UC SANTA CRUZ

UNIVERSITY of California, Santa Cruz

Electrical and Computer Engineering

Introduction to Mechatronics LABORATORY

ECE118

The Spy Who Slimed Me (Final Project)

STUDENT ID :
1676781 trhotran
1481947 jfortner
1658336 lurmerca

Contents

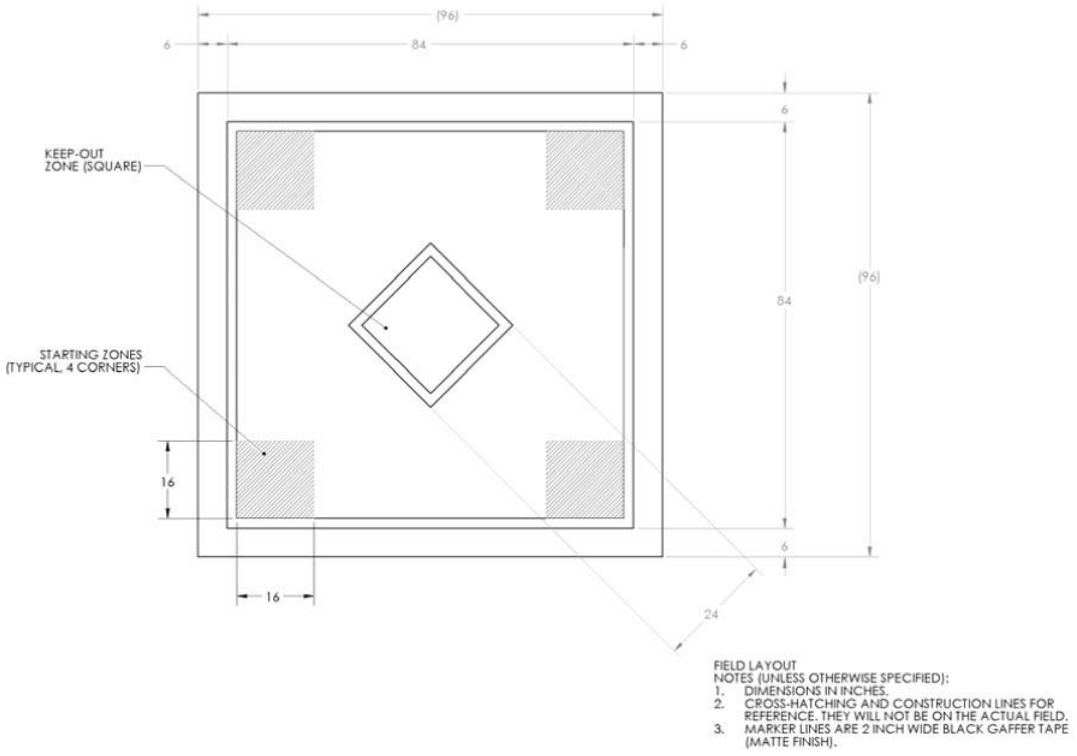
1 ABSTRACT

The threat of SALT (Serial Annihilation of Living Things) is at an all-time high. We have just received recent intel, that they have constructed a deadly weapon.

Our mission is to disable the weapon. However, the area where the weapon is being stored in has heavy levels of radiation. In order to avoid radiation, we must build a droid that will sneak in the facility and disable the weapon.

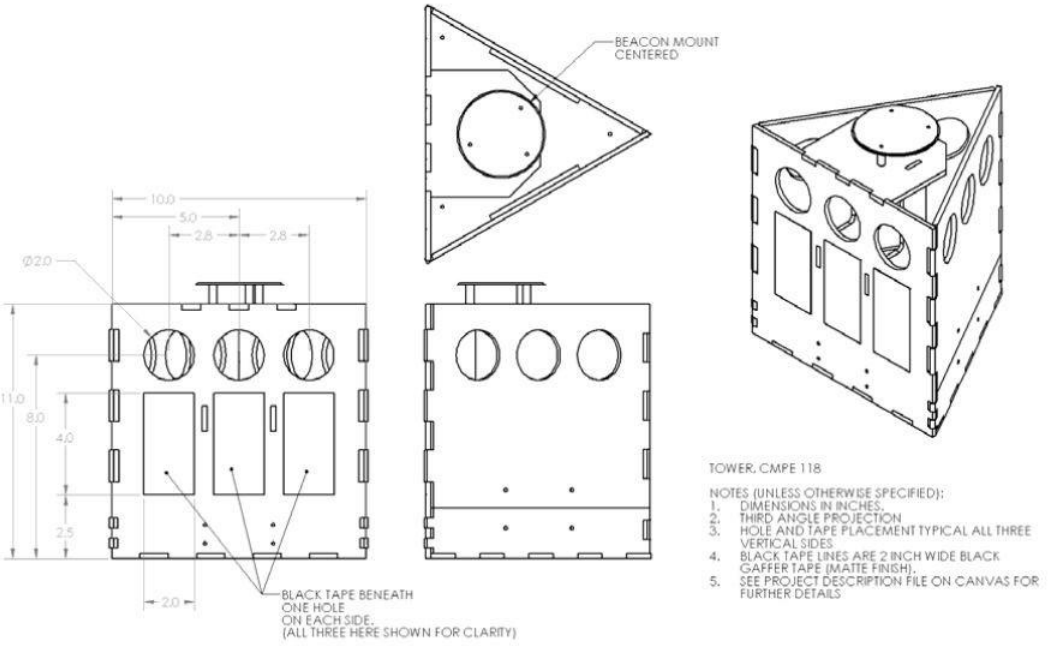
2 INTRODUCTION

The Task at hand for the ECE 118 Mechatronics final project is to create an autonomous robot that can navigate a field to a beacon tower, locate the correct side of the beacon tower, locate the correct hole of that side of the tower and deposit a ball into the hole. This task must be completed twice under two minutes the robot can no longer than an 11”x 11”x 11” cube. The field is an 8’ x 8’ square outlined by black tape with a 2’x 2’ restricted diamond in the center



Field of play

The beacon tower is 10” x 11” triangular shape from the plan view.

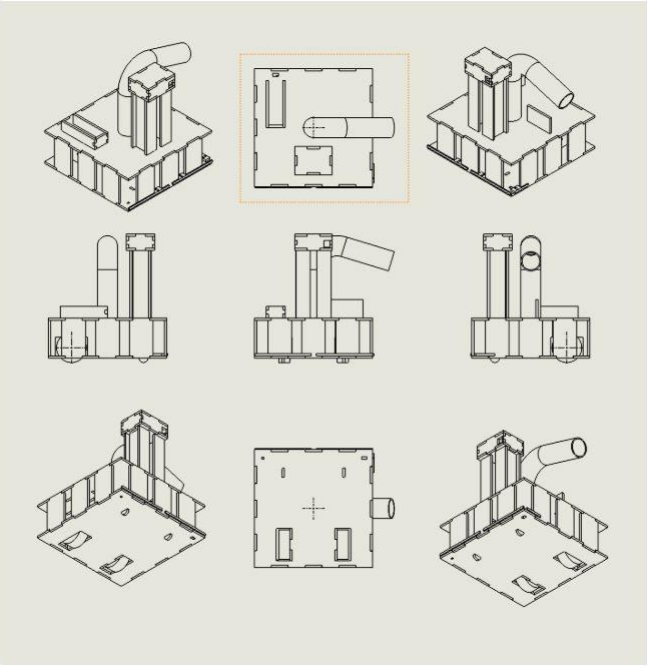


Beacon Tower

The correct side of the tower is indicated by a track wire around the perimeter of the inside of the tower wall. The correct hole is indicated by black tape that is directly underneath of the hole. The steps we took to solve the problem, as well as issues, we encountered are outlined in the sections below.

3 Mechanical

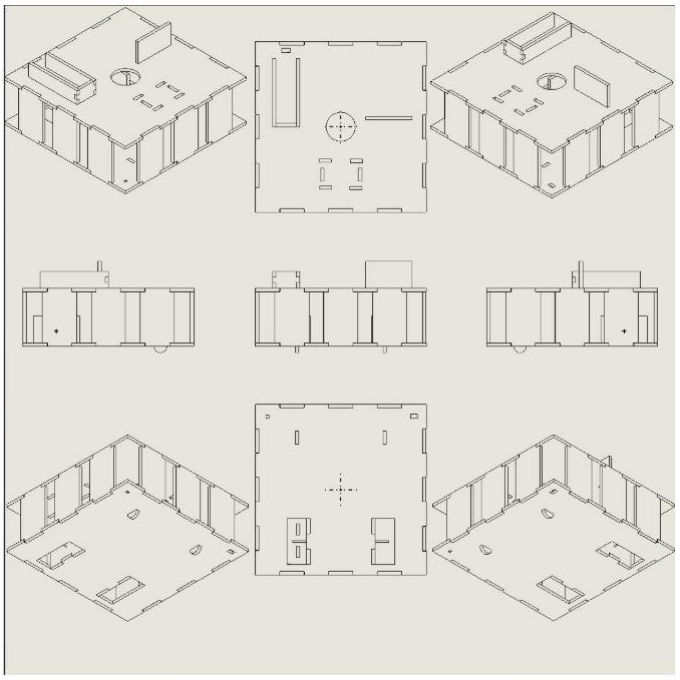
3.1 Overview



CAD design of Robot

The mechanical design laid out above features a square base and 2nd level, that is separated by a 3-inch walls circumnavigating the square perimeter. Skids are placed at the front of the body to keep the bot level, while two rear mounted wheels drive the robot. Two front and side bumper are present on the bottom level of the robot. Two tape sensors are also on the bottom base, one in the front right corner and the other one the left front corner. The beacon detector sits atop a beacon tower, and inside the beacon box. The elevator sits in the center of the second level. The elevator has a ramp that is hanging just off the side of the robot, this is done to get as close to the hole as possible. The elevator, itself, is being raised and lower by another motor. Two track wires are placed on the side of the second level of the robot. One track wire is placed on the front, and the other track wire is placed in the rear of the robot. The last tape sensor is also on the left side, in line with the elevator to the signal the correct hole. The last box on the top level secures in the battery. Overall, this design has led to a sturdy, modular, and ultimately successful robot.

3.2 Body



CAD design of Robot Base

The body is constructed from laser cut MDF. The two floors both measure to be 10x10”. This specific length was chosen due to the beacon towers also being 10” in length. Thus, when mounting the two track wire detectors at the front and the rear of the robot, the correct side can only be found if both track wires are triggered. The levels are supported by 3-inch tall walls. 3 can be used per side, however not all are necessary. On sides where maximum access is needed, no walls can be used. While on the other walls where stability is key, all three walls can be added. Thus, adding to the modularity and adaptability of the robot.

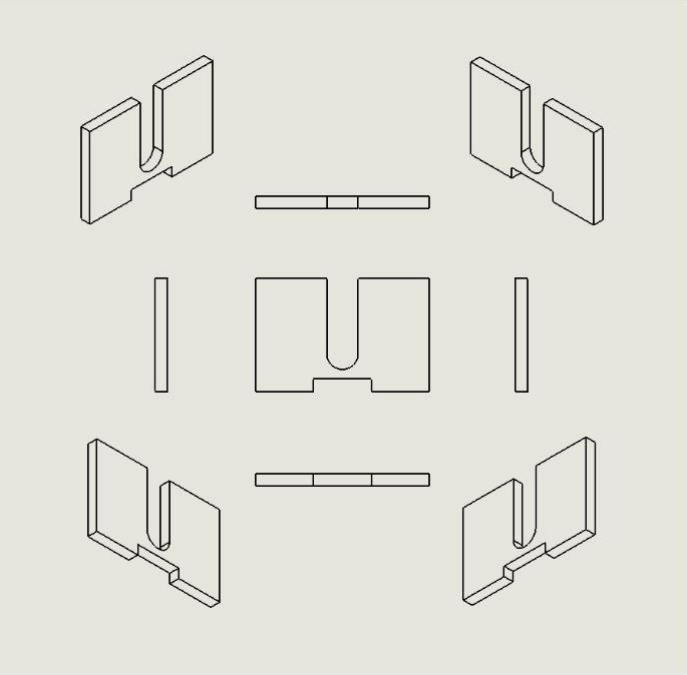
The bottom base is supported by the rear mounted wheels and by the front mounted acrylic skids. This combination allows for a large amount of mobility, stability, and traction over the center or front mounted wheels due to the even distribution of weight. The bottom base is also home to the UNO board, Hbridge, DS3568 Driver, two front bumpers, one side bumper, and two front mounted tape sensors (one per front corner of the robot). With the UNO board being the heaviest thing on the robot, keeping it as low as possible and it being centered mounted allows for maximum stability. Unfortunately, this makes it hard to access. This is why the walls were designed modular as described in the paragraph above.

The motor drive boards are mounted near the rear on the lower base. This is to keep them close to the motors and away from the sensors (to keep away any unwanted noise). The front bumpers are being used to find the beacon tower, while the side bumper is being used to traverse around the beacon tower. These are mounted on the first level to keep the point of impact lower to the ground and on a secure base. The tape sensors are mounted below the base to keep them as close to the ground as possible. Thus, giving the highest accuracy for the robot.

The second base is supported by the walls and is held three inches above the base. Leaving room for all the boards on the first level to fit comfortably. On the second level, there is a ping pong elevator, ping pong elevator winder, beacon tower, track wire detectors, and a tape sensor. The track sensors are mounted 10 inches apart, and they are as far as possible from the motors to ensure there is minimum noise from the motors being picked up by the track wire detectors. The tape sensor is mounted in the center of the ping pong elevator. This is so that when the tape is detected, it will move for a short time longer and stop to deposit the ball. Lastly, the beacon tower is on the second level. This will all be in more detail in a following section.

If we were to redesign, we would add in laser cut holes for the board mounting, and cable management. Rather than using a drill to machine cut holes as needed throughout the project. For this specific implementation however, the rapid prototyping flexibility the drill provided was the highest priority.

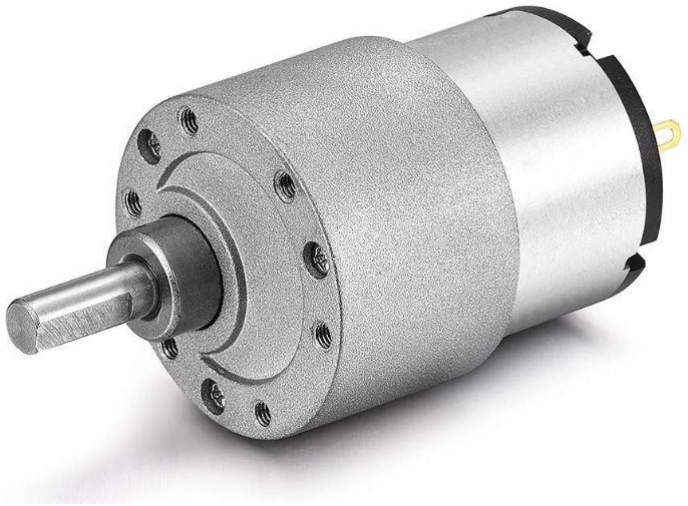
3.3 Motor Mounts



CAD design of Motor Mounts

There are three motor mounts being used throughout the robot. Two are on the lower level of the robot, and the final on is on the second level. All three motor mounts function the same way. The motor shaft is slotted into the center cutout of the motor mount. The motor is rotated around the center axis in order to adjust the height of the motor shaft, which is placed off center of the motor. Holes are then drilled through the wood and screws were placed into the motors to secure it at the correct position. All three motors were placed with the shaft in the lower most position and secured with M5 screws.

3.4 Motors



Motors

All three motors are identical, thus only one image will be needed to represent all three. The two bottom motors are both 200 RPM, while the elevator motor is 60 RPM.

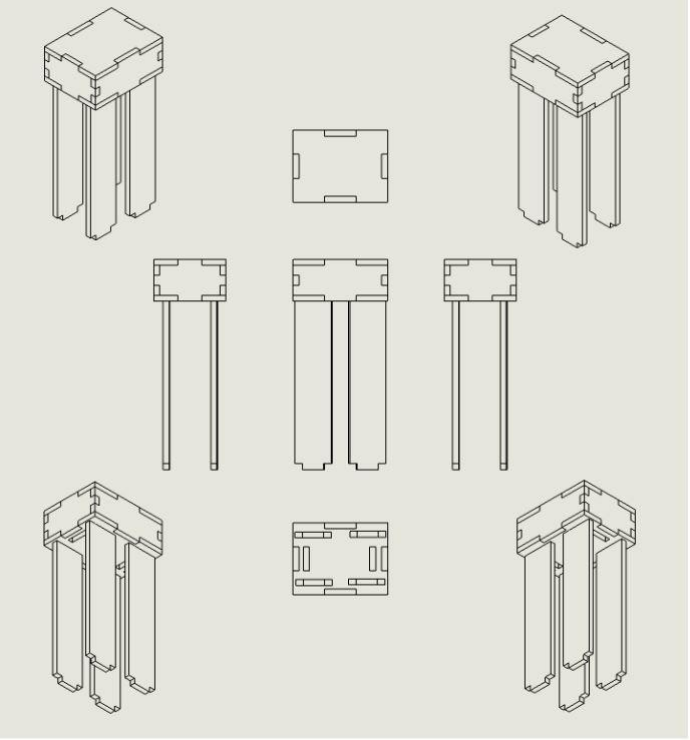
Originally, we chose to use 3 high torque (60 RPM) motors, as we were worry that the weight of the robot may hinder the performance of the higher RPM motors. Upon testing, it was concluded that the weight would not affect the higher RPM motors. This conclusion was not realized until approximately halfway through the build, we noticed that the robot was not moving fast enough to meet the two minute time limit. The decision was then made to switch to the 200 RPM motors, which gave us a much faster robot. While the motors cannot be driven under a 50% duty cycle PWM wave, the overall top speed was greatly improved. This is much more important for this application than the powerful 60 RPM motors. As for the third motor that winds, and unwinds, the elevator, we stayed with the 60 RPM motor. The lower speed allows for a more accurate wind. Ensuring the we drop only one ball into each hole each time. While the high torque allows for the motor to overcome the frictional forces present by the tape along the ping pong tube.

3.5 Wheels

Wheels

We chose to use 64mm diameter scooter wheels. This diameter was chosen as it was the smallest diameter that kept the robot at an appropriate height for the tape sensors and produced the smallest footprint on the robot itself. The rubber material of the scooter wheel provides enough traction for our robot to maintain predictable movement without slippage.

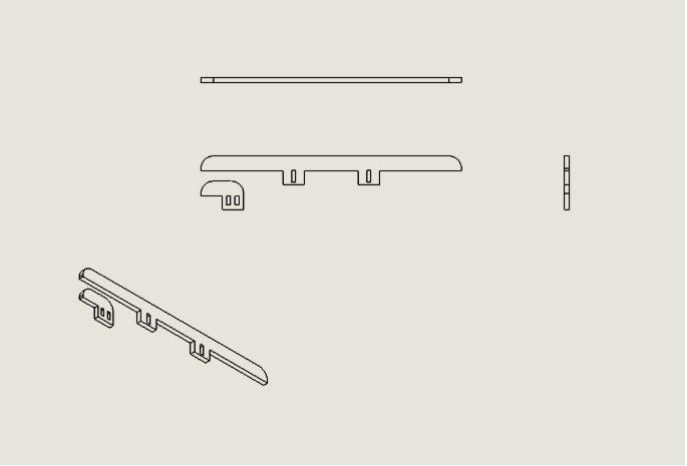
3.6 Beacon Tower



CAD design of Beacon Tower

The beacon tower lifts the beacon detector off the second level, 5.5 inches, to bring the beacon detector to ten inches. This is to level the beacon tower and gives the maximum amount of range for the sensor to be able to detect the beacon signal coming from the towers. The beacon is housed within the beacon box. This MDF box mechanically shields the sensor from as much noise as possible, by surrounding the sensors fully expect for a sensor size cutout in the front.

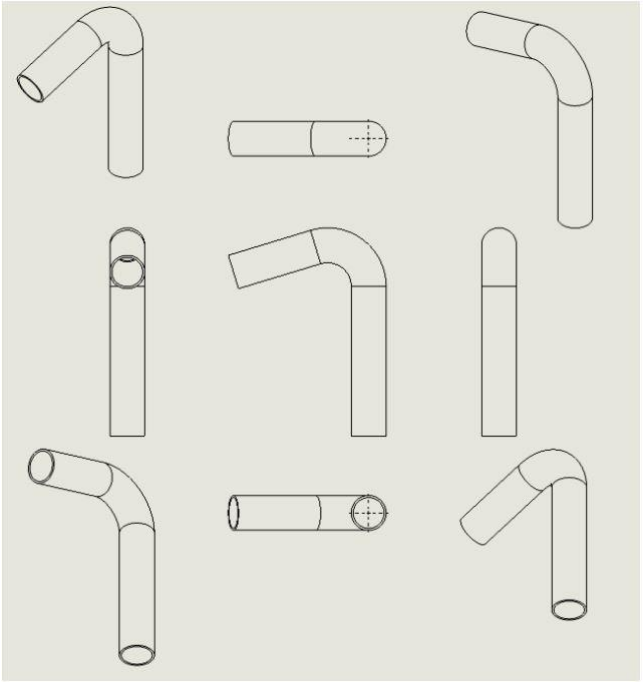
3.7 Bumpers



CAD design of Bumpers

The acrylic bumpers alter the robot for a collision. One is placed along the front of the robot, the longer one of the two, with a bumper at either end of the robot. The smaller bumper is used as a side bumper, and it is placed on the left side of the robot, near the front. The side bumper is used to bounce around the side of the beacon tower and to remain parallel. This ensures parallelism, which is necessary for the track wires and the tape sensor to work correctly and to keep the ball dropper directly adjacent to the hole.

3.8 Ball Dropper



CAD design of Ball Dropper

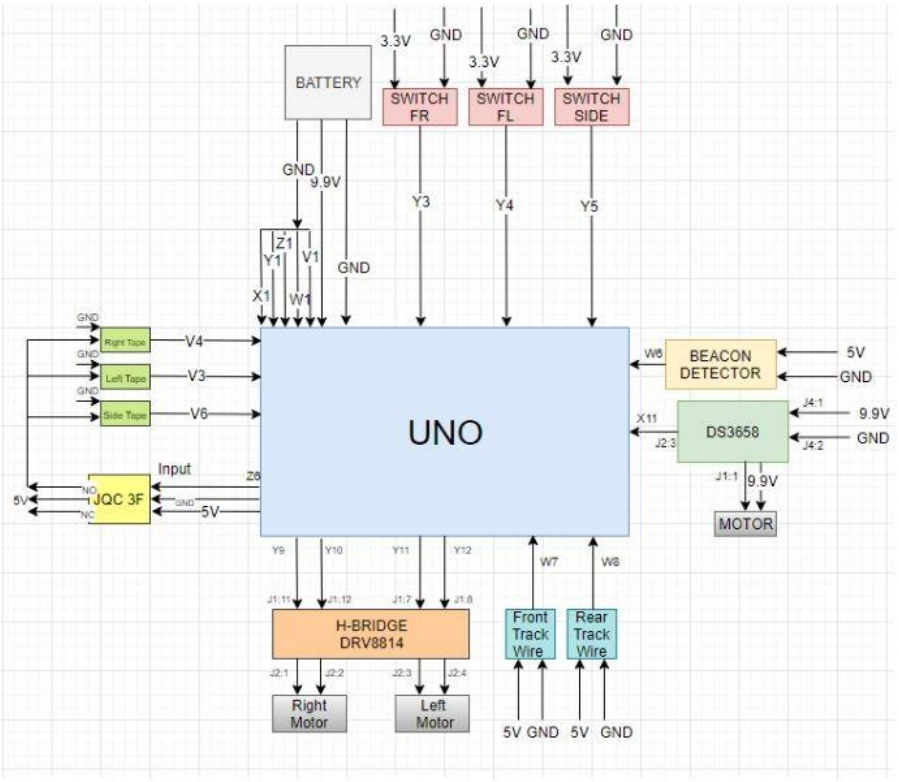
The PVC elevator ball dropper is centered mounted on the robot. This was just to make forward and backward movements along the wall symmetric and easier to program. The inside diameter is slightly larger than the diameter of the ping pong ball, his is to allow the elevator floor room to raise and lower. The tower along stands seven inches tall and is capable of holding five ping pong balls. While it was not necessary to have extra balls, it was preferable to have extra balls in case the robot was to malfunction and require a retry. The elevator floor is raised and lowered by the winding and unwinding of a string by the motor on the second base of the robot.

4 Electrical

4.1 overview

In order to accomplish the task at hand, the full system requires many electrical components to work together. These components include a beacon detector, two track wire detectors, three tape sensors with a power relay, DC motors

with control boards, a power distribution board. All these components were connected and controlled by an UNO 32 board. The Full electrical block diagram is pictured below

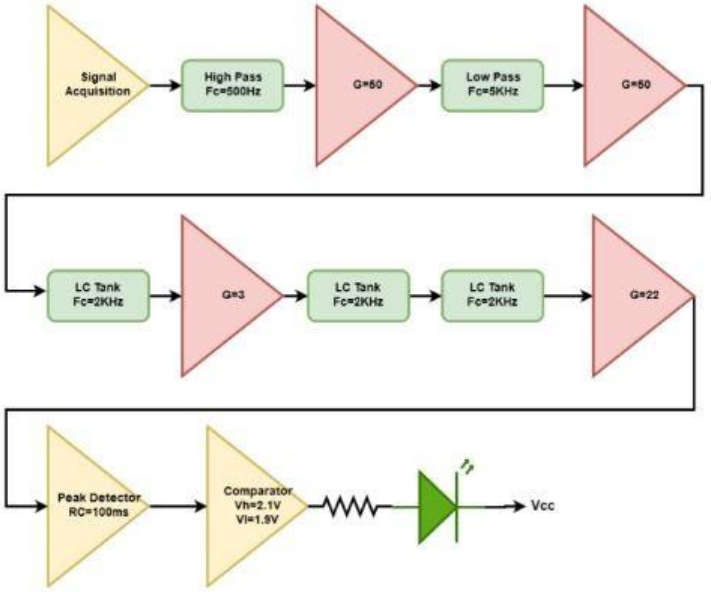


Block diagram of UNO 32 connection

4.2 Beacon Detector

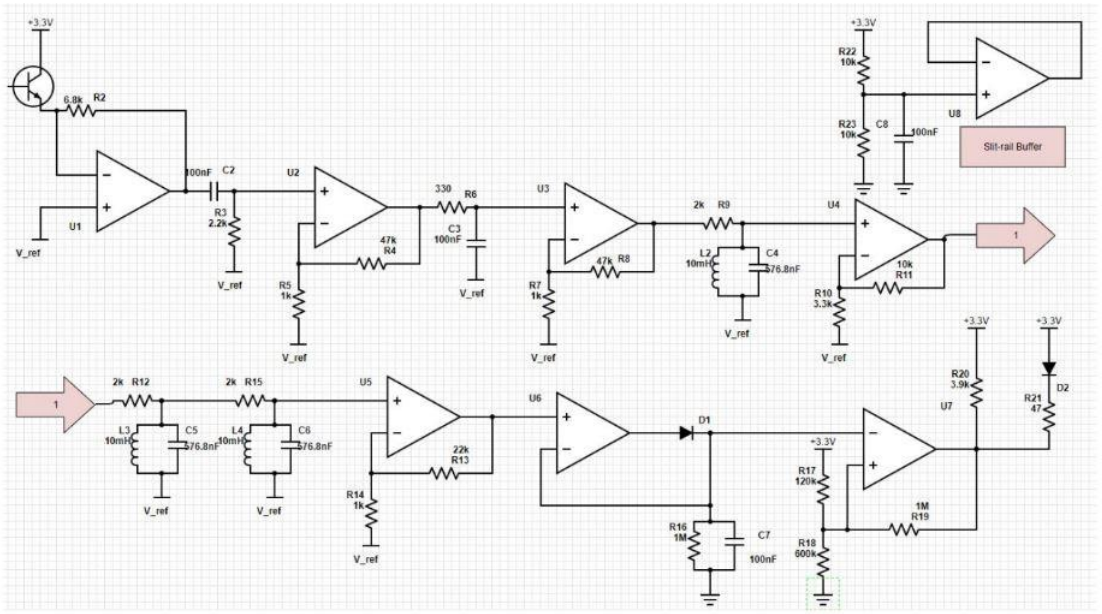
For our design, we used all passive amplifiers. This was important, since our filters are simple. At each stage we could instantly tell if the filter was working properly, or not. The signal acquisition phase was a trans-resistive amplifier with a gain of ten thousand. Next, the signal is sent through a RC high pass filter, with a corner frequency of 500Hz. This is mostly intended to remove any DC components of the signal, as well as the 60Hz noise coming from the lights. Next, the signal goes through heavy amplification, followed by a low pass filter; The low pass filter used to attenuate high frequency noise. The initial amplification stages are meant to convert the signal coming from the phototransistor, into a square wave with the same frequency. This was done to have the filter stages work with the same initial

input signal, regardless of distance from the beacon. The filter input at ten feet is exactly the same as the input at six inches.



Flow diagram of Beacon Detector circuit

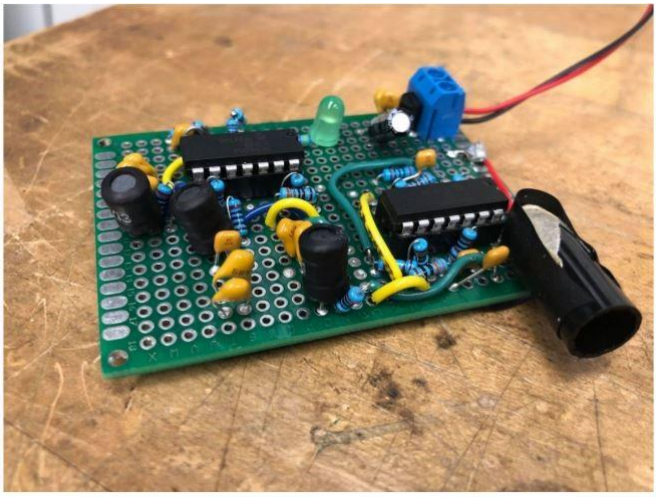
To filter the signal, we used a RLC bandpass filter circuit. The simple circuit is easily able to turn into a second order bandpass filter with an adjustable Q factor. After each stage of filtering, we apply enough gain to get the 2KHz signal back to nearly railing amplitude. We used two LC filters in a row at one point to increase our comparator's hysteresis without adding an additional opamp. Finally, we put the signal through a peak detector and the comparator.

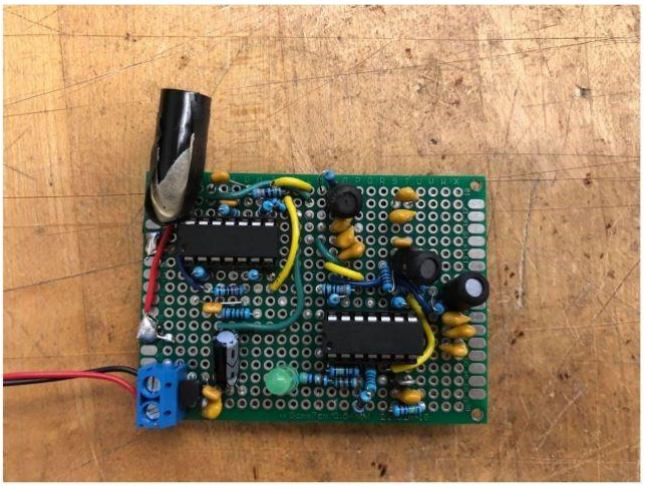


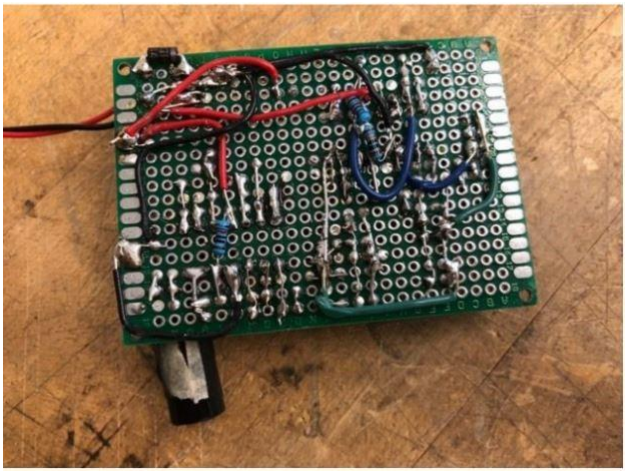
Beacon detector schematic

With this design, we can detect a signal from many feet away, we tested the range to be able to detect a signal from a distance of twelve feet away. The beacon detector uses only two IC (Integrated circuit) chips, both of them being the MCP6004 chips. All filter stages are passive. This design was more favorable to build, as it gave use more flexibility to tune, and troubleshooting. The following pictures are the final product of the beacon detector:

Beacon detector







We first built the beacon detector circuit on a breadboard. After testing, and verifying results, from the circuit on the breadboard, we replicated the same circuit onto a second breadboard. This was done to ensure there were not errors when building the circuit onto a perfboard. We started with the LDO, putting that in the upper right corner. After that was verified and tested, we built the trans-resisitive, high pass, low pass, and split rail. All of these go on the first IC chip. We built, and tested, all of those parts at once.

During testing, we noticed that we were picking up some strange noise in our signal. Upon testing, it was determined that the sensor was not sufficiently shielded. To fix the issue, we built a beacon box, and we covered the back end of the phototransistor. Following that, we moved onto the LC tank filters. At the end of the

tank filters, we learned that our hysteresis bounds needed some adjustments. We fixed the bounds by simply increasing the gain stages. This was done so that the signal would rail sooner, thus giving us a signal that fit fine with our hysteresis bounds. We then added another LC filter, cascading it directly from the output of the second LC filter. This was done to cause more attenuation, which we compensated for by increasing the gain on the next amplification stage.

Finally, we built the peak detector. We verified the magnitude of the 1.5KHZ and the 2.5KHz signals up close. We also tested the 2KHz signal from various distances.

This was the beacon detector that we soldered during Lab 2. Even though we did not change anything in the circuit, it did not function properly as it supposed to. We had to use the oscilloscope to observe the output of each stage to figure it out what happened. Based on these tests, we realized that the output from the peak detector had change and we had to make some changes to the previous comparator so that we could send the digital output to the UNO. Finally, we tested the whole circuit and added the LED, to signal when a signal is detected.

4.3 Track Wire Detector

In this application, the track wire is being used to identify the correct side of the beacon tower.

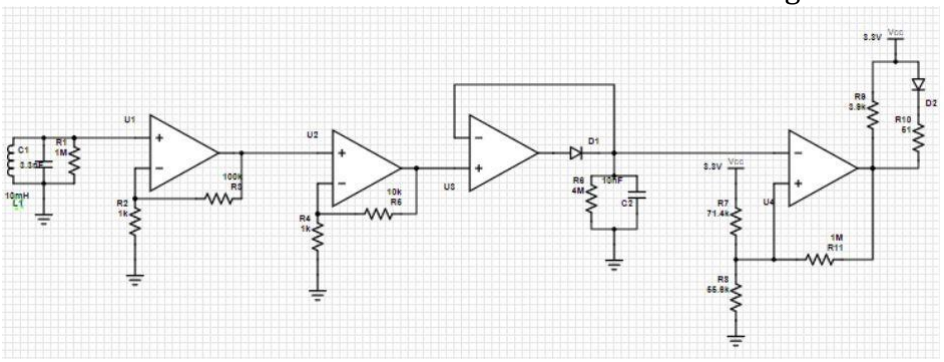
The first stage of many in our journey of converting a simple phototransistor into a usable signal that could power an LED involves a tank circuit. A tank circuit is a very simple LC oscillator that can oscillate at a resonant frequency close to 25KHz when exposed to the track wire. It serves as a test circuit in place of our phototransistor sensor by acting like an input sensor by sending an output signal in order to test the rest of our circuit, such as the peak detector or the comparator.

The non-inverting amplifier is the second stage of the circuit series and should be placed right after the tank circuit. This is a very useful circuit as it can serve as an amplifier, boosting the gain of any input signal to a voltage we desire (as long as we have the right gain stage set up). We want the track wire detector to be able to detect the signal from 3 inches away, so we added only two stages of amplification. The total gain we have from our track wire detector is thousand.

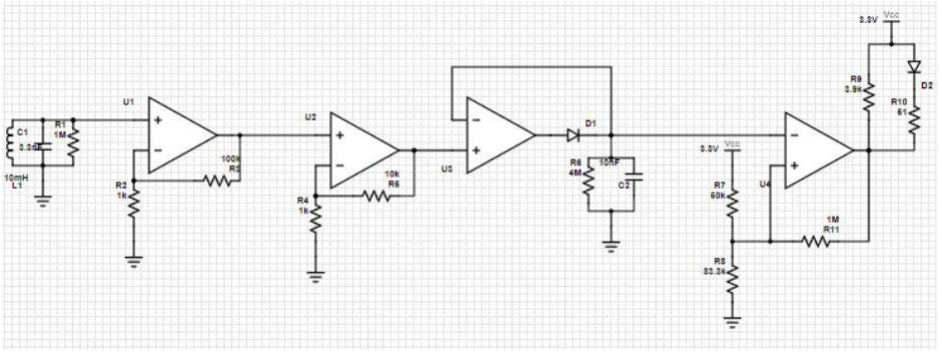
The peak detector is followed by the comparator. This circuit simply takes the constant value from the peak detector and generates a high or low, depending on the hysteresis bounds. Initially, we miscalculated the hysteresis bounds, because we were under the impression that we had to subtract before dividing when using the equation to determine the bounds for hysteresis. It took a while to realize as we tried to figure out why our comparator was signaling too early. We decided to use

three to four resistors for each each R value, to get as close to the calculated R value as possible. This proved to fix the bounds perfectly for us. Within the two track wire detectors, that we are using, each of them has different comparators.

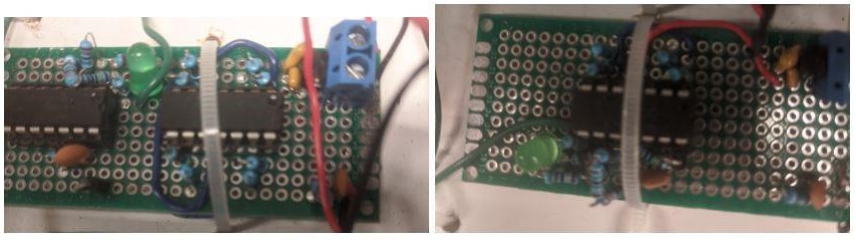
Finally, to be able to see the output from the whole circuit, we used a LED light. The LED was used to signal to us when a signal was detected from the track wire detectors. There is a resistor to limit the current flow through the LED.



First track wire schematic



Second track wire schematic

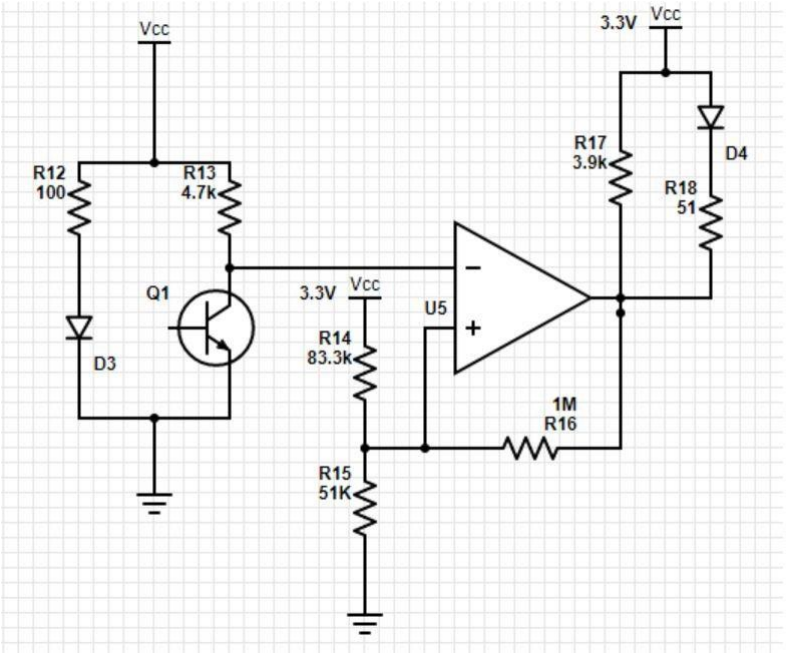


Track wire detectors

4.4 Tape Sensors

1.3cm We are using three tape sensors in total. Two of those sensors on the bottom of the robot, and the third one if located on the left side of the robot. All tape sensors

are powered with 3.3 V. We are using a 3.3 V voltage regulator on the perfboard. This is to protect the whole circuit. To limit the current flow through the diode, we used a 100Ω resistor and a 4.7kΩ resistor in parallel orientation for the phototransistor. The output coming from the phototransistor is a DC voltage, since we decided not to use a peak detector. The output from the collector is then fed into a comparator. We set the bounds so that the LED is on the tape, we have an ON TAPE event. Finally, to be able to see the output from the whole circuit, we used LEDs that light up every time the tape is detected. There is a resistor to limit the current flow through the LED. Even though we had the digital outputs available for all the track wire detectors, we used the analog outputs. We realized that the analog values would be more useful in term of finding the correct wall.

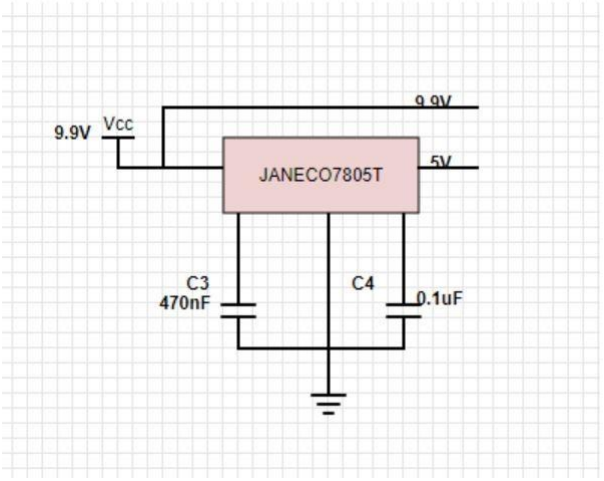


Tape sensors schematic

4.5 Power Distribution Board

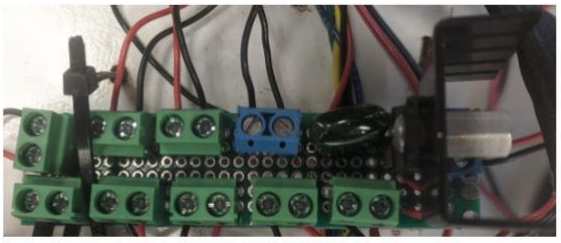
All of these electrical components require power; thus we created a power distribution board. Boards such as the H-bridge, UNO 32, DS3658 Driver, and the motor for the elevator all require a 10 V input. While the beacon detector, track wire detectors, and the all three tape sensors only require an input of 5 V, since they all have a voltage regulator of 3.3 V.

This is accomplished by powering the power distribution board when it takes in the 9.9 V from the battery and creates two power rails. One rail is 9.9 V, the second rail is 5 V, which is created by passing the 9.9 V through a 5 V voltage regulator.



Power distribution schematic

After several hours of use, it came to our attention that the 5 V voltage regulator would become hot. To fix the temperate problem, we added a heatsink to the 5 V voltage regulator. Adding the heatsink help tremendously and we were no longer seeing any adverse effect caused by the increase in temperature.



Power distribution

5 Software

5.1 Overview

With all the electrical and mechanical components working independently, the task for the software is now to get all of those components working together. To be able to get the robot to navigate through the field, avoid tape, locate one of the

beacon towers, drive towards the beacon tower (while still avoiding tape), circle around the tower to look for the track wire, locate the correct hole in the tower, deposit a ball, and find a new tower. This was done by creating multiple services, that would read any change from the robot, and by creating a hierarchy state machine. The language of the program is in C, and it was programmed using MPLab X, and the hex file was uploaded to the UNO 32 board. The program was also utilizing the ES Framework that was provided.

5.2 Services

To begin programming for the robot, the first step we took was to get the UNO 32 board to respond to services. Each service would have a preset event, and it would only post an event to the state machine if there was a change in events. With the bumpers being the easiest to connect to the UNO, the bumper service was the first file written.

5.2.1 Bumpers

```
ES_Event RunRobotBumper(ES_Event ThisEvent) {
    ES_Event ReturnEvent;
    ReturnEvent.EventType = ES_NO_EVENT; // assume no errors

    /********************************************
     in here you write your service code
     ********************************************/
    static ES_EventTyp_t lastEvent1 = NoFrontRightBump;
    static ES_EventTyp_t lastEvent2 = NoFrontLeftBump;
    static ES_EventTyp_t lastEvent3 = NoSideBump;

    ES_EventTyp_t curEvent1 = NoFrontRightBump;
    ES_EventTyp_t curEvent2 = NoFrontLeftBump;
    ES_EventTyp_t curEvent3 = NoSideBump;

    switch (ThisEvent.EventType) {
        case ES_INIT:
            // No hardware initialization or single time setups, those
            // go in the init function above.
            //
            // This section is used to reset service for some reason
            break;

        case ES_TIMERACTIVE:

        case ES_TIMERSTOPPED:
            break;

        case ES_TIMEOUT:
            if (Robot_ReadBumpers() & 0x02) {
                if (Robot_ReadBumpers() & 0x02) {
                    if (Robot_ReadBumpers() & 0x02) { // now we can count
                        curEvent1 = FrontRightBump;
                    }
                }
            } else if (Robot_ReadBumpers() & 0x01) {
                if (Robot_ReadBumpers() & 0x01) {
                    if (Robot_ReadBumpers() & 0x01) {
                        curEvent2 = FrontLeftBump;
                    }
                }
            } else if (Robot_ReadBumpers() & 0x04) {
                if (Robot_ReadBumpers() & 0x04) {
                    if (Robot_ReadBumpers() & 0x04) {
                        if (Robot_ReadBumpers() & 0x04) {
                            if (Robot_ReadBumpers() & 0x04) {
                                curEvent3 = SideBump;
                            }
                        }
                    }
                }
            }
            if (curEvent1 != lastEvent1) { // check for change from last time
                ReturnEvent.EventType = curEvent1;
                ReturnEvent.EventParam = (uint16_t) curEvent1;
                lastEvent1 = curEvent1;
                PostRobotHSM(ReturnEvent);
```

Bumper Service Code

The bumper service would read all bumpers using the ReadBumpers function. This function would return a four-bit number, then we would use the bit wise AND operation. Depending which AND operation would return a one, we would know which bumper is being pressed. This service would look for a change in events every five milliseconds.

The only problem that came from this service was the de-bouncing for the side bumper. Originally, we had the same amount of de-bouncing for the side bumper as the two front bumpers. However, during testing, the side bumper was showing to be more sensitive than the two front bumpers. As the ES TattleTail will show that it would randomly post a side bumper event when the side bumper was

not being pressed. To fix this bug, we added a longer de-bouncing for the side bumper.

5.2.2 Sensors

1.3cm The robot consisted of three different type of sensors, which are all stated in the electrical section. As for the software, the goal was to set bounds for the value being read from the UNO 32 board, and post an event if those bounds are crossed.

```
 * @author J. Edward Carryer, 2011.10.23 19:25 */
ES_Event RunBeacon(ES_Event ThisEvent)
{
    ES_Event ReturnEvent;
    ReturnEvent.EventType = ES_NO_EVENT; // assume no errors

    /********************************************
     in here you write your service code
     *******************************************/
    static ES_EventTyp_t lastEvent = No_Beacon_found;
    ES_EventTyp_t curEvent;
    uint16_t beacon_Sig; // read the beacon value
    beacon_Sig = AD_ReadADPin(AD_PORTW6);
    switch (ThisEvent.EventType) {
    case ES_INIT:
        // No hardware initialization or single time setups, those
        // go in the init function above.
        //
        // This section is used to reset service for some reason
        break;

    case ES_TIMEOUT:

//        printf("Beacon : %d\n", beacon_Sig);
        if(beacon_Sig < 300){
            curEvent = Beacon_found;
        }
        else{
            curEvent = No_Beacon_found;
        }
        if(curEvent != lastEvent){
            ReturnEvent.EventType = curEvent;
            lastEvent = curEvent;
            PostRobotHSM(ReturnEvent);
        }
        ES_Timer_InitTimer(BEACON_TIMER, TIMER_4_TICKS);
```

Beacon Service Code

The first sensor we took on was the beacon detector. The beacon detector sensors were set to give a digital reading, since we were not worried about the distance of the beacon.

The beacon detector had an output wire that was connected to a pin on the UNO board. Using the AD functions, that were provided to us, we would simply read the value being read by that certain pin. The bounds for the beacon detector were set by displaying the values being read by UNO on screen. Then we would see what value were being read when there was no beacon, and when there was a beacon. Again, the reading was set to a digital reading, so the bounds was 1023 for off, and 300 (or lower) for on. There were little to no problems with the beacon detector code.

The second type of sensors we worked on was the tape sensors. For the tape sensors, it was mostly similar to the beacon detector code. The only main difference was that we decided to use analog for the front right tape sensor. The reason why we decided to go with analog for only this sensor was that we were experiencing problems with it. We originally had the sensor giving a digital reading, but it would read black tape when there was no tape. We believe that the reason behind this was that the robot's weight may have not been evenly distributed. This could have the right tape sensor to be closer to the ground, and the shadow cast by the robot may have cause it to see tape. Instead of taking the sensor out and manually fixing it, we decided to fix it by giving an analog reading. Then the bounds were adjusted to only post a tape reading event when it passed a certain threshold.

```
ES_Event RunTapeSensor(ES_Event ThisEvent)
{
    ES_Event ReturnEvent;
    ReturnEvent.EventType = ES_NO_EVENT; // assume no errors

    /********************************************
     in here you write your service code
     *******************************************/
    static ES_EventTyp_t lastEvent1 = NoFrontRightTape;
    static ES_EventTyp_t lastEvent2 = NoFrontLeftTape;
    static ES_EventTyp_t lastEvent3 = NoCannonTape;

    ES_EventTyp_t curEvent1 = NoFrontRightTape;
    ES_EventTyp_t curEvent2 = NoFrontLeftTape;
    ES_EventTyp_t curEvent3 = NoCannonTape;
    PWM_AddPins(PWM_PORTZ06);
    PWM_SetDutyCycle(PWM_PORTZ06, 500);

    uint16_t Tape3 = AD_ReadADPin(AD_PORTV6); //Read cannon tape
    uint16_t Tape1 = AD_ReadADPin(AD_PORTV4); //Read right tape sensor
    uint16_t Tape2 = AD_ReadADPin(AD_PORTV3);//Read left tape sensor

    switch (ThisEvent.EventType) {
    case ES_INIT:
        // No hardware initialization or single time setups, those
        // go in the init function above.
        //
        // This section is used to reset service for some reason
        break;

    case ES_TIMEOUT:
//        printf("Cannon Tape Value: %d\r\n", Tape3);
//        printf("Right Tape Value: %d\r\n", Tape1);
//        printf("Left Tape Value: %d\r\n", Tape2);
        if (Tape3 < 300) {
            curEvent3 = CannonTape;
        }
        else{
            curEvent3 = NoCannonTape;
        }
        if (Tape2 < 300) {
            curEvent2 = FrontLeftTape;
        }
        else{
            curEvent2 = NoFrontLeftTape;
        }
        if (Tape1 > 700) {
            curEvent1 = FrontRightTape;
        }
        else{
            curEvent1 = NoFrontRightTape;
        }
        if (curEvent3 != lastEvent3) { // check for change from last time
            ReturnEvent.EventType = curEvent3;
            ReturnEvent.EventParam = Tape3;
            lastEvent3 = curEvent3; // update history
            PostRobotHSM(ReturnEvent);
        }
        if (curEvent1 != lastEvent1) { // check for change from last time
            ReturnEvent.EventType = curEvent1;
            ReturnEvent.EventParam = Tape1;
            lastEvent1 = curEvent1; // update history
            PostRobotHSM(ReturnEvent);
        }
        if (curEvent2 != lastEvent2) { // check for change from last time
            ReturnEvent.EventType = curEvent2;
            ReturnEvent.EventParam = Tape2;
            lastEvent2 = curEvent2; // update history
            PostRobotHSM(ReturnEvent);
        }
        ES_Timer_InitTimer(TAPE_SENSOR_SERVICE_TIMER, TIMER_1_TICKS);
/*#ifndef SIMPLESERVICE_TEST           // keep this as is for test harness
```

Tape Service Code

The final type of sensor was the track wire sensors. Again, these sensors were very similar to the previous sensors. There was an service created that would read the pin value from the UNO 32 board, which was coming from the track wire. The service would begin at a null value and would only post an event if both of the value being read was pass a certain threshold. The main difference for the track wire detectors was that the output was a analog reading, instead of it being digital. The reason why we decided to have an analog reading for the track wire was to make it easier to find the wall when hugging the beacon tower. When we had the track wire give an digital readings, the track wire near the motors would sometimes read the noise coming from the motors (and this was the rear track wire). Due to this, there were sometimes when the robot would be circling around the beacon tower, and the front track wire detector would pick up the actual track wire; and the robot would think that it found the tower. So, switching to analog values was an easy fix for this problem. Since the noise coming from the motors would not be enough to pass the threshold.

```
ES_Event RunTrackWire(ES_Event ThisEvent) {
    ES_Event ReturnEvent;
    ReturnEvent.EventType = ES_NO_EVENT;
    static ES_EventTyp_t lastEvent = No_Wall_found;
    ES_EventTyp_t curEvent = No_Wall_found;
    uint16_t Track1 = AD_ReadADPin(AD_PORTW7);
    uint16_t Track2 = AD_ReadADPin(AD_PORTW8);

    /*******************************************
     in here you write your service code
     *******************************************/

    switch (ThisEvent.EventType) {
        case ES_INIT:
            // No hardware initialization or single time setups, those
            // go in the init function above.
            //
            // This section is used to reset service for some reason
            break;

        case ES_TIMEOUT:
            //          Track1 = AD_ReadADPin(AD_PORTW7);
            //                          printf("Front Track Wire: %d\n", Track1);
            //          Track2 = AD_ReadADPin(AD_PORTW8);
//            printf("Front Track Wire: %d\n", Track1);
//            printf(" Rear Track Wire: %d\n", Track2); // assume no errors
            if ((Track1) > 850) {
                if (Track2 > 850) {// is battery connected?
                    curEvent = Wall_found;
                }
//            } else
//                if ((Track1) < 300) {
//                    if ((Track2) > 880) {
//                        curEvent = Wall_found;
//                }
//            } else {
//                //                curEvent = No_Wall_found;
//                //                curEvent = No_Wall_found;
//                curEvent = lastEvent;
            }
            if (curEvent != lastEvent) { // check for change from last time
                ReturnEvent.EventType = curEvent;
                lastEvent = curEvent; // update history
                PostRobotHSM(ReturnEvent);
            }
            ES_Timer_InitTimer(TRACK_WIRE_SERVICE_TIMER, TIMER_2_TICKS);
```

Track Wire Service Code

Those three services were the only service code created for this project. All that was left was to add the events to the ES configure file. As the ES configure file would hold all the events (as shown in the figure below).

```
static const char *EventNames[] = {
    "ES_NO_EVENT",
    "ES_ERROR",
    "ES_INIT",
    "ES_ENTRY",
    "ES_EXIT",
    "ES_KEYINPUT",
    "ES_LISTEVENTS",
    "ES_TIMEOUT",
    "ES_TIMERACTIVE",
    "ES_TIMERSTOPPED",
    "BATTERY_CONNECTED",
    "BATTERY_DISCONNECTED",
    "NUMBEROFEVENTS",
    "No_Beacon_found",
    "Beacon_found",
    "No_Wall_found",
    "Wall_found",
    "No_Ball_deposit",
    "Ball_deposit",
    "NoFrontRightBump",
    "NoFrontLeftBump",
    "NoSideBump",
    "FrontRightBump",
    "FrontLeftBump",
    "SideBump",
    "NoFrontRightTape",
    "NoFrontLeftTape",
    "FrontRightTape",
    "FrontLeftTape",
    "NoCannonTape",
    "CannonTape",
    "NoSeeking",
    "GoSeeking",
};
```

Events

5.3 Hierarchy State Machine

We decided to design a hierarchy state machine, in order to keep the software from being too clustered, and more flexibility when telling the robot what to do and when to do something.

```
typedef enum {
    InitPState,
    Lookout,
    Search,
    Pursue,
    Flank,
    Destroy,
    Escape,
} TemplateHSMState_t;
```

HSM States

In the figure above, it displays the states that are in the hierarchy state machine (with the flank state not being used at all). Each state had its own sub state machine for it. With the first actual state being the Lookout state.

In the Lookout state, the robot would simple spin in place. The objective of this state was to find a beacon quicker. Instead of having the robot start moving mindlessly, the robot would do a ”lookout” and look around for the beacon. Initially, the robot would spin the other direction if it detection floor tape, since we were under the impression that the robot would be starting inside the playing field. However, once we were testing, we were informed that the robot would not start fully inside the playing field. This caused us to take out the tape events from this sub state. Since if we kept them, the robot would never look inside since the tape would block it.

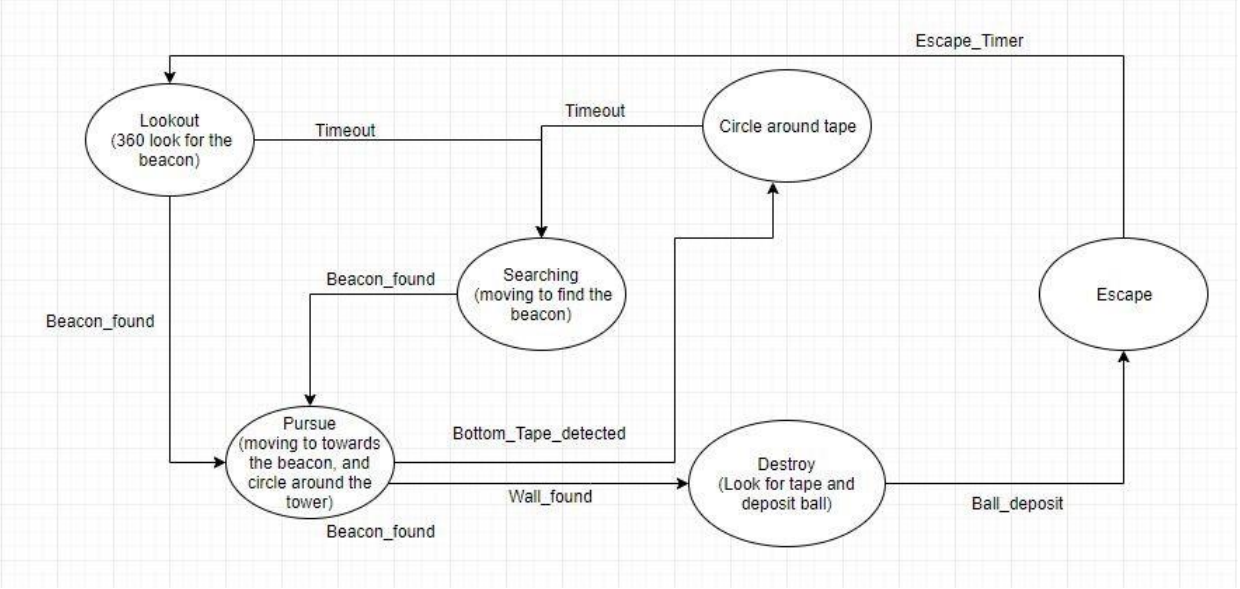
If the robot would not detect any beacon tower withing a time limit, then the state machine would transition into the Search state. The Search state would simple have the robot move straight. If the robot would detect floor tape, it would start hugging the tape on the side that detected the tape. This state would also have a time limit, that if no beacon is detected within that time limit, then the state machine would transition back into the Lookout state to look for a beacon. If a beacon is found, in either the Lookout state or the Search state, then the state machine would transition into the Pursue state.

Inside the Pursue state, the robot would move full speed towards the beacon. However, since out beacon detector was offset to the left, the robot would move slightly to the left when pursuing the beacon tower. Due to this offset, when the beacon was lost, the robot would pivot right until the beacon signal is detected again. Then the robot would continue pursuing the beacon tower. Once the robot bumps into the beacon tower, either the left front bumper or the right front bumper, the robot would then begin to line up with the tower. Since we had a left side shooter, we would have to line up with the left side of the robot facing the tower. The method we implemented to line up was to utilize our bumpers to line up. We would have the robot back up a small amount, and go forward again, but the robot would turn slightly to the right. The robot would continue to do this until the left side bumper would hit the tower. Hitting the side bumper would cause us plenty of problems. As the robot would sometimes miss the bumper, or the bumper would get stuck. To fix this problem, we added a round little side bumper to the already made side bumper. This would stop it from getting stuck, and it would make it easier to hit the bumper when trying to line up with the tower. Once the left side bumper was hit, we would

begin to hug the tower. Hugging the tower is also how we would circle around the beacon tower. The robot would continue to hug around the beacon, until it finds the track wire.

Once the robot would find the track wire wall, it would transition into the Destroy state. Inside this state, the robot would move back and forward looking for the tape on the beacon tower wall. Once the tape was detected, the robot would reverse until it loses the tape. This was done due to our wall tape sensor being behind our ball dropper. So to line up correctly with the hole, the robot needs to reverse until it loses the tape. Once the tape is lose, the ball dropper would be correctly line up with the hole. Then the ball dropper motor would begin to move up the elevator. The motor would move for only a certain amount of time (just enough to deposit just one ball). After the ball is deposited, the state machine would transition into the Escape state.

In the Escape state, the robot would tank turn to the right (away from the beacon tower). Then the robot would move forward for a small amount of time. Once that time is up, it would transition back into the Lookout state, and restart the whole process again to find a new beacon tower.



Flow diagram of the Hierarchy State Machine

6 FEATURES

1.Can move forward.
2.Can move backwards.

3.Can turn around.
4.Can register bumps.
5.Can locate black tape on floor.
6.Can locate a beacon tower.
7.Can locate the track wire on the beacon tower.
8.Can locate black tape on the tower.
9.Can deposit a ball in the tower.

7 RESULT

The final result was a robot with multiple working components inside it. A beacon detector that pick up a signal from a beacon. Three tape sensors, all of which can detect black tape on the floor, and black tape on the beacon tower. Two track wire detectors, that are capable of detecting a track wire inside a beacon tower. Working bumpers that can register and event when pressed. Two working motors, with adjustable speed, and bi-direction. One uni-directional motor, also with adjustable speed. With those components, the robot was able to locate the beacon tower, and locate the track wire inside the beacon tower. The robot also had a working ball dropper that would deposit one ball. The robot is also capable of locating another beacon and repeating the process with the new beacon. Here is a video of the robot in action, Video Link.

8 CONCLUSION

In conclusion, our goal for this project was to design, and construct, a robot with the ultimate goal of depositing a ping pong ball in the correct hole of the correct side of a beacon tower within a playing field. This was accomplished by having the robot detect and avoid black tape on the ground, while locating the beacon tower. Once found the robot would encircle the tower to find the correct side, indicated by a track wire. The robot would then move along this wall in search of the black tapel. Once the black tape was found a ball would be deposited and the robot would move onto the next tower. Throughout the project, we made many design revisions as the task would challenge our current implementations.

When designing the body of the robot, originally, we had a triangle body. However, with the triangle body, it was proving to have many problems when making the state machine; thus, we made the choice to go with a square base. The triangle base mainly complicated the navigation of the robot. As it is essential to get to the beacon, we opted to go for the easier to navigate square base. As for the other mechanical aspects of the robot, we opted for a minimalist approach. As few sensors, bumpers and motors as we could get away with the better

Electrically we knew consistency of our sensors was key. A reliable input from our sensors would drive the state machine to correctly function. Without reliable sensors we could not guarantee consistent performance from our robot. Without this consistency we would not be able to fall under the 66

For the software, we mixed the two approaches of the mechanical and electrical. We did not want to create a complex state machine, as it would prove to be difficult to debug, but we did not want to create a very simple modular state machine. An example of a simplification we enabled was we did not use an event checker (that was given to us via ES framework), since the event checker essentially behaves the same as a service checker. In return, our services are simple to read. The one drawback to this approach is the design may be less robust to edge cases. We noticed this when checking off and during the competition. Fortunately, there were not many edge cases and our approach worked as intended. 1.3cm While we were three different people, with different work schedules, and different thought process, we were able to come together and work towards a common goal effectively and efficiently while learning a great deal about what goes into creating and executing a project. We were successful in designing a robot to put an end to SALT. The robot was a success.