

The Y configuration consisted of the Center and Rear Tape Sensors. Combinations of these tape sensors indicated different events such as ON/OFF_X/YTAPE events, ON/OFF_Y/X_CORNER events and more.

The beacon detector utilized was from lab 2 where the strongest beacon detector of partners would be soldered to a perf board. However, a previous partner had decided to mill a PCB thus creating a cleaner and better board to utilize for the beacon detector. This board was chosen due to its sturdiness, cleanliness, and strong detection capability. However, later the transresistive resistor was swapped out for a larger value in order to increase the gain in that stage. Another issue noticed was the original photodiode and its placement not being on the peripheral of the board. The diode was replaced with a round diode connected to wiring separating it from the board. While this diode picked up more signals all around, this was useful as the diode was placed in a “cannon”.

The firing mechanism utilized consisted of two motors placed vertically, the top one running on a 5V power supply and the bottom with a 3.3V power supply. This combination provided the necessary suction to bring balls into the barrel while also providing the arc and backspin to hit the target. The rubber wheels allowed for more traction in order to get more grip on the balls while still allowing them to pass through the barrel and not lose too much velocity.

The loading mechanism consisted of a hopper large enough to accommodate around 18 balls and a motor with a water wheel design that would fluctuate between forward and rear motions in order to prevent jams from occurring with balls prior to reaching the barrel. An H-bridge was utilized along with the loading mechanism in order to allow for the change in directions along with a PWM cycle being sent. However in the end, the duty cycle was not necessary as the maximum was sent to the H-Bridge.

Beacon Detector

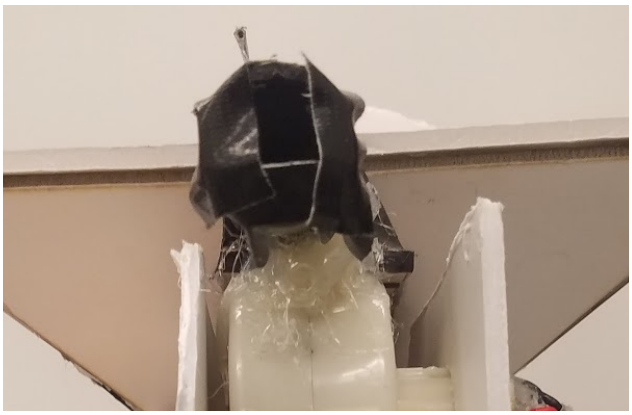


Figure 12: Mechanical Shielding of Photodiode

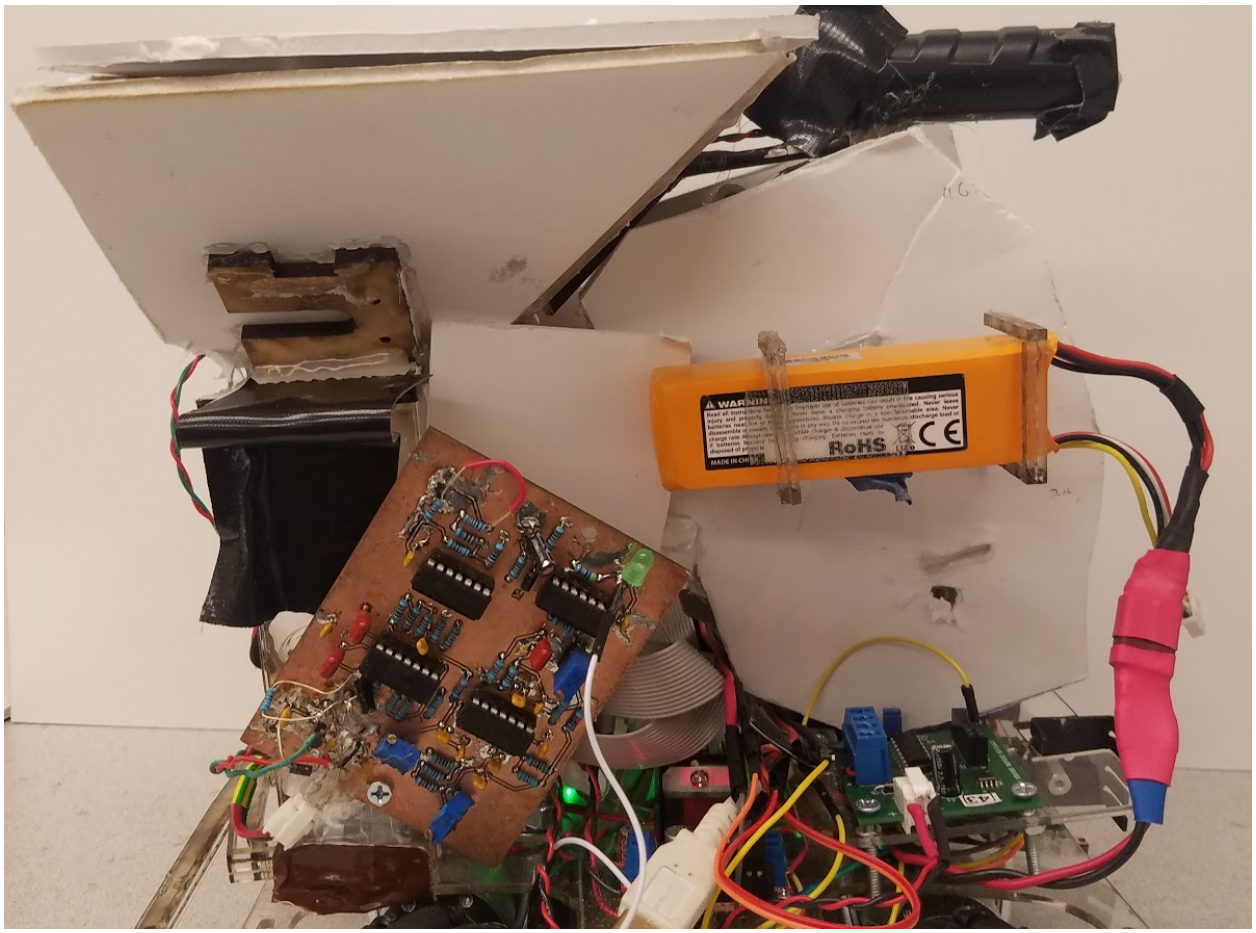
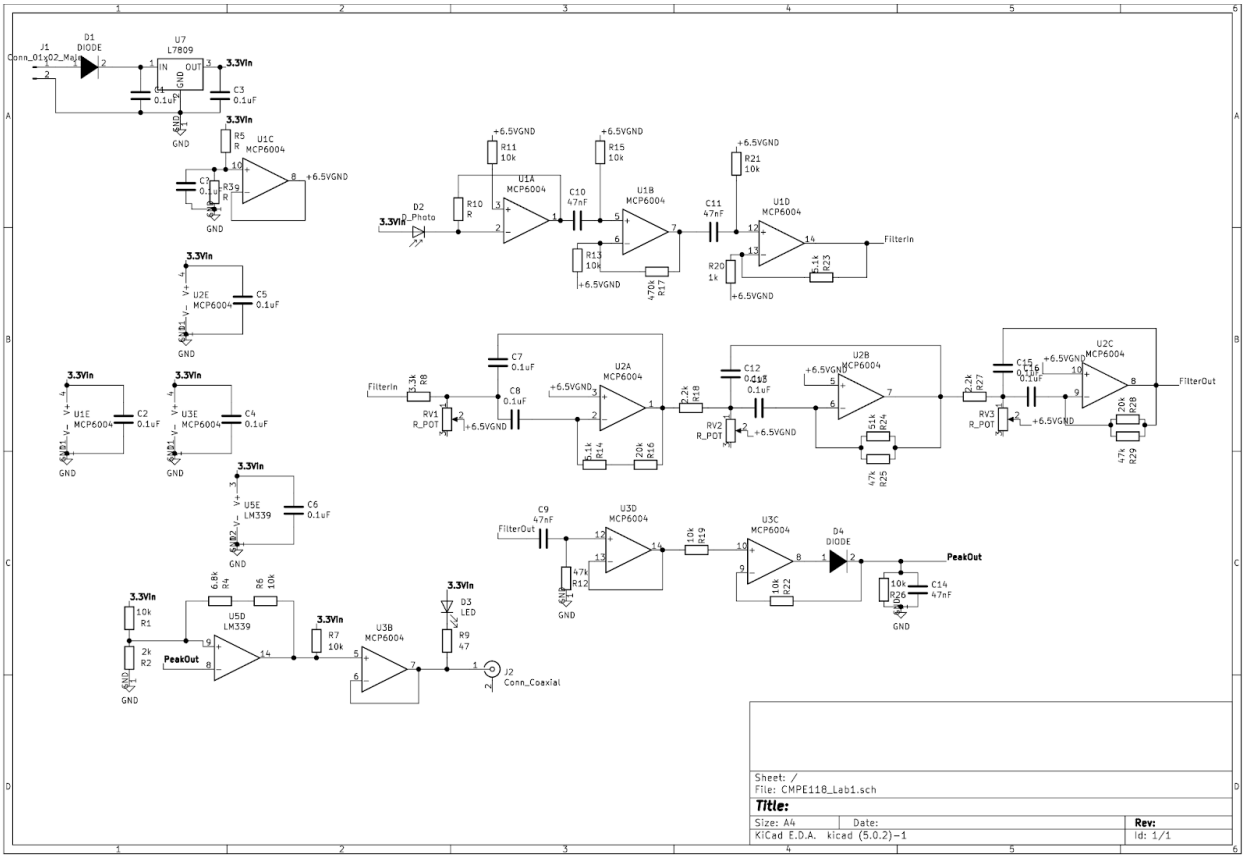


Figure 13: PCB Emplaced on the Right Pillar of Bot

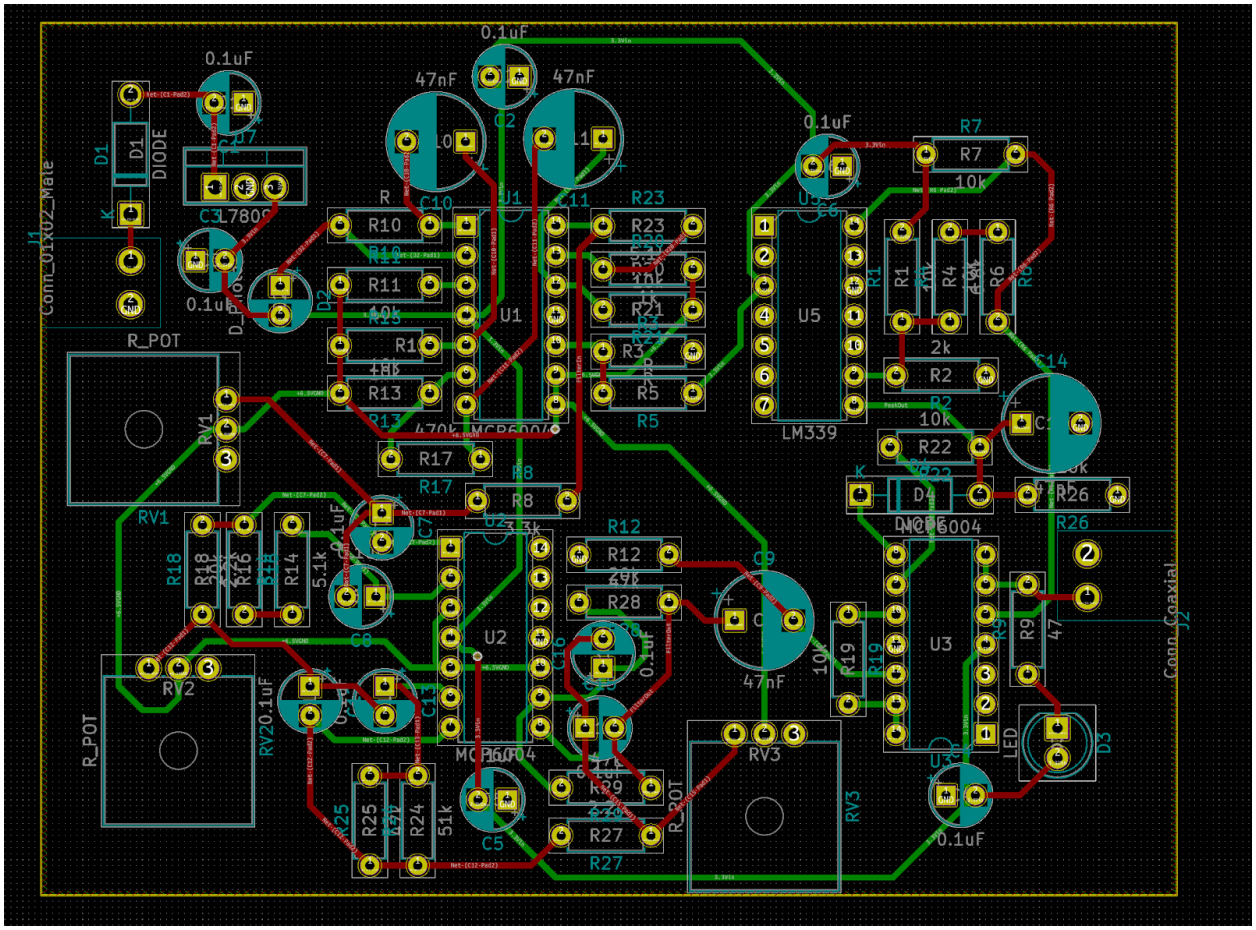
All the droids are required to mount a 2K Hertz IR emitting bacon at a height of exact 11 in. Outside of DSP (Digital Signal processing) a camera image, a beacon detector would be the easiest and most reliable way to locate the opponent droid, which is essential for the navigation of our own field and the complete completion of the final goal. However, the opponent Droid is not the only source of IR on the field. All of the obstacles come with their own 1.5K Hertzand 2.5K hertz mini-beacon emitters mounted around 7 in high. This meant the beacon detector must have a decent bandpass filter that only sees 2k hertz signals and rejects the obstacle frequencies.

Figure 14: Field Obstacles

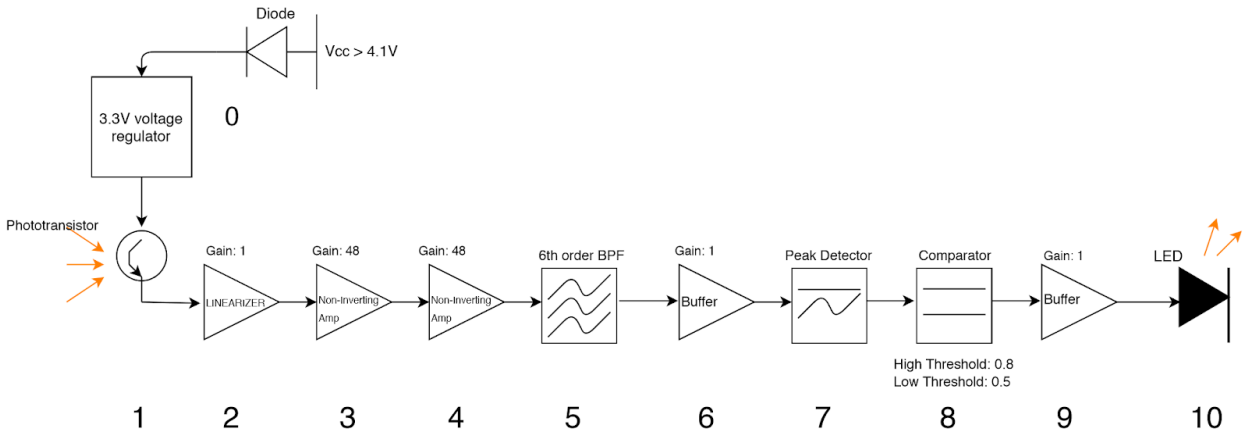
Schematic:


J1
Conn_01x02_Male
D1
DIODE
U7
L7809
IN
GND
OUT
3.3Vin
C4
0.1uF
C3
0.1uF
GND
R5
R
U1C
MCP6004
C1
0.1uF
R39
R
+6.5VGND
U2E
MCP6004
C5
0.1uF
U1E
MCP6004
C2
0.1uF
U3E
MCP6004
C4
0.1uF
U5E
LM339
C6
0.1uF
D2
Photo
R10
R
U1A
MCP6004
R11
10k
C10
47nF
R13
10k
U1B
MCP6004
470k
R17
C11
47nF
R20
1k
U1D
MCP6004
R21
10k
R23
5.1k
FilterIn
C7
0.1uF
C8
0.1uF
RV1
R_POT
3.3k
R8
U2A
MCP6004
1k
R14
20k
R16
C12
0.1uF
RV2
R_POT
2.2k
R18
U2B
MCP6004
51k
R24
47k
R25
C15
0.1uF
RV3
R_POT
2.2k
R27
U2C
MCP6004
20k
R28
47k
R29
FilterOut
C9
47nF
R12
47k
U3D
MCP6004
10k
R19
U3C
MCP6004
10k
R22
D4
DIODE
PeakOut
10k
R26
C14
47nF
10k
R1
2k
R2
PeakOut
6.8k
R4
10k
R6
U5D
LM339
R7
10k
U3B
MCP6004
D3
LED
R9
47
J2
Conn_Coaxial
Sheet: /
File: CMPE118_Lab1.sch
Title:
Size: A4
Date:
Rev:
KiCad E.D.A. kicad (5.0.2)-1
Id: 1/1

PCB Design:



Block Diagram:



0. **Voltage Regulator and Diode:** The initial stage is used to regulate the voltage coming in from the power supply. The first measure taken was to put in a diode from Vcc to the rest of the circuit, this is done so that if one for example plugs in the power incorrectly (where ground is power and power is ground), the reverse

polarization will not be able to travel through the diode due to the diodes characteristics. The current can only flow through it in one direction thus for their to be current flowing only one polarization will work. The voltage drop across it as well, when on, will be approximately 0.7V. After this diode you will have a 3.3V regulator which will regulate the income voltage no matter how high to be 3.3V. Of course if you insert a voltage that is too high the element will start to break down and not work. But in the set up we gave it, the regulator is used just in case there are any surges in power that the voltage provided to the circuit will remain at 3.3V. This will also be useful in future projects if we need a power rail for other components that is higher than 3.3V, the inserted power will be regulated to 3.3V Of course to do this you will have to make sure that the incoming voltage is greater than 4.1V. This is so that the voltage that finally does reach the voltage regulator (after the 0.7V drop across the diode), will be greater than 3.3V.

1. **Phototransistor:** The first stage of the beacon detector is the phototransistor. This is where the input signal will originate due to light being emitted from the beacon being converted into voltage by the phototransistor that sees it. The light that will be emitted from the beacon oscillates at the frequencies of 2kHz, 1.5kHz, and 2.5kHz. These light signals will be translated by the phototransistor which will turn on and produce a Vbe proportional to the signal being picked up. This Vbe will let a certain amount of collector current through and thus this current is proportional to the signal that is being seen by the phototransistor.

2. **Linearizer:** This stage is used to help improve the non-linearity of the later amplifier stages, this will also help improve the power eciency of the circuit. You can see a linearizer as a transresistive op-amp stage, this stage will output a voltage proportional to its input current. One can also think of this stage as a current controlled voltage source (CCVS). Since the phototransistor outputs a current proportional to the signal, this stage will then convert this current to a voltage signal proportional to the beacon signal being detected.

3. **Non-Inverting Op-Amp:** Then a non-inverting amplifier stage is used to boost the signal being received. One will want to put a DC blocking capacitor between this stage and the linearizer so that the DC o↵set of the linearizer output isn't also amplified. You will also want to put a load resistor of 10kΩ to properly load the input of the op-amp. You will be using non-inverting op-amps so that the beacon can be placed at farther distances from the phototransistor and will still be readable. This non-inverting amplifier used resistor values of 10kΩ (for resistor to virtual ground) and 47Ω (for feedback) giving us a gain of 5.7. One will also want to use a virtual ground of 1.65V so as to keep the signal above zero volts for maximum swing. If biased at real ground, the negative amplitude of the signal

will rail at 0V, only keeping half of the signal. We used a smaller gain so as to deter the circuit from oscillating. We will use several amplification stages so as to maximize amplification without risk of oscillations from one particular stage due to too much gain.

4. **Non-Inverting Op-Amp:** This stage was used to further amplify our signal and thus give us a bigger range that our beacon can be detected. You will want to do the same thing to the input of this op-amp by using a DC blocking capacitor and 10kΩ resistor to ground so as to properly load the input and to make sure that the DC bias of the output of the last stage does not get amplified in this stage. You will also want to use a virtual ground of 1.65 volts for the same reasons given in the previous stage. The resistor values used for this amplifier were 5.1kΩ (for the feedback resistor) and 1kΩ (for resistor to virtual ground). This results in a gain of 6.1. Thus the total gain thus far has been 34.8.

5. **6th Order BandPass Filter:** The next stage implemented was the bandpass filter. This stage was used so that the beacon detector could be selective between the frequencies of 1.5kHz, 2kHz, and 2.5kHz. This bandpass filter will pass a 2 kHz signal while attenuating any other frequency. We decided to use a 6th order bandpass filter to make sure that there was substantial rolloff. This means that the filter will contain 6 poles thus having a steeper drop off slope from the 3dB point. For this filter, a good reference to use would be Dale's homemade robotics and switched out the grounds for a virtual ground of 1.65V. To derive the values you will need for the 6th order filter topology you will want to use Electronic Filter Design Handbook by Arthur P. Williams and Fred J. Taylor. This book helps you derive the values that Dale's website will provide to you thus giving you a better understanding of the workings of your bandpass filter. The derivation for the circuit values are shown on a later page. This bandpass filter, having 6 poles, will thus have 3 stages, each containing 2 poles. The resistor from the negative input to virtual ground are the ones that affect the center frequency of each stage. Thus we inserted 100Ω potentiometers that we could very to get detailed center frequencies of 2 kHz as well as have more flexability in the attenuation of the 1.5kHz and 2.5kHz signals. Each stage will be its own bandpass and will have different Q's according to the derivations. We will want our overall Q to be high because we want the bandwidth to be as small as possible. This is so that the -3dB drop off can happen as soon as it can from the max 2 kHz peak thus giving the other 2 frequencies maximum attenuation. We designed our filter to have a 250 Hz bandwidth with a gain of 10. Since we are using active filters the gain is an advantage, instead of using a passive filter.

6. **Buffer:** The next stage is a buffer, this is so the next stage does not interfere with the previous filter stage. This stage is also used to help impedance match the circuit so that maximum power transfer can be achieved. (mismatched impedances leads to reflections and less transmissions thus resulting in power loss and some interference). You will want to use a capacitor and a 10k to ground at the input to the buffer so as to DC block the signal to only let the AC through while also loading the input properly and allowing the capacitor to discharge when it needs to fluctuate in voltage (for the voltage signal).

7. **Peak Detector:** This stage is used to convert the AC signal into a somewhat DC signal (with some ripple tolerated due to the fast discharge speed). In order to connect the non-inverting op-amp stages to the peak detector's input, a DC blocking capacitor is used to remove the bias. In order to make sure that the DC blocking capacitor works efficiently, a large resistor to ground is placed after it to minimize signal loss. The resistor is needed to discharge the DC blocking capacitor when the output of the peak detector is lower than the capacitor voltage. The resistor and capacitor values we decided to use were 47nF and 47kΩ. We chose these values due to the standard capacitor value and the magnitude of the resistor. These values can be flexible but they 12 should stay in the same magnitude.

We used a feedback resistor of 10k with an input resistor of 10kΩ as described in Basic Electronics Tutorials. Using these resistor values instead of just doing a feedback and input with no impedances will help avoid loading while charging the capacitor. We found the RC values as shown below: (equations and inspiration derived the EE-171 textbook Fundamentals of Microelectronics) A suitable decay time given the input can be around 0.74ms depending on your input frequency and DC bias voltage. The decision was to make the input frequency 27kHz with a DC bias of 3.3V, thus the period is 40 x 10^6. This will be shown in the peak detector calculations that are given after the block diagram explanations.

8. **Comparator:** The next stage is used to turn the mostly DC signal from the peak detector turn the signal into a true DC logic level voltage. The comparator is used with hysteresis bounds, where if the voltage is above these bounds the comparator will output a low signal (0V DC approx.). If the voltage is below these bounds the comparator will output a high signal (2.7-3.3V DC approx.) We tested our circuit at the closest and farthest ranges we thought suitable for this lab 1 ft - 7 ft and we took measurements on what the peak voltages were for the 3 frequencies being sent by the beacon. We set the low hysteresis threshold to be 0.5V because this was the highest voltage output of the 2.5kHz and 1.5kHz at the closest range. Then we set the highest hysteresis threshold to be 0.8V because this

was the minimum voltage that the 2 kHz signal was giving at 7ft away. Using these voltages as our bounds we calculated the resistor values needed to implement this comparator as shown on a later page. Using these values and a pull up resistor of 3.3kΩ to maximize the hysteresis bounds (due to the comparator being an open collector output device) we were able to achieve a very good comparator that worked for the signals given.

9. **Buffer:** Another buffer stage was used to isolate the comparator from the LED so as to not alter the hysteresis given by the comparator. Without the buffer, changing voltages that turns on and off the LED can change the resistances that the comparator needs to set the thresholds. This buffer will not only help isolate the stages, but it will also help with impedance matching as well.

10. **LED:** The output of the buffer will be inserted into the LED stage which has an LED connected from Vcc at the anode and then goes through a 47 ohm resistor to the output of the buffer. More detail on this stage is written in section 2, since we implemented the same design. Due to this particular design the LED will light up when the inputting voltage from the buffer is low and will turn off when the inputting voltage is high. This coincides with the comparator since it outputs a low voltage for when there is a 2 kHz signal received and a high voltage for when the 2kHz is not received. Thus the stages and the design of our beacon detector was implemented and finished successfully!

Here are the calculations done for the comparator, peak detector, and filter:

Peak Detector Calculations

Equation for ripple voltage, assuming we will want a maximum ripple of 5%, we will use the equation for ripple voltage as stated:

$$V_R = 0.05 \times 3.3V = 0.165V$$

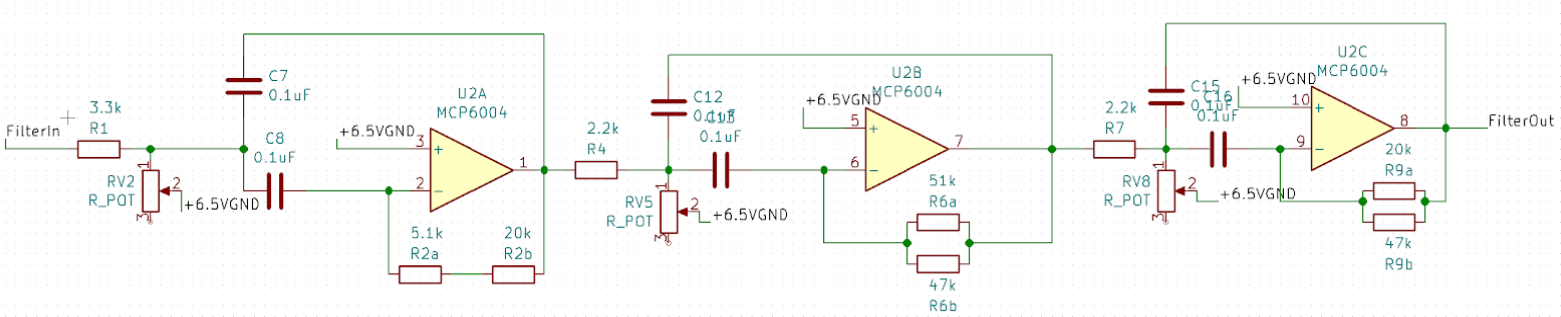
$$V_R = \frac{V_P - V_{D,on}}{R_L} \times \frac{T_{on}}{C_1}, T_{in} = \frac{1}{f_r}$$

$$R_L C_1 = \frac{V_{in} \times T_{in}}{V_R} = \frac{3.3(\frac{1}{27\times10^3})}{0.165} = 0.7407$$

Choosing $C_1 = 10uF$, we can calculate R_L to equal 7.4kΩ. We rounded the value to 10kΩ because it is a standard value, while not changing the ripple error by much.

We wanted the "leak" time constant to be small enough to prevent lagging thus we just used the frequency of the signal given to derive our period as our time constant τ.

Schematic of calculations shown for filter with standard resistor values accounted for:



Active Bandpass Filter Calculation

This bandpass filter is an active 6th order Butterworth filter with a gain of 10. The filter will have 6 poles, which will make a steeper rolloff past the -3dB point. To find the topology and calculate the values needed, we looked at Electronic Filter Design Handbook by Arthur P. Williams and Fred J. Taylor. The book helped us derive the equations to find the resistors and capacitors needed. We also referenced a website that was a calculator version of the derivation to double check our answers (http://www.wa4dsy.com/robot/bandpass-filter-calc). The derivation for the filter is shown below:

Bandpass Q:

$$Q_{bp} = \frac{f_0}{BW_{3dB}} = \frac{2kHz}{250Hz} = 8$$

For a 6^{th} order filter:

$$\alpha = 0.5$$
$$\beta = 0.866$$
$$C = \alpha^2 + \beta^2 = 0.99956$$
$$D = \frac{2\alpha}{Q_{bp}} = 0.125$$
$$E = \frac{C}{Q_{bp}^2 + 4} = 4.0156$$
$$G = \sqrt{E^2 - 4D^2} = 4.0078$$
$$Q_1 = \sqrt{\frac{E+G}{2D^2}} = 16.023$$
$$M = \frac{\alpha Q_1}{Q_{bp}} = 1.00146$$
$$W = M + \sqrt{M^2 - 1} = 1.056$$
$$f_{ra} = \frac{f_o}{W} = 1.89475kHz$$
$$f_{rb} = W f_0 = 2.111089kHz$$
$$Q_2 = \frac{Q_{bp}}{\alpha_0} = 8$$
$$f_r = f_0 = 2kHz$$

$$A_0 = \sqrt[3]{A} = 2.15$$

$$A_{r1} = A_0\sqrt{1 + Q_1^2(\frac{f_0}{f_{ra}} - \frac{f_{ra}}{f_0})^2} = 4.302$$

$$A_{r2} = A_0\sqrt{1 + Q_1^2(\frac{f_0}{f_{rb}} - \frac{f_{rb}}{f_0})^2} = 4.302$$

$$A_{r3} = A_0\sqrt{1 + Q_2^2(\frac{f_0}{f_0} - \frac{f_0}{f_0})^2} = 2.15$$

$$R_3 = \frac{Q}{\pi f_r C} = 26.917k\Omega$$

$$R_1 = \frac{R_3}{2A_r} = 3.13k\Omega$$

$$R_2 = \frac{R_3}{2(2(2Q^2) - A_r)} = 26.43\Omega$$

$$R_6 = \frac{Q}{\pi f_r C} = 24.159k\Omega$$

$$R_4 = \frac{R_6}{2A_r} = 2.807k\Omega$$

$$R_5 = \frac{R_6}{2(2(2Q^2) - A_r)} = 23.73\Omega$$

$$R_9 = \frac{Q}{\pi f_r C} = 12.732k\Omega$$

$$R_7 = \frac{R_9}{2A_r} = 2.961k\Omega$$

$$R_8 = \frac{R_9}{2(2(2Q^2) - A_r)} = 50.5\Omega$$

Comparator Calculation

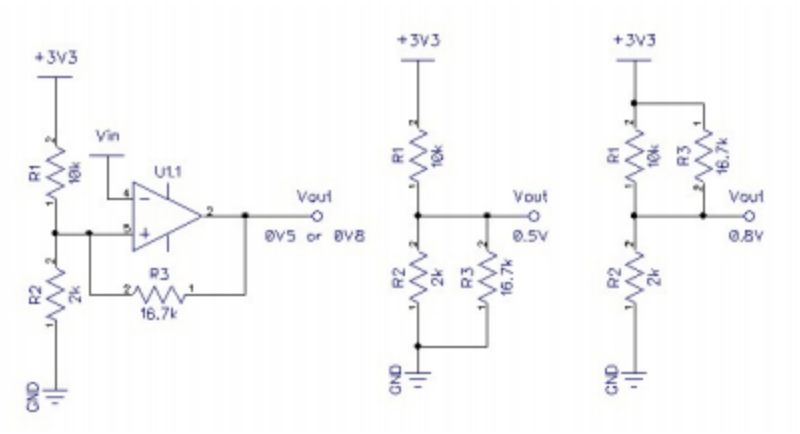


Figure 26: Comparator Equivalent Circuit

As shown in TI Designs - Comparator with Hysteresis Reference Design

$$V_H = 0.8V$$
$$V_L = 0.5V$$
$$V_{CC} = 3.3V$$

$$\frac{R_3}{R_1} = \frac{V_L}{V_H - V_L} = \frac{0.5}{0.3} = 1.67$$
$$\frac{R_2}{R_1} = \frac{V_L}{V_{CC} - V_H} = \frac{0.5}{2.5} = 0.2$$

Choosing $R_1 = 10k\Omega$

$$R_3 = 1.67 \times R_1 = 16.7k\Omega$$
$$R_2 = 0.2 \times R_1 = 2k\Omega$$

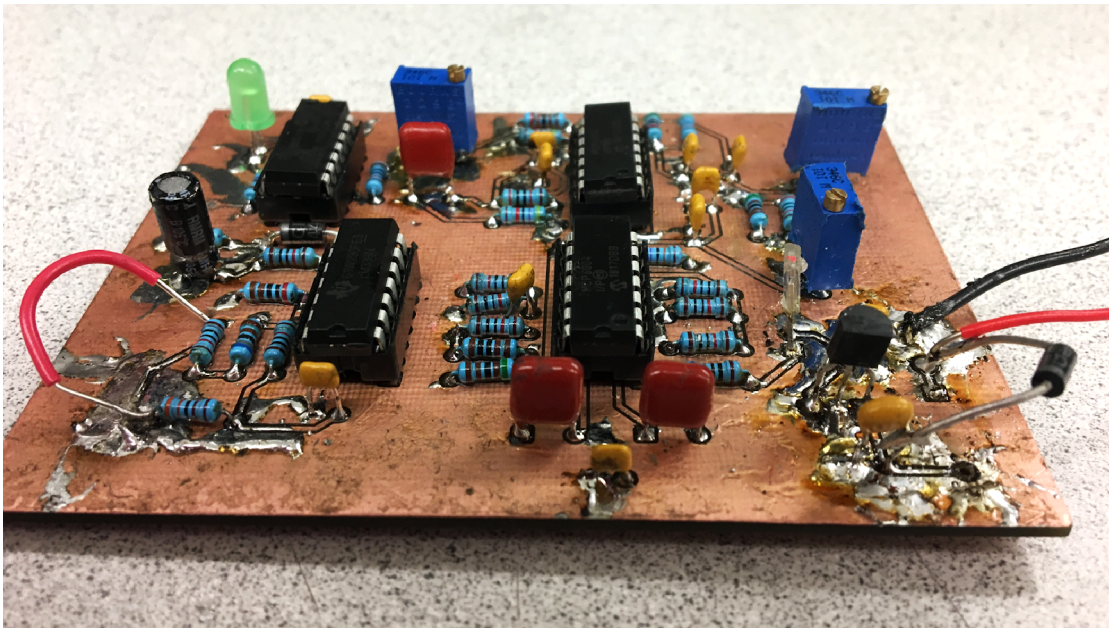


Figure 15: Beacon Detector PCB

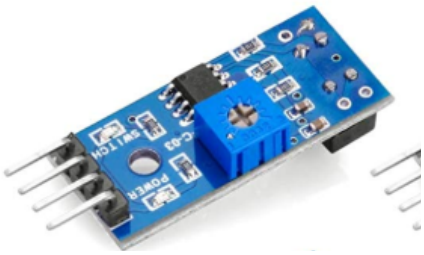
In order to utilize this beacon detector, the signal from the board was fed into an AD pin. This was chosen as the signal out was not railed high or low enough to signal a 0 or 1, digitally. The analog output resulted in being under 400 for a Beacon detection and above 800 when no detection was noted. After determining these values, the values were used as threshold bounds for beacon events of TARGET_LOCATED and NO_TARGET along with some small debouncing. A helper library, BeaconDetector.h/c was created where the initialization of the pin and reading the pin would be called. This allowed for a cleaner Hierarchical State Machine where all functions are defined outside of the states and event checker.

```
/**
 * @Function BeaconDetector_Init(void)
 * @param None.
 * @return None.
 * @brief  Performs all the initialization necessary for the Beacon Detector.
 * @note  None.
 * @author George Leece 5/9/2019 */
void BeaconDetector_Init(void){
   AD_Init();
   AD_AddPins(AD_PORTW7);
}

/**
```

```
 * @Function CheckCenterTapeSensor(void)
 * @param None.
 * @return char - Returns the status of the Beacon Detector.
 * @brief  Checks the status of the Beacon Detector.
 * @note  None.
 * @author George Leece 5/9/2019 */
uint16_t CheckBeaconDetector(void){
   return AD_ReadADPin(AD_PORTW7);
}
```

Tape Sensors



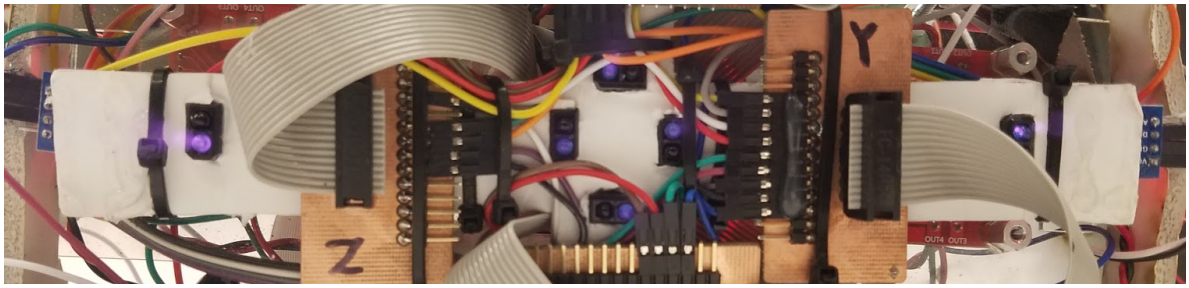


Figure 16: Underside of Tape Sensors

To navigate the field, tape sensors are used. Synchronous sampling is used to interpret the readings from the tape sensors and there reading are used to produce tape sensor events.The final layout of the tape sensors consisted of four in a square orientation in the center of the bot, with two on the “X” extremities. The X configuration consisted of the Far Right, Right, Left, and Far Left. The Y configuration consisted of the Center and Rear Tape Sensors. Combinations of these tape sensors indicated different events such as ON/OFF_X/YTAPE events, ON/OFF_Y/X_CORNER events and more.

A helper library, TapeSensors.h/c was created where initialization, individual checking of sensors and a function checking the output of all tape sensors are housed. Again this helped provide a cleaner state machine with all of the functions defined in a name specific library. Along with this another library Corners.h/c was created, where all corner

events and tape events are defined. Masks utilized with a #define would be utilized to check the tape sensors activity and to signify events corresponding to certain sensors being off tape. Depending on the bot's movement of right/left or of forward/reverse, Along with these libraries were test harnesses to ensure the proper functionality of these libraries. When these functions were utilized in the state machine, debouncing of events was included in the event checker to not spam the state machine with a mass of events.

TapeSensors.c:

```
/**
 * @Function TapeSensors_Init(void)
 * @param None.
 * @return None.
 * @brief  Performs all the initialization necessary for the Tape Sensors.
 * @note  None.
 * @author George Leece 5/9/2019 */
void TapeSensors_Init(void) {
   PORTY09_TRIS = 1; // Far Right Sensor
   PORTY08_TRIS = 1; // Right Sensor
   PORTY07_TRIS = 1; // Center Sensor
   PORTY06_TRIS = 1; // Left Sensor
   PORTY05_TRIS = 1; // Far Left Sensor
   PORTY11_TRIS = 1; // Rear Sensor
}

/**
 * @Function CheckCenterTapeSensor(void)
 * @param None.
 * @return char - Returns ON_TAPE or OFF_TAPE for the Center Tape Sensor.
 * @brief  Checks the status of the center tape sensor.
 * @note  None.
 * @author George Leece 5/9/2019 */
uint8_t CheckCenterTapeSensor(void) {
   return PORTY07_BIT;
}

/**
 * @Function CheckRightTapeSensor(void)
 * @param None.
 * @return char - Returns ON_TAPE or OFF_TAPE for the Right Tape Sensor.
 * @brief  Checks the status of the Right Tape Sensor.
 * @note  None.
 * @author George Leece 5/9/2019 */
```

```
uint8_t CheckRightTapeSensor(void) {
   return PORTY08_BIT;
}

/**
 * @Function CheckLeftTapeSensor(void)
 * @param None.
 * @return char - Returns ON_TAPE or OFF_TAPE for the Left Tape Sensor.
 * @brief  Checks the status of the Left Tape Sensor.
 * @note  None.
 * @author George Leece 5/9/2019 */
uint8_t CheckLeftTapeSensor(void) {
   return PORTY06_BIT;
}

/**
 * @Function CheckFarRightTapeSensor(void)
 * @param None.
 * @return char - Returns ON_TAPE or OFF_TAPE for the Far Right Tape Sensor.
 * @brief  Checks the status of the Far Right Tape Sensor.
 * @note  None.
 * @author George Leece 5/9/2019 */
uint8_t CheckFarRightTapeSensor(void) {
   return PORTY09_BIT;
}

/**
 * @Function CheckFarLeftTapeSensor(void)
 * @param None.
 * @return char - Returns ON_TAPE or OFF_TAPE for the Far Left Tape Sensor.
 * @brief  Checks the status of the Far Left Tape Sensor.
 * @note  None.
 * @author George Leece 5/9/2019 */
uint8_t CheckFarLeftTapeSensor(void) {
   return PORTY05_BIT;
}


/**
 * @Function CheckRearTapeSensor(void)
 * @param None.
 * @return char - Returns ON_TAPE or OFF_TAPE for the Rear Tape Sensor.
 * @brief  Checks the status of the Rear Tape Sensor.
 * @note  None.
```

```
 * @author George Leece 5/10/2019 */
uint8_t CheckRearTapeSensor(void) {
   return PORTY11_BIT;
}


/**
 * @Function CheckAllTapeSensors(void)
 * @param None.
 * @return char - Calls the Check Tape Function for all of the sensors. The result
 * is then ORed with the first bit corresponding to the Far Right, second bit is
 * Right, third is Center, fourth is Left, fifth is Far Left, and sixth is Rear. A 1 in
 * represents ON_TAPE for the sensor, 0 is OFF_TAPE for the sensor.
 * @brief  Checks the status of the All Tape Sensors. ORs all of the bits
 * @note  None.
 * @author George Leece 5/9/2019 */
uint8_t CheckAllTapeSensors(void) {
   uint8_t x;
   x = CheckRearTapeSensor() << 5;
   x |= CheckFarLeftTapeSensor() << 4;
   x |= (CheckLeftTapeSensor() << 3);
   x |= (CheckCenterTapeSensor() << 2);
   x |= (CheckRightTapeSensor() << 1);
   x |= CheckFarRightTapeSensor();
   return x;
}
```

Corners.c:

```
uint16_t Far_Left_Off(void) {
   sensors = CheckAllTapeSensors();
   if ((sensors == 0x2f)){
      return TRUE;
   } else {
      return FALSE;
   }
}

uint16_t Far_Right_Off(void) {
   sensors = CheckAllTapeSensors();
   if ((sensors == 0x3e)){
      return TRUE;
   } else {
```

```
      return FALSE;
   }
}

uint16_t Right_Off(void) {
   sensors = CheckAllTapeSensors();
   if((sensors == 0x3C)){
      return TRUE;
   } else {
      return FALSE;
   }
}

uint16_t Left_Off(void) {
   sensors = CheckAllTapeSensors();
   if((sensors == 0x27)){
      return TRUE;
   } else {
      return FALSE;
   }
}

uint16_t On_YTape(void) {
   sensors = CheckAllTapeSensors();
   if ((sensors & CMASK) && (sensors & REMASK)) {
      return TRUE;
   } else {
      return FALSE;
   }
}

uint16_t On_XTape(void) {
   sensors = CheckAllTapeSensors();
   if ((sensors & LMASK)&&(sensors & RMASK)&&((sensors & FRMASK)||(sensors & FLMASK))) {
      return TRUE;
   } else {
      return FALSE;
   }
}

uint16_t Off_XTape(void) {
   sensors = CheckAllTapeSensors();
   if ((On_XTape() == 0) && (All_On_Tape() == 0)) {
      return TRUE;
```

```
    } else {
        return FALSE;
    }
}

uint16_t Off_YTape(void) {
    sensors = CheckAllTapeSensors();
    if ((On_YTape() == 0) && (All_On_Tape() == 0)) {
        return TRUE;
    } else {
        return FALSE;
    }
}

uint16_t All_On_Tape(void) {
    sensors = CheckAllTapeSensors();
    if (sensors == (FRMASK | RMASK | CMASK | LMASK | FLMASK | REMASK)) {
        return TRUE;
    } else {
        return FALSE;
    }
}

uint16_t In_Y_Corner(void) {
    if (Far_Left_Off() || Far_Right_Off()) {
        return TRUE;
    } else {
        return FALSE;
    }
}

uint16_t In_X_Corner(void) {
    if (Right_Off() || Left_Off()) {
        return TRUE;
    } else {
        return FALSE;
    }
}
```

Mecanum

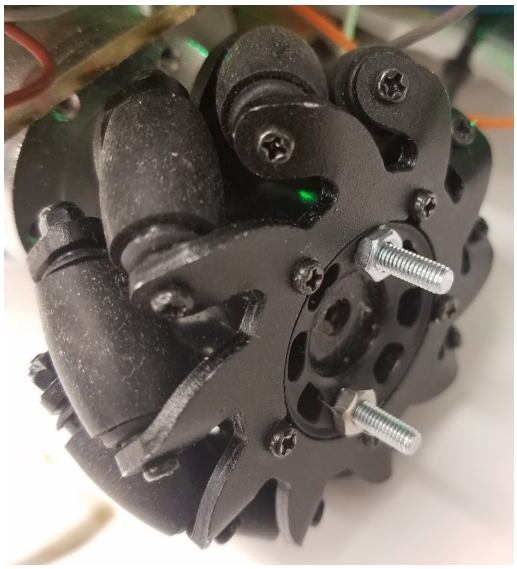


Figure 17: A Mecanum Wheel

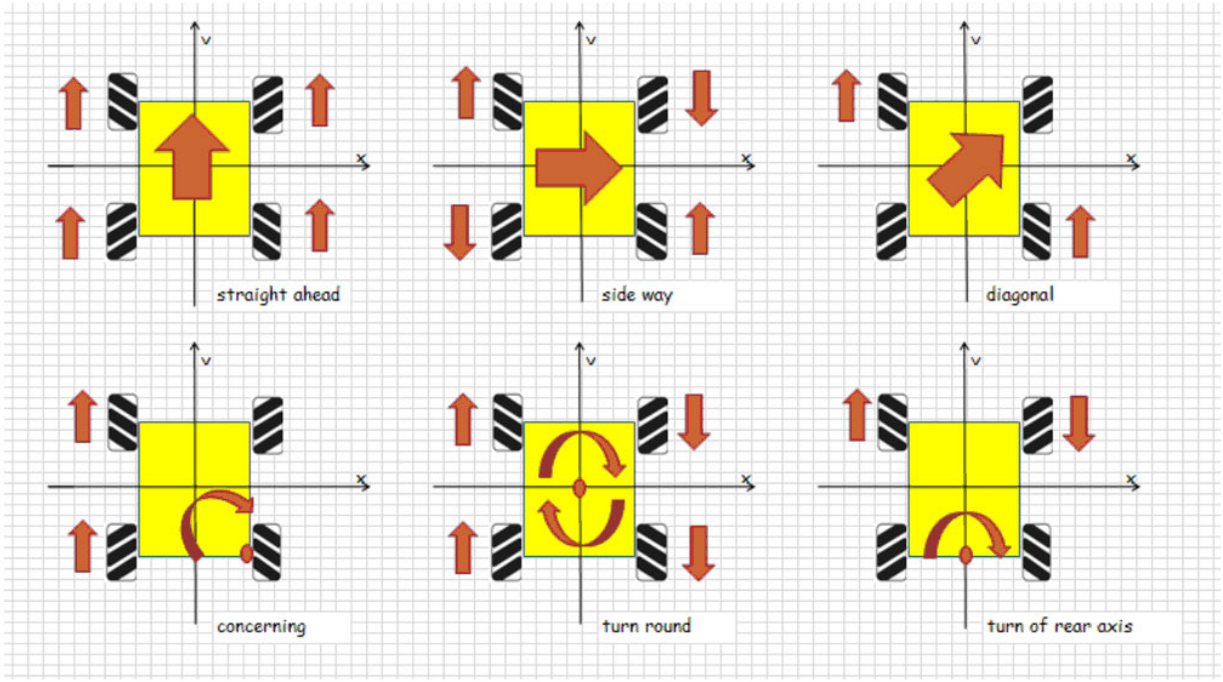


Figure 18: Ranges of Motion for Mecanum Wheels

Mecanum wheels were chosen for their ability to traverse across the field in a multitude of directions without changing the orientation of the bot. Even after being advised against utilizing the unusual drive, the drive design was accomplished within the first

In order to operate the mecanum wheels two a single library Mecanum.h was constructed. This header is composed of two source files, Mecanum.c and Motor_Drive.c. The Motor_Drive source file contained the code for individual wheel movement by defining each wheel, the PWM duty cycle to be sent, and the direction to the H-Bridge. Mecanum.c combines these three simple forward, reverse, and stop motor movements into the bot's movements of forward, reverse, left, right, and rotation clockwise/counterclockwise. While mecanum wheels offer more movement directions such as diagonal movement, these were not utilized as no need was seen for them.

Motor_Drive.c:

```
/**
 * @Function Motors_Init(void)
 * @param None.
 * @return None.
 * @brief  Performs all the initialization necessary for the bot. This includes initializing
 * the PWM module and the data directions on some pins
 * @note  None.
 * @author George Leece 5/8/2019 */
void Motors_Init(void){
   PWM_Init();
   PWM_SetFrequency(PWM_DEFAULT_FREQUENCY);
   PWM_AddPins(PWM_PORTZ06 | PWM_PORTY12 | PWM_PORTY10 | PWM_PORTY04);
   PORTV03_TRIS = 0;
   PORTV04_TRIS = 0;
   PORTZ11_TRIS = 0;
   PORTZ08_TRIS = 0;
}


/**
 * @Function Drive_Stop(Motors Motor)
 * @param Motors Motor - The motor to be changed
 * @return None.
 * @brief  This function is used to set the designated motor to a duty cycle of 0
 * @note  None.
 * @author George Leece 5/8/2019 */
void Drive_Stop(Motors Motor){
   switch (Motor) {
   case FRMotor:
      PWM_SetDutyCycle(PWM_PORTZ06, DutyCycleZero);
               PORTZ11_LAT = 0;
      break;
   case FLMotor:
      PWM_SetDutyCycle(PWM_PORTY12, DutyCycleZero);
      PORTV03_LAT = 0;
      break;
   case BRMotor:
      PWM_SetDutyCycle(PWM_PORTY10, DutyCycleZero);
      PORTV04_LAT = 0;
      break;
   case BLMotor:
      PWM_SetDutyCycle(PWM_PORTY04, DutyCycleZero);
                        PORTZ08_LAT = 0;
      break;
   default:
      break;
   }
```

```
}
/**
 * @Function Drive_Forwards(Motors Motor)
 * @param Motors Motor - The motor to be changed
 * @return None.
 * @brief  This function is used to set the designated motor to a defined Duty Cycle.
 * the direction is set to 0 for the designated motor
 * @note  None.
 * @author George Leece 5/8/2019 */
void Drive_Forwards(Motors Motor, uint16_t Motor_Speed){
   switch (Motor) {
   case FRMotor:
      PWM_SetDutyCycle(PWM_PORTZ06, Motor_Speed);
                        PORTZ11_LAT = 0;
      break;
   case FLMotor:
      PWM_SetDutyCycle(PWM_PORTY12, Motor_Speed);
      PORTV03_LAT = 0;
      break;
   case BRMotor:
      PWM_SetDutyCycle(PWM_PORTY10, Motor_Speed);
      PORTV04_LAT = 0;
      break;
   case BLMotor:
      PWM_SetDutyCycle(PWM_PORTY04, Motor_Speed);
                        PORTZ08_LAT = 0;
      break;
   default:
      break;
   }
}


/**
 * @Function Drive_Reverse(Motors Motor)
 * @param Motors Motor - The motor to be changed
 * @return None.
 * @brief  This function is used to set the designated motor to a defined Duty Cycle.
 * the direction is set to 1 for the designated motor
 * @note  None.
 * @author George Leece 5/8/2019 */
void Drive_Reverse(Motors Motor, uint16_t Motor_Speed){

   switch (Motor) {
   case FRMotor:
      PWM_SetDutyCycle(PWM_PORTZ06, Motor_Speed);
               PORTZ11_LAT = 1;
      break;
   case FLMotor:
      PWM_SetDutyCycle(PWM_PORTY12, Motor_Speed);
      PORTV03_LAT = 1;
      break;
   case BRMotor:
      PWM_SetDutyCycle(PWM_PORTY10, Motor_Speed);
      PORTV04_LAT = 1;
      break;
   case BLMotor:
      PWM_SetDutyCycle(PWM_PORTY04, Motor_Speed);
                        PORTZ08_LAT = 1;
      break;
   default:
      break;
   }
}
```

Mecanum.c:

```
#define DutyCycle 500
#define DutyCycleFR
#define DutyCycleRot 500


void Bot_Forwards(void) {
   Drive_Forwards(FRMotor,DutyCycle);
   Drive_Forwards(FLMotor,DutyCycle);
   Drive_Forwards(BRMotor,DutyCycle);
   Drive_Forwards(BLMotor,DutyCycle);
}

void Bot_Reverse(void){
   Drive_Reverse(FRMotor,DutyCycle);
   Drive_Reverse(FLMotor,DutyCycle);
   Drive_Reverse(BRMotor,DutyCycle);
   Drive_Reverse(BLMotor,DutyCycle);
}

void Bot_Left(void){
   Drive_Forwards(FRMotor,DutyCycle);
   Drive_Reverse(FLMotor,DutyCycle);
   Drive_Reverse(BRMotor,DutyCycle);
   Drive_Forwards(BLMotor,DutyCycle);
}

void Bot_Right(void){
   Drive_Reverse(FRMotor,DutyCycle);
   Drive_Forwards(FLMotor,DutyCycle);
   Drive_Forwards(BRMotor,DutyCycle);
   Drive_Reverse(BLMotor,DutyCycle);
}

void Bot_DiagonalFL(void){
   Drive_Forwards(FRMotor,DutyCycle);
   Drive_Forwards(BLMotor,DutyCycle);
   Drive_Stop(FLMotor);
   Drive_Stop(BRMotor);
}

void Bot_DiagonalFR(void){
   Drive_Stop(FRMotor);
   Drive_Forwards(FLMotor,DutyCycle);
   Drive_Forwards(BRMotor,DutyCycle);
   Drive_Stop(BLMotor);
}

void Bot_DiagonalBL(void){
   Drive_Stop(FRMotor);
   Drive_Reverse(FLMotor,DutyCycle);
   Drive_Reverse(BRMotor,DutyCycle);
   Drive_Stop(BLMotor);
}


void Bot_DiagonalBR(void){
   Drive_Reverse(FRMotor,DutyCycle);
   Drive_Stop(FLMotor);
   Drive_Stop(BRMotor);
   Drive_Reverse(BLMotor,DutyCycle);
}
```

```
void Bot_Stop(void){
   Drive_Stop(FRMotor);
   Drive_Stop(FLMotor);
   Drive_Stop(BRMotor);
   Drive_Stop(BLMotor);
}

void Bot_RotateCC(void){
   Drive_Reverse(FRMotor,DutyCycleRot);
   Drive_Forwards(FLMotor,DutyCycleRot);
   Drive_Reverse(BRMotor,DutyCycleRot);
   Drive_Forwards(BLMotor,DutyCycleRot);
}

//void Bot_RotateCC(void){
//    Drive_Stop(FLMotor);
//    Drive_Forwards(FRMotor,DutyCycle);
//    Drive_Reverse(BLMotor,DutyCycle);
//    Drive_Stop(BRMotor);
//}

void Bot_RotateACC(void){
   Drive_Forwards(FRMotor,DutyCycleRot);
   Drive_Reverse(FLMotor,DutyCycleRot);
   Drive_Forwards(BRMotor,DutyCycleRot);
   Drive_Reverse(BLMotor,DutyCycleRot);
}

//void Bot_RotateACC(void){
//    Drive_Stop(FLMotor);
//    Drive_Reverse(FRMotor,DutyCycle);
//    Drive_Forwards(BLMotor,DutyCycle);
//    Drive_Stop(BRMotor);
//}

void Bot_RotateRight(void){
   Drive_Forwards(FLMotor,DutyCycle);
   Drive_Forwards(BLMotor,DutyCycle);
   Drive_Stop(FRMotor);
   Drive_Stop(BRMotor);
}


void Bot_RotateLeft(void){
   Drive_Forwards(FRMotor,DutyCycle);
   Drive_Forwards(BRMotor,DutyCycle);
   Drive_Stop(FLMotor);
   Drive_Stop(BLMotor);
}

void Bot_Rotatetest(void){
   Drive_Reverse(FLMotor,DutyCycleRot);
   Drive_Forwards(BRMotor,DutyCycleRot);
   Drive_Stop(FRMotor);
   Drive_Stop(BLMotor);
}
```

State Machine Design: Plankton HSM

Top Level: Switch State

The hierarchical state machine design for plankton consisted of 4 state machines within 3 hierarchies. The initial state machine began with 2 states corresponding to on and off switch events. This level allowed for hardware checks like ensuring that all tape sensors and the beacon detector were properly functioning and calibrated accordingly without having to change the software. This helped immensely with all “preflight checks” conducted during checkoffs and during the competition. Switching on and off would cause all substates to be reinitialized, thus causing all state machines to begin at the beginning. While in the SwitchOn state, the substate, ToIFZ would run.

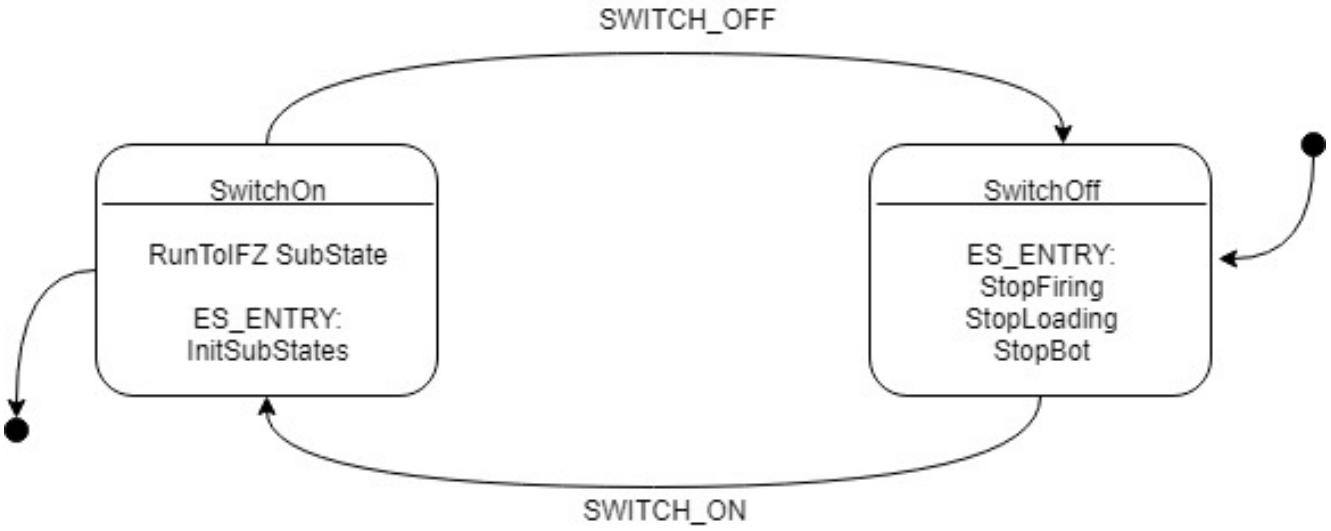


Figure 19: Top Level HSM

To IFZ Substate:

Upon entering the the ToIFZ state machine, the bot would begin with locating the enemy in order to orientate itself towards the front line. Upon properly orientation, the bot would reverse with a timer in order to ensure that it is within bounds before moving forward and aligning itself upon the front tape. While attempting to align itself on the front tape the bot would rotate until it reaches a point where all six tape sensors are aligned on the tape. The bot's rotation depended upon the ON_XTAPE event and the corresponding far tape sensor mask. When all tape sensors are aligned on the tape the bot would then strafe to the left to reach the corner. Originally the right corner was selected, but after several failed attempts to properly detect the right corner, the bot was directed to the left where a higher success rate occurred. This first corner event proved to be very difficult with the final bot design as the tape sensors seemed to either not detect a corner appropriately and/or detected an OFF_XTAPE event prior to a corner event. However, once all of the tape sensors were ALL_ON_TAPE, the state would transition into the move state, where the first third level hierarchy was entered.

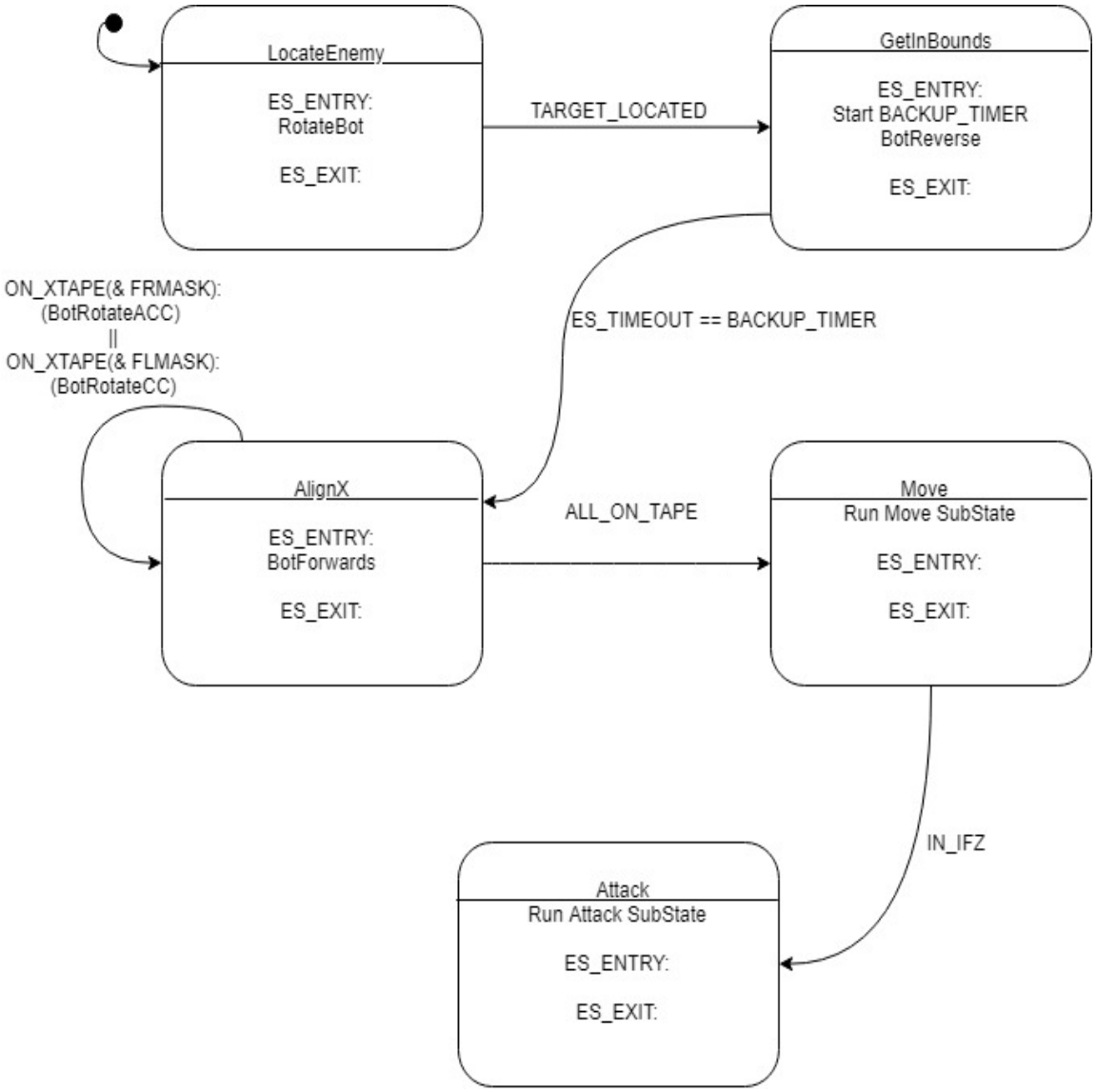


Figure 20: To IFZ SubState

Move Substate:

The first third level hierarchical state is the Move SubState. The Move SubState consists of all of the proper alignments required for the bot to reach the IFZ before transitioning into its attack phase. The bot enters the Left state where alignments are attached in order to keep the bot on the X tape sensors while approaching the front left corner with an X corner event. In order to increase the chances of catching corner event, a timer was added to the Align reverse state, where if timed out the bot would transition into alignRight and when getting an ON_YTAPE event, the bot would transition into Reverse. This was implemented after noticing several iterations where the bot would not properly transition but instead reverse while in an align state. This greatly increased the proper transition into reverse and continuation of proper states. However, a proper transition of the forward alignment to reverse could not be determined, so about a 90% successful corner transition ensued.The transition from the Left state to the AvoidCorner State following an X_Corner event was utilized to prevent a double corner event from occuring with the transition straight from Left to Reverse as noted in previous trials. However, even after utilizing this state, the bot would still occasionally detect an ON_Y_CORNER event when no event should have occurred and was believed to be related to sensitivity of the tape sensors. In order to properly transition between alignments and bump states, a previous direction would be saved when transitioning out of a major state such as Left, AvoidCorner, or Reverse. An additional bump counter was utilized to determine the proper direction to strafe in order to avoid obstacles. Upon finally hitting another Y_CORNER event while reversing to the IFZ, the bot would then transition this event to an IN_IFZ event in order to advance to the next state in state machine above.

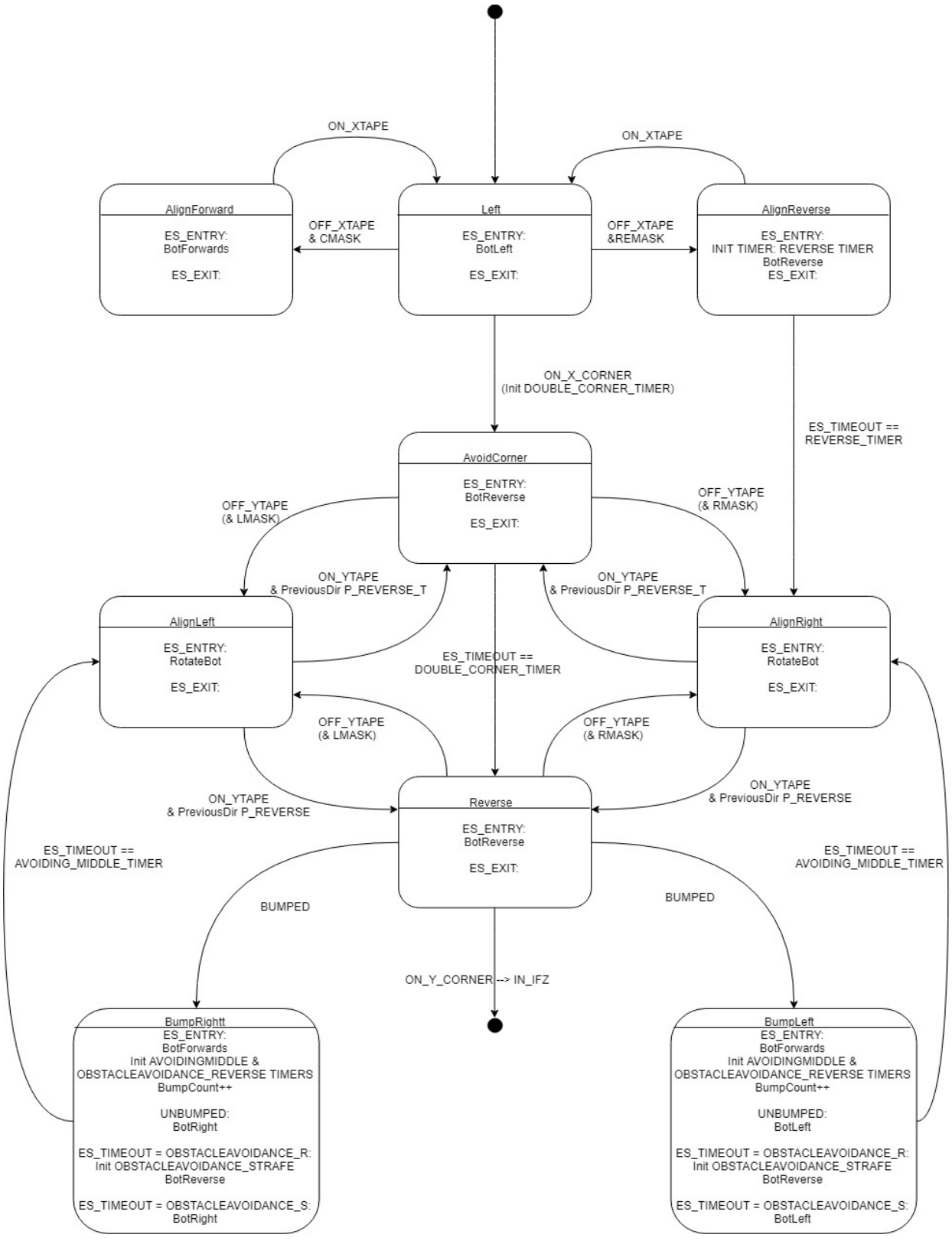


Figure 21: To IFZ

Attack Substate:

The second, third level hierarchical state machine consisted of the attack phase of the bot. This state machine would be entered upon an IN_IFZ event, where the bot has detected an ON_Y_CORNER event. This state machine consisted of the bot aligning itself on the Rear tape line and then continuously strafing across the rear tape in the IFZ while searching for the enemy bot. Utilizing the mecanum wheels to search for the opposing bot in IFZ helped prevent the bot from detecting its own beacon by rotating and possibly bouncing the signal off a nearby while or object. Strafing would also attempt to save time from rotating around to find the beacon and prevent the opposing enemy from detecting our own beacon while being mobile. While strafing the bot would align itself along the X axis of the bot in order to maintain within bounds and utilize ON_Y_CORNER events to switch between strafing movements. Upon detecting the enemy, the bot would then stop and initialize the firing motors. There was concern of running all 7 motors simultaneously, the 2 firing motors, loading motor, and the 4 drive motors, so that is why the bot was stopped. A timer allowed the motors to get up to speed in order to create the vacuum needed to suction ping pong balls into the chamber and to fire the balls an appropriated distance. Upon the warming up of the firing mechanism, the loading mechanism began which consisted of 2 states, a forward loading and reverse loading. The forward loading would help lower balls into the chamber and consisted of a 3 second timer while the reverse state would help prevent jams and push any balls towards the barrel in order to be suctioned in. The firing mechanism would run for a total of a minute and a half in order to help preserve battery and by that time most if not all balls would be fired.

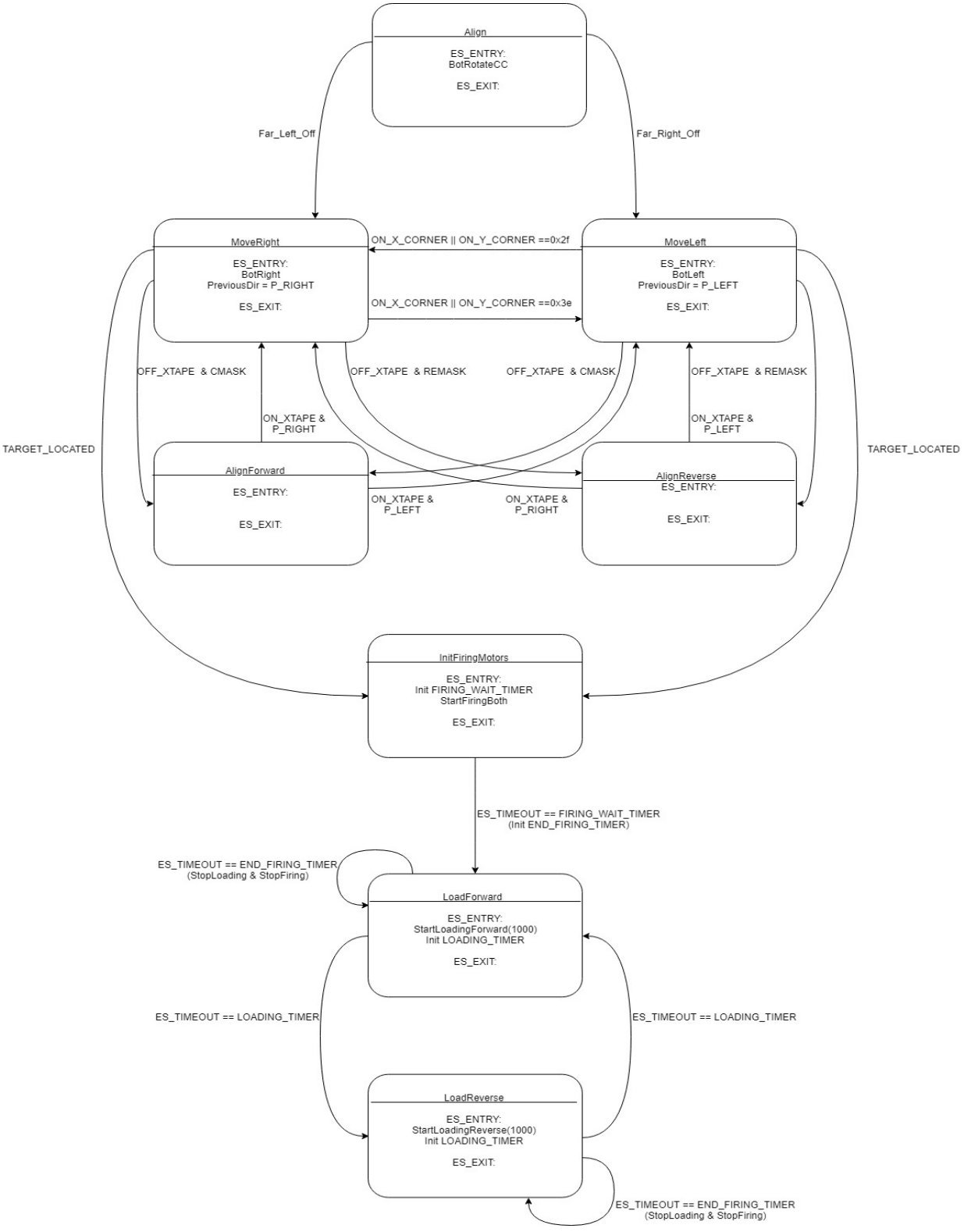

Align
ES_ENTRY:
BotRotateCC
ES_EXIT:
Far_Left_Off
Far_Right_Off
MoveRight
ES_ENTRY:
BotRight
PreviousDir = P_RIGHT
ES_EXIT:
ON_X_CORNER || ON_Y_CORNER ==0x2f
ON_X_CORNER || ON_Y_CORNER ==0x3e
MoveLeft
ES_ENTRY:
BotLeft
PreviousDir = P_LEFT
ES_EXIT:
OFF_XTAPE & CMASK
OFF_XTAPE & REMASK
OFF_XTAPE & CMASK
OFF_XTAPE & REMASK
ON_XTAPE & P_RIGHT
ON_XTAPE & P_LEFT
TARGET_LOCATED
TARGET_LOCATED
AlignForward
ES_ENTRY:
ES_EXIT:
ON_XTAPE & P_LEFT
ON_XTAPE & P_RIGHT
AlignReverse
ES_ENTRY:
ES_EXIT:
InitFiringMotors
ES_ENTRY:
Init FIRING_WAIT_TIMER
StartFiringBoth
ES_EXIT:
ES_TIMEOUT == FIRING_WAIT_TIMER
(Init END_FIRING_TIMER)
ES_TIMEOUT == END_FIRING_TIMER
(StopLoading & StopFiring)
LoadForward
ES_ENTRY:
StartLoadingForward(1000)
Init LOADING_TIMER
ES_EXIT:
ES_TIMEOUT == LOADING_TIMER
ES_TIMEOUT == LOADING_TIMER
LoadReverse
ES_ENTRY:
StartLoadingReverse(1000)
Init LOADING_TIMER
ES_EXIT:
ES_TIMEOUT == END_FIRING_TIMER
(StopLoading & StopFiring)

Figure 22:

Launcher Design

Figure 23: Barrel of Firing Mechanism

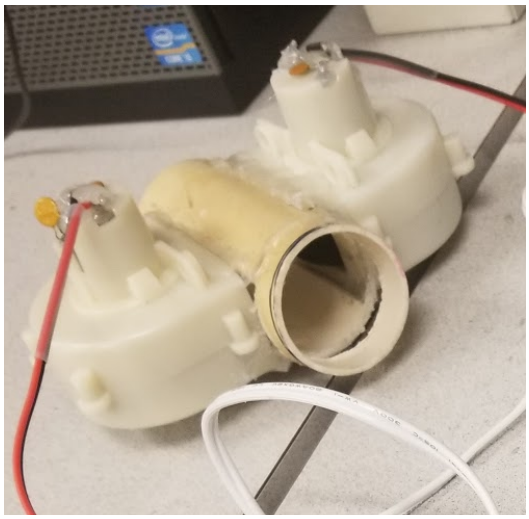


Figure 24: Firing design

The firing mechanism utilized consisted of two motors placed vertically. the top motor ran on a 5V power supply and the bottom with a 3.3V power supply. This combination created a vacuum that could suck in a ball about 4 inches away. The balls would be dropped 3 inches away from the chamber of the shooter mechanism. The vacuum provided the necessary suction to bring balls into the barrel while also providing the arc and backspin to hit the target from the IFZ. Before the final design, several iterations of prototypes with varying motors and wheels were attempted. Originally, stiff wheels were first attempted, which worked, but did not provide enough distance when it would shoot a ping pong ball. The stiff wheels were then switched with rubber wheels, allowing for more traction in order to get more grip on the balls while still allowing them to pass through the barrel and not lose too much velocity. In order to operate the firing mechanism, the two motors were connected via a driver, with each motor connected to 2 inputs in order to sink up to 1.2A of current. These motors were activated through a simple signal through a single pin that was then connected to all 4 inputs. This was then placed on top of the acrylic covering the power distribution boards, centered on the bot and as far forward as possible.

Hopper Design

Figure 29: Hopper

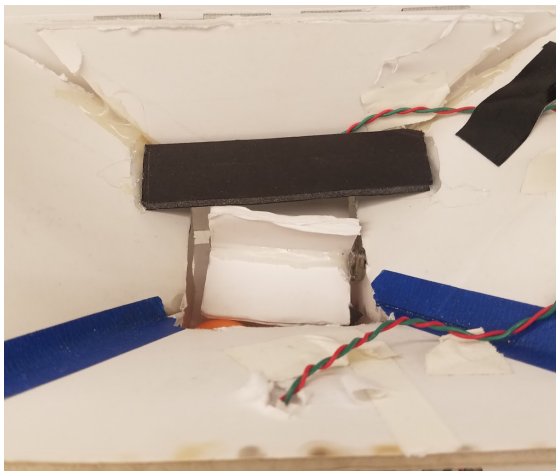


Figure 30: Inner Water Wheel

Gravity was decided as the action force to lower the balls from the hopper to the chamber of the shooting mechanism. The design had a funnel like design consisting of a small opening at the bottom and centered to allow one ball to go through at a time. The hopper design was chosen since many balls can be stored due to its volume. The loading mechanism consisted of a hopper large enough to accommodate around 18 balls and a motor with a water wheel design that would fluctuate between forward and rear motions in order to prevent jams from occurring with balls prior to reaching the barrel. An H-bridge was utilized along with the loading mechanism in order to allow for the change in directions along with a PWM duty cycle being sent. However in the end, the duty cycle was not necessary as the maximum was constantly sent to the H-Bridge.

Challenges

Vibrations

The use of the mecanum wheels lead to vibrations with the wheels moving in different directions depending on the movement of the bot. In the beginning there was no problem with these vibrations, but as time progressed issues started to become noticeable with some connections coming loose. This was an easy fix, for the final design connections would be soldered or glued together. While this resolved about 90% of the issues of wiring, there were still a couple of odd connections that did not seem to be functioning despite working to emplace the connections in the most robust method possible. Towards the end, an issue with a connection of the Right and Rear Tape Sensor were found where the tape sensors would flicker on and off with vibrations or wire manipulation. In order to resolve this, the wiring was glued, and taped in several locations. This helped prevent flickering on and off, but would still require occasional readjustments and constant close monitoring to ensure that the tape sensor would function properly when the bot was turned on. If the bot were to continue further, the tape sensors and wiring would have to be replaced entirely with a new set to produce better and more consistent results.

Cabling

For the project a plethora of components were needed for the design implemented thus leading to numerous wires being needed to route all components to the UNO, power, and other connections. The use of mecanum wheels for the design required the use of 4 wheels and a motor for each wheel. The firing mechanism then had two motors: the bottom motor running at 3.3 Volts and the top motor running at 5 Volts. Another motor was needed for the Hopper to stir the balls allowing balls to flow into the firing mechanism and to prevent jams within the funnel and Hopper. Thus, in total 7 motors were utilized, but not all of them were run at the same time. The limitation of motors operating simultaneously was done in order to prevent a fuse from blowing by drawing to much current and placing too much strain on the UNO power distribution board. In

addition to the motors, were 6 tape sensors for navigation. In the beginning of the project, wiring was a mess and hard to keep track of. As this was a prototype, used to ensure that the design concept was plausible, wiring was lower on the priority list to complete. For the final design the issue of wiring was resolved by adding ribbon cable under the robot to reduce the wire messiness. The ribbon cable was connected to four PCBs that were attached to the tape sensor base. These cable connected to the V, W, X, and Y ports of the UNO thus reducing the cables on the top of the bottom and into the UNO. With the ribbon cabling the checking of connections was simpler as checks were made by checking the bottom and the connection to the component itself.

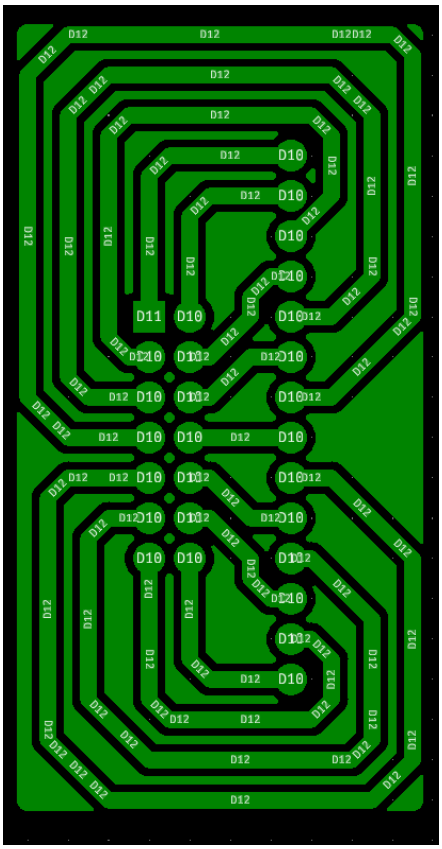


Figure 31: Routing Board Design

Beacon Detector

Several issues with the beacon detector were identified through throughout the entirety of the project. The placement of the beacon detector, shielding and the design of the board. Initially the beacon detector lacked the ability to detect the beacon in the fabrication lab past the first obstacle. The range of detection was increased through replacing the transresistive resistor value from 670Ω to 5,000Ω. This greatly improved the ability of the beacon detector to pick up the beacon from the beginning of the field to the IFZ. Several tests with varying resistors was conducted, but found the best solution at 5k, without over-amplifying noise and still reaching the desired range. The beacon detector was placed directly above the firing mechanism at the highest possible level. as this would be aiming the firing mechanism. With the photodiode placed in line with the firing mechanism and at the highest point of the bot, shielding was required to help mechanically filter out nearby noise and to narrow the signal from the beacon detector. The mechanical shielding utilized for the beacon detector consisted of the body of a flashlight in combination with tape at the entrance of the flashlight body. The flashlight body helped reduce reflections while the tape helped narrow the range of detection, improving accuracy. The culmination of all these additions and fixes lead to a beacon detector that would detect the beacon if it was aiming straight at the beacon which the bot get a better shot at the target.

Mecanum Wheels

The choice of mecanum wheels was intended to be a challenge and learning experience working with a different and unique driving mechanism. The wheels and motors were ordered immediately upon deciding to incorporate the wheels into the design to first overcome this obstacle. While the software for the wheels was straightforward with the codes described and exhibited in the design portion, the issues faced with the wheels were mostly contained to the mechanical and electrical aspects. Initially the motors were to be mounted on the top of the first base, but with attempting to mount the wheels to the first prototype base and motor mounts, the wheels were not low enough to raise the base off the ground. Instead of utilizing laser cut motor mounts, glue, and screws, 4

U-bolts strapped the motors to the bottom of the base thus ensuring proper lifting of the base and securing the motor thoroughly. Another mechanical issue encountered was connecting the wheel to the shaft of the motor. The motor shaft consisted of a D shaft, so a laser cut D-Key of acrylic was cut to match the inner hexagon shape of the wheel. Glueing several acrylic keys together did create a firm key, but the key would not stay attached to both the wheel and the shaft causing the wheel to constantly detach. In order to resolve this issue, a larger key of the circular portion of the wheel was then utilized in combination with 2 screws to ensure consistent wheel and motor performance. These 2 mechanical issues were identified within the first 24 hours of receiving the wheels and resolved within the following couple of days. A problem identified approximately a week into utilizing the wheels was with the initial H-Bridges utilized, the DRV-8814. 2 of these bridges were required with the 4 wheel drive. However, a regulator was broken off of one of the bridges while the other bridge had issues with its direction input signal. These bridges were then swapped for L298N H-Bridges. These h-bridges differed from the originally used bridges where the bridges only required 2 direction pins in combination with the jumper pin on the enable input. The 2 direction pin inputs would cause the bridge to drive a motor forwards or reverse when the pins are offset 1-0 or 0-1. In order to reduce completely modifying the code designed earlier, the duty cycle was standardized to 500, 50% on and 50% off in order to have consistent speeds at all wheels in all directions.

Shooting Design

After the final iteration of the droid's ping pong launcher was completed it was decided that it would be beneficial to tilt the launcher by some small angle in order to gain additional range as well as increase the probability of firing over the near obstacle. Hitting obstacles was an issue from the first iteration of the launcher with no angle and using stiff wheels. Since a launcher and ammunition holder was designed to maximize the number of ping pong balls and be gravity-fed, the actual launcher was placed below the reserve and as such was only about four inches from the ground. This close proximity to the ground, in conjunction with the pressing need to meet the deadlines resulted in raising the barrel by the maximum angle while still maintaining the droids designated measurements within the box of compliance. In addition to this to

modification of tilting the wheels upward, making the shooting mechanism vertical with a backspin led to always hitting the target with a full holder.

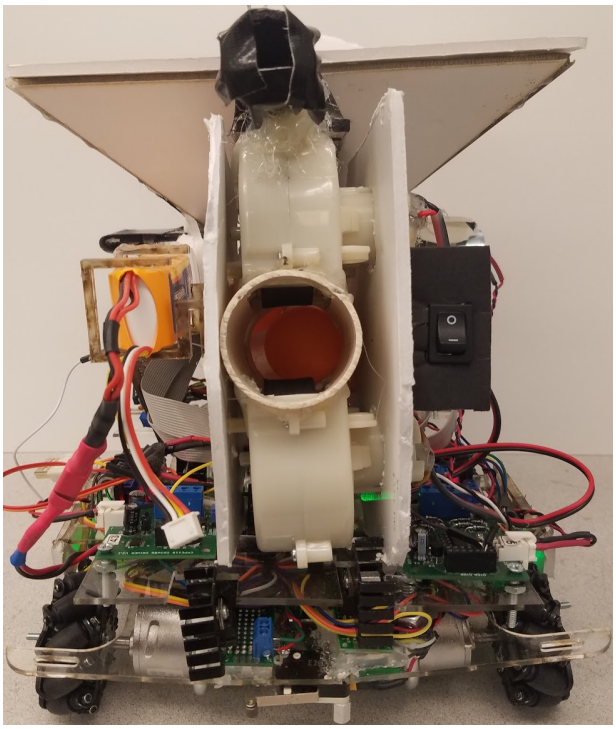


Figure 32: Inclined barrel of Firing

Overheating of voltage regulators

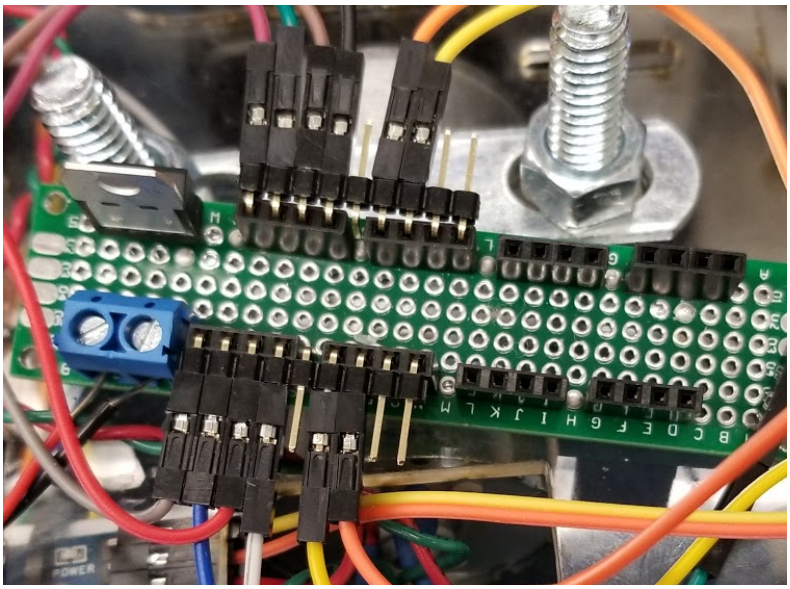


Figure 33.1: Singular design of power distribution boards (3.3V Regulator)

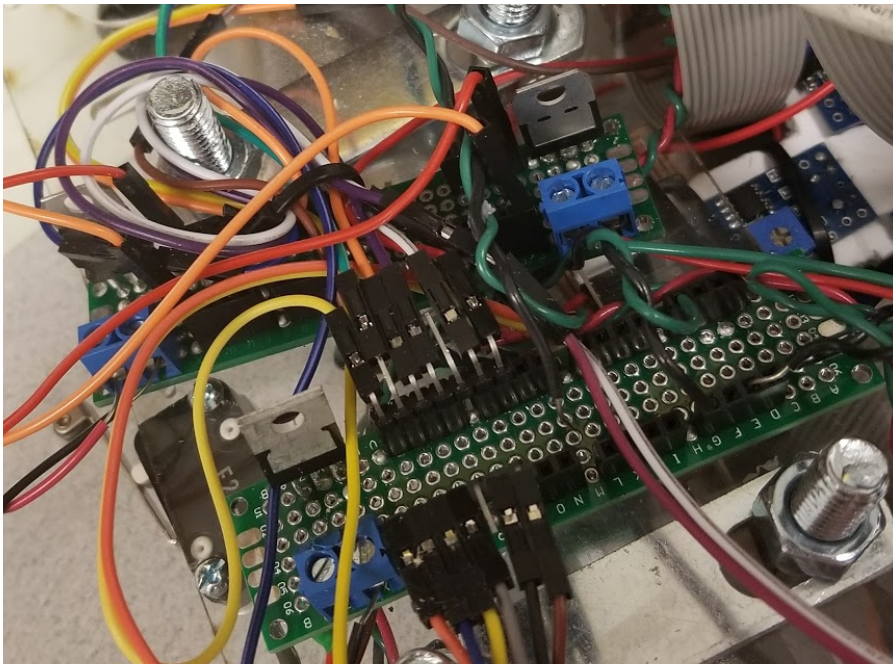


Figure 33.2: Power distribution boards for 5V (Left) and 3.3V(Right) Regulators

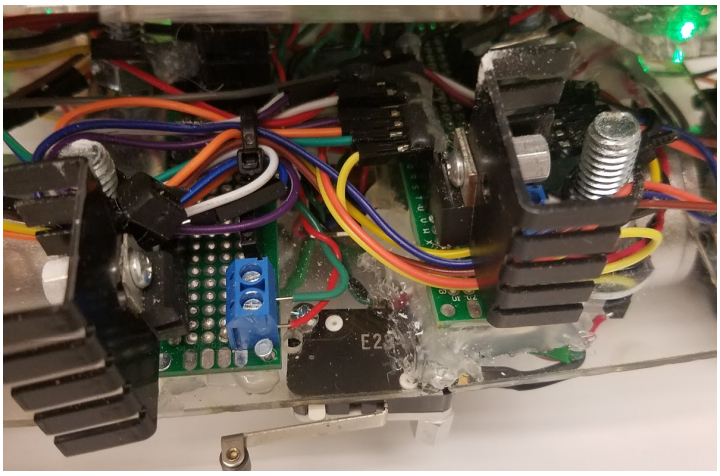


Figure 33.3: Heatsinks on 5V (Left) and 3.3V(Right) Regulators

3 voltage regulators were utilized for 3 power distribution boards, 1 3.3 Volt board and 2 5V boards. The 3.3V board housed the power and ground for all of the tape sensors, the ground for the switch, and the power and ground for the lower firing mechanism. The 2 5V boards were created to house a single motor each of the firing mechanism, before the bottom motor was connected to the 3.3V board. When testing with both motors on a 5V board each, the current drawn would cause the regulators to generate much heat within the first minute causing concern for overheating and blowing a regulator. In order to prevent this possible drawback, large heatsinks were added to each regulator to help

disperse the heat quickly and thoroughly before damaging the regulator or motors. When the bottom motor was transferred over to the 3.3V board, another large heatsink was incorporated as a safety precaution prior to even testing. Especially since the 3.3V board was linked to so many components, being cautious and adding the heatsink may have saved tape sensors and the regulator from being destroyed or damaged with so many components attached to the same power board.

Tape Sensors

A total of 6 tape sensors were decided to be utilized to generate the required tape events. The tape sensors were decided to be placed with 4 in the center of the bot with in a 2 inch box and the far left and far right being at least 2 inches away from these center sensors. This placement for the prototype required several iterations of testing and designing for the bot to properly detect all tape events. For the final design, the 4 inner tape sensors were all moved closer together to being within 1.5” away and the extremities were at about 2.5” away from the nearest sensor. Along with 6 tape sensors comes the requirement for 3 wires each, totaling 18 wires, 6 to power, 6 to ground and 6 to the UNO. This mass of wires during the prototype phase led to a mess of wiring that had to be resolved with the transition to the final platform. In order to handle the wiring better, ribbon cabling was utilized and 5 wiring cavities were added to the final platform. During the transition to the final platform, one of the tape sensors was improperly connected to ground and power, twice consecutively. This caused two tape sensors to become inoperable and careful connections with a requirement of 2 sets of eyes to confirm proper connections were made before powering on the bot.

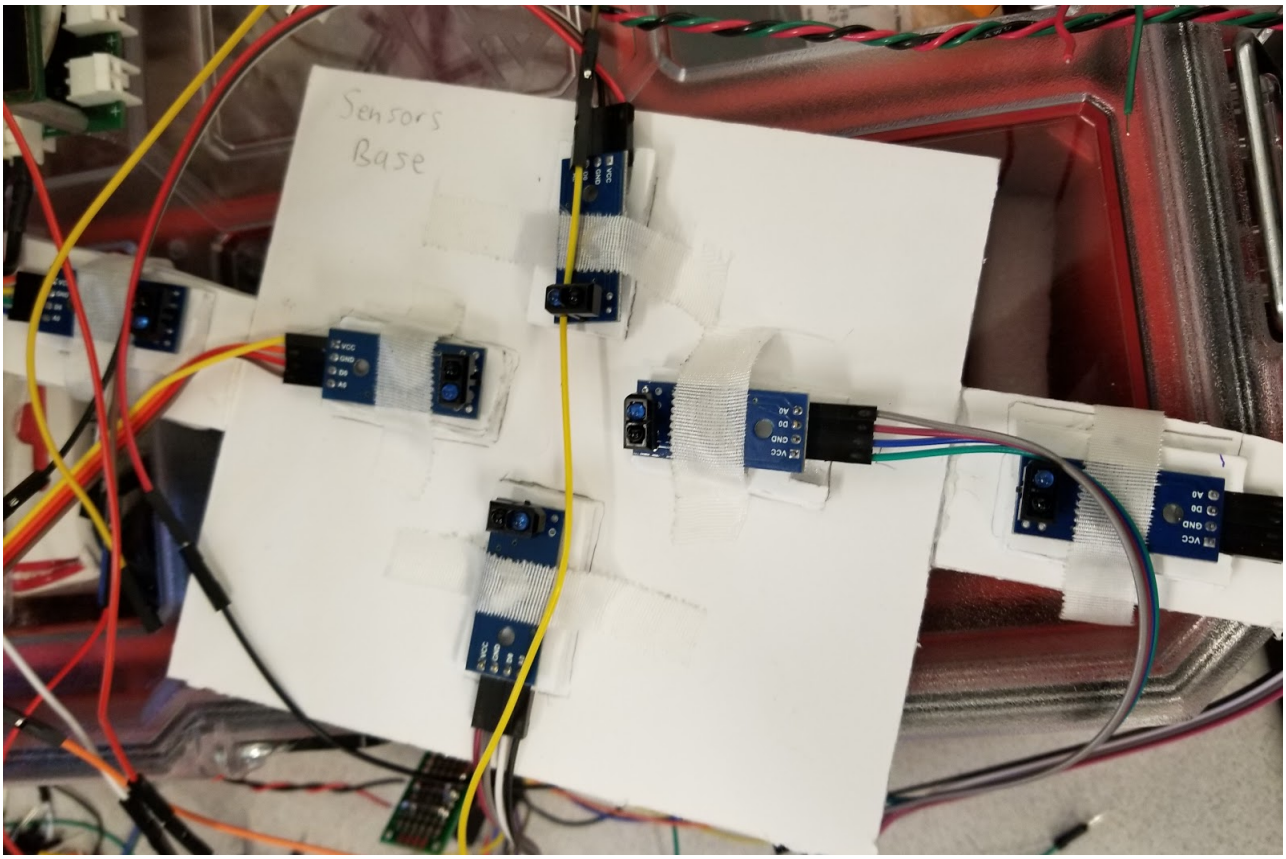


Figure 34: Tape sensor configuration

Successes

Board Placement

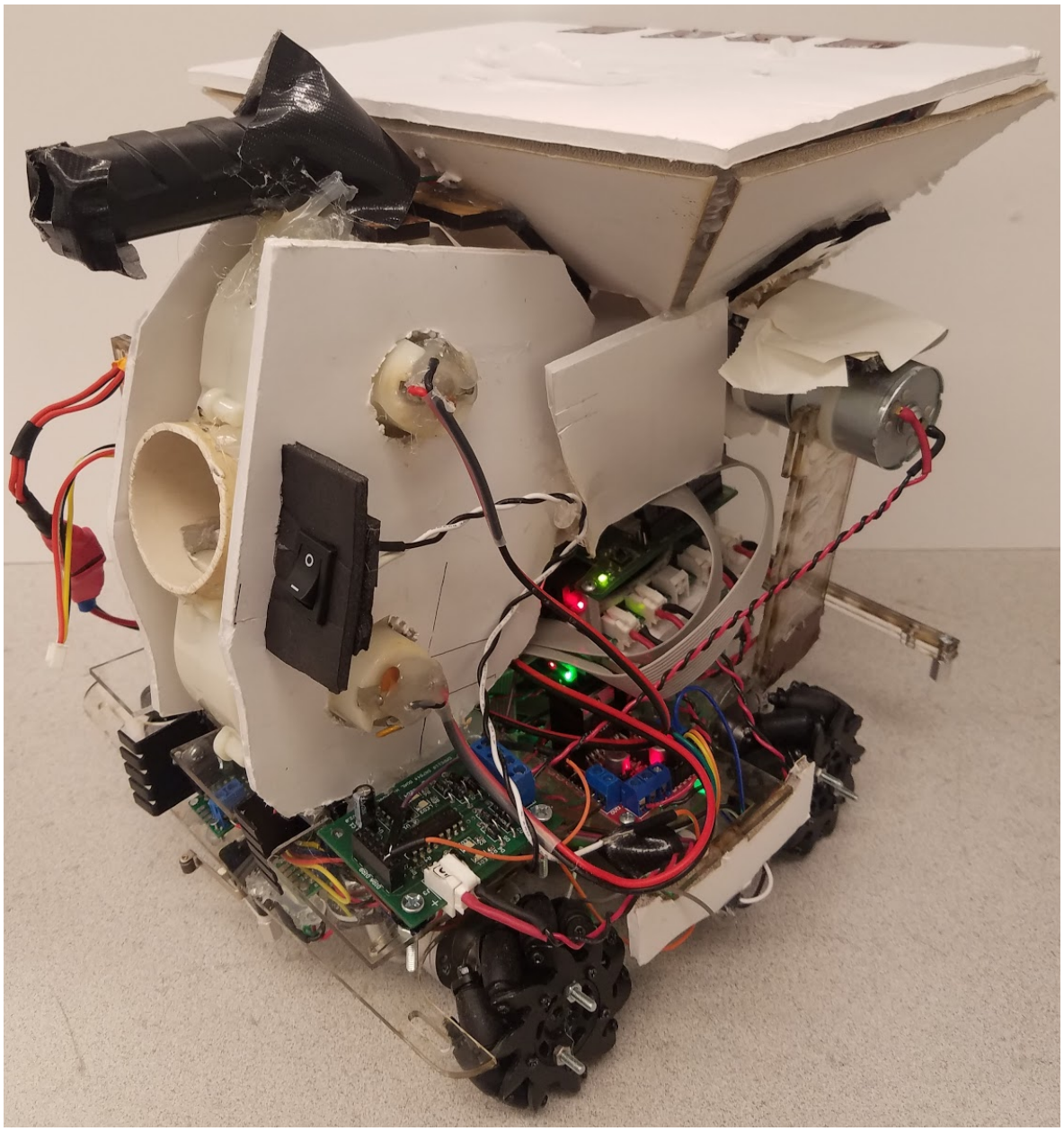


Figure 35: Bot

With the prototype board being cut from MDF, making adjustments to the board with saws and drills was easy enough. This allowed for quick adjustments and testing of

different models before finalizing the final design for cutting. In the end the base platform housed the motors, mounted by U-bolts on the bottom, 5 wiring cavities, the UNO stack mounted on the rear protruding bolts, the tape sensors mounted below on foamcore with the 4 PCBs connecting to the UNO via ribbon cable. The base board then had small holes for the adjustment of tape sensors as necessary as well. Overall, all components were fairly easily accessible for adjustment, replacement or analysis.

Mecanum Wheels

After being thoroughly advised against the implementation of mecanum wheels, the wheels were up and running within the first week, before the termination deadline. The wheels were more of a challenge mechanically with properly attaching the wheel to the motor and then attaching the motor to the base, but after several design ideas and trials, this was achieved with U-Bolts attaching the motor to the base platform and cutting circular D-Keys that had screw implacements to attach to the wheels. Overall, the wheels performed as desired and provided extra mobility with the ability to strafe which had potential to save more time with avoiding the requirement to turn at obstacles or while in the IFZ.

Timely Check Offs

Throughout the entirety of the project, all check offs were met on time despite having encountered several large setbacks and challenges. This quarter nearly all of the team or all of the teams successfully met each check off on time and ended well with having achieved min-spec check off prior to the competition.

Ribbon Cable

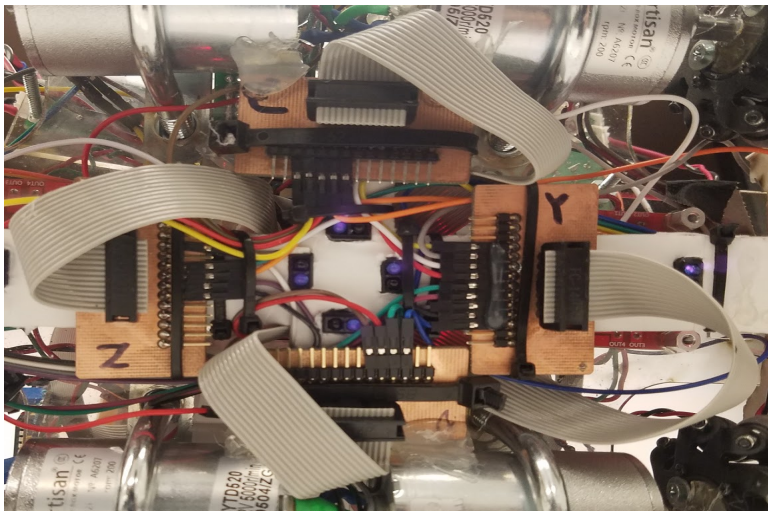


Figure 36.1: Underbelly of bot consisting of ribbon cable, tape sensors, motors and mounts bottom view

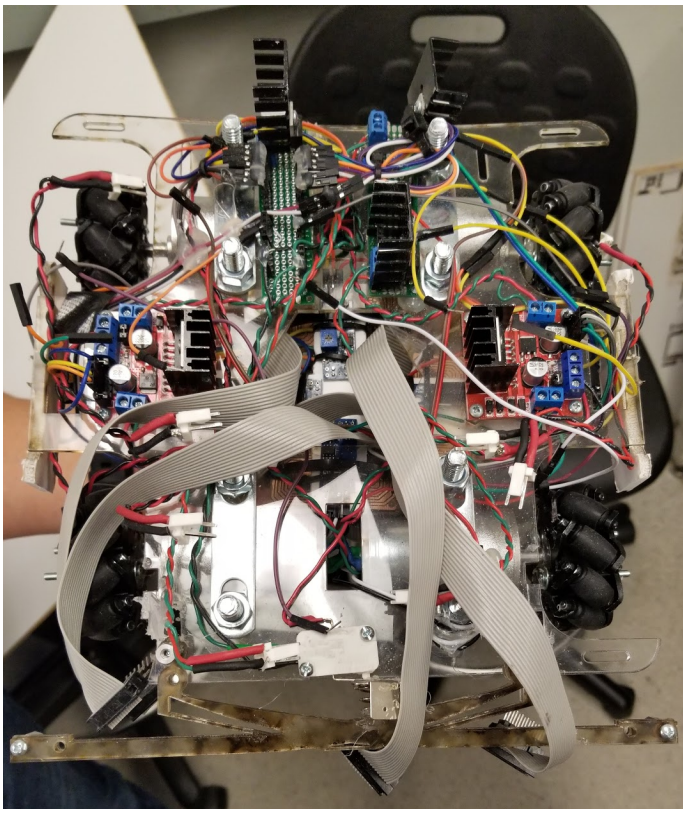


Figure 36.2: Top view

In order to resolve the mess of wiring that was encountered during the prototype phase, ribbon cabling was implemented. The ribbon cabling greatly reduced the amount of wiring that was seen on the top level of the board as only wiring for power was generally visual. All UNO connections, except for the PWM signal to the loading mechanism were connected to the base of the bot to the 4 PCBs mounted to the foamcore tape sensor board. Wire cleanliness helped reduce the possibility of shorting and blowing a fuse on the UNO which did happen during the prototype phase several times, but was only encountered once during the final stage.

Strong Beacon Detection

The beacon detector did generate plenty of problems, but in the end the beacon detector functioned well at all ranges from the front of the field to the IFZ. After several iterations of testing with

Conclusion

As one of the few teams that attempted the use of multi directional wheels we were the only team that were bold enough to go against the teaching team's advice and attempt the Mecanum Wheel design. We definitely jumped through significantly more hoops than we needed. Even though we made it work in the end, we had to scrap many cool ideas such as having a gatling gun. For all future teams, it is advisable to implement feedback control systems for the velocity of each wheel and to make sure all connections are secure by gluing them since by using omni wheels it cases the bot to have many vibrations. In the end, the uniqueness of the bot led to a multitude of learning opportunities in design, implementation and testing in all aspects, mechanical, electrical and software.