# CMPE118/L Fall 2017 Slug Wars: Han Yolo Robot

Esteban Guerrero, Aleksandr Kagan, Derek Quiroz

December 13, 2017

## Introduction

A long time ago in a galaxy far, far away, the First Order, led by Kylo Ren, has been making massive strides in their galactic conquest of the galaxy with the ultimate goal of enforcing stability and order. After successfully launching their mega weapon, Star Killer Base, and neutralizing the Alliance, the only chance of resistance lies within a small band of rebels led by Finn and Rey. Aided by a rebel spy, Star Killer Base was successfully destroyed, but this only fueled the First Order to strike back even harder.

After intercepting an encrypted message to Rebel leader Leia Organa, the First Order is now aware of the location of the rebel base. In a huge effort to wipe the rebels off the face of the galaxy, the First Order is launching a huge strike on the rebel base, led by Kylo Ren himself. Now, the rebels' only hope lies with a ragtag bunch of CMPE118 students who have mastered the art of state machines and autonomous robots.

Will the rebels be able to protect their base long enough to evacuate?

# Mechanical

## Physical Design

Our original design was a gross overestimation of the size constraints and our mechanical engineering capabilities. In our original design, we figured that we would have more than enough real estate on the top level of our bot for a plethora of ping-pong balls, but we had to redesign our launcher numerous times before settling on our final design. Originally, we were planning on a modular design consisting of three layers. And, in the end, we kept that modular design, but each level had to be tweaked as problems arose.

Also from our original design, we kept a centered two wheel drive and a circular shape for mobility purposes. When approaching corners, the easiest approach is to stay in place and turn exactly 90 degrees. Therefore, having the wheels centered (as opposed to a rear wheel drive) would make it simplest. The circular design was mostly for aesthetic purposes originally, but after purchasing our wheels, we got lucky since the bot only fit in the 11' by 11' dimensions diagonally. If it were to be a square bot, we would have had to make it skinnier to account for wheel width. Below is the SolidWorks mockup of our robot:

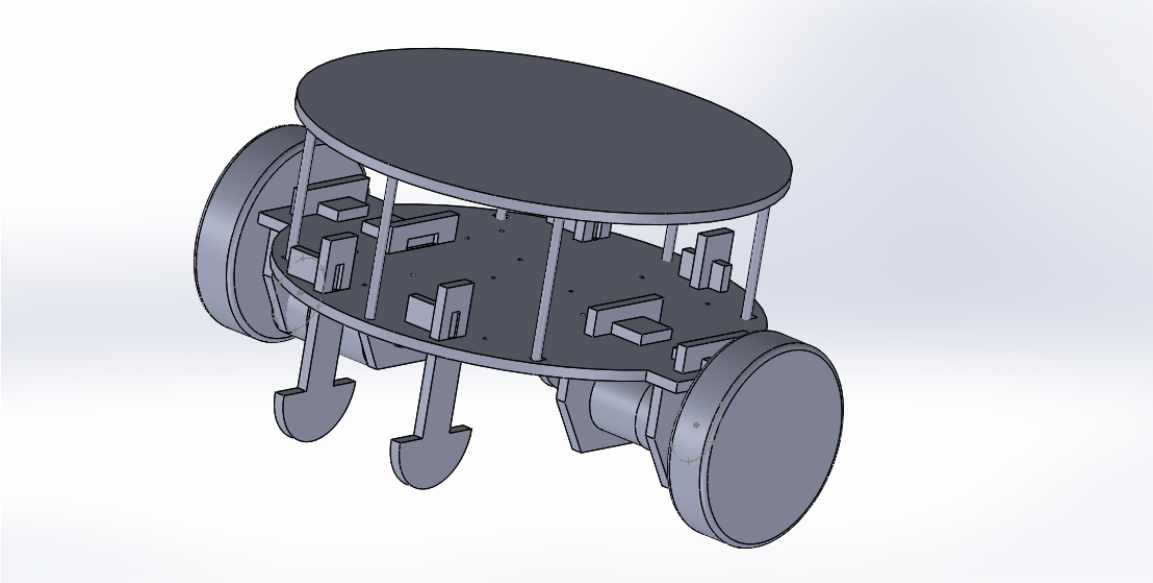

Figure 1: SolidWorks Model

From the bottom up, we essentially had three layers. The bottommost layer was our

motors and wheels, skids, battery mount, and tape sensors for line following. With the motors only taking up space on the sides, and the tape sensors being so small, we had more than enough space to mount the battery to the underside as well. The battery mount was directly behind the front skids, and the tape sensors were extended down to roughly an inch above the ground for accuracy's sake.

On the middle layer was the majority of our circuitry, bumpers, track wire sensor, and Ren detector. Most of the circuitry will be discussed in the Electrical section below, but we relied on the UNO, two bump sensors, dual H-bridge, unidirectional motor driver, beacon detector, and track wire sensor. There was enough space for everything but we wouldn't have been able to fit much more into the space. The inductor for the track wire sensor was mounted on the left side of the bot near the rear since we planned on going clockwise around the field.

Lastly, on the top level was the launching mechanism(s) and beacon detector. There were definitely a few iterations that we had to go through, but we settled on two different launchers, one for each type of target. In order to take down the AT-M6's we had a side mounted launcher using an accelerator wheel and an RC servo as our indexer. The Ren Ship launcher was a slapper arm on an RC servo with guide rails. We designed the hopper for the AT-M6 launcher to hold up to four or five balls to account for a misfire or two. Pictured below is the fully assembled robot in all its glory:

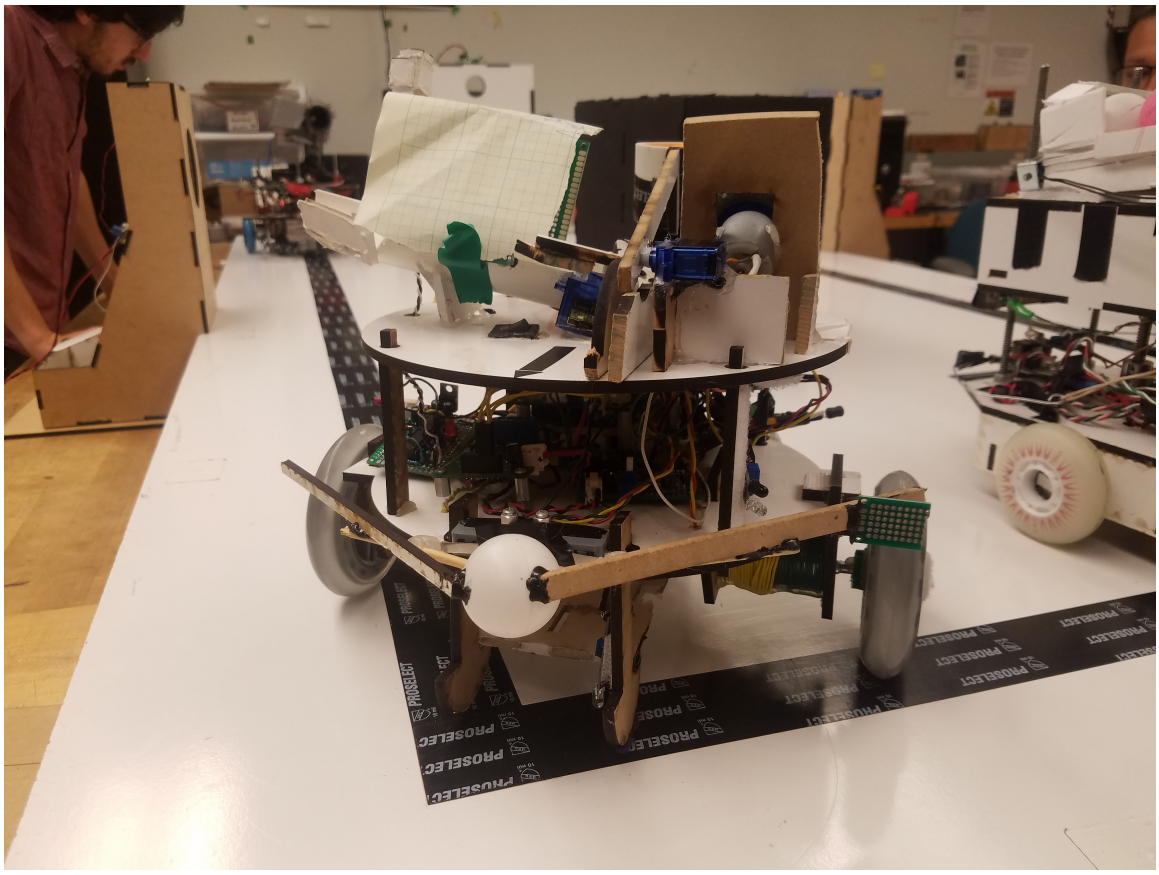


Figure 2: Final Robot Front View

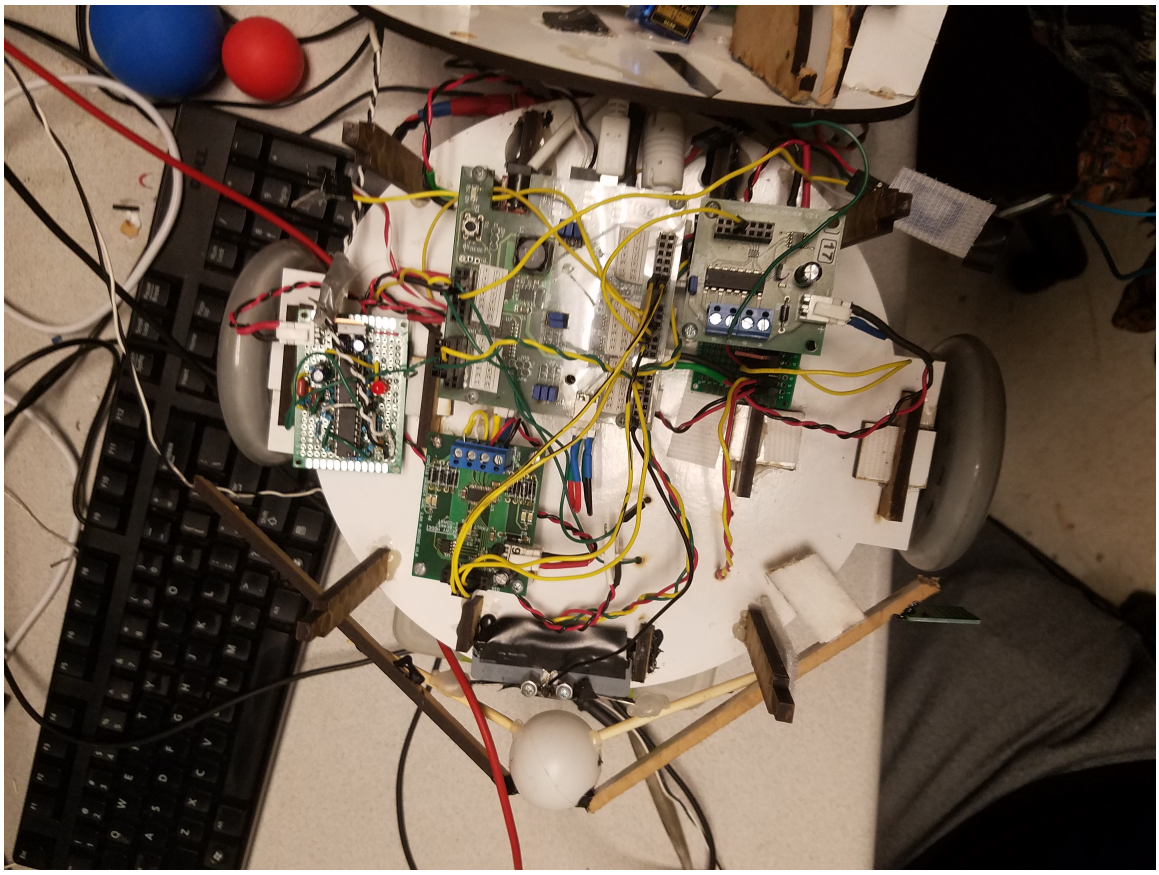

Figure 3: Robot Layer 1 Top View

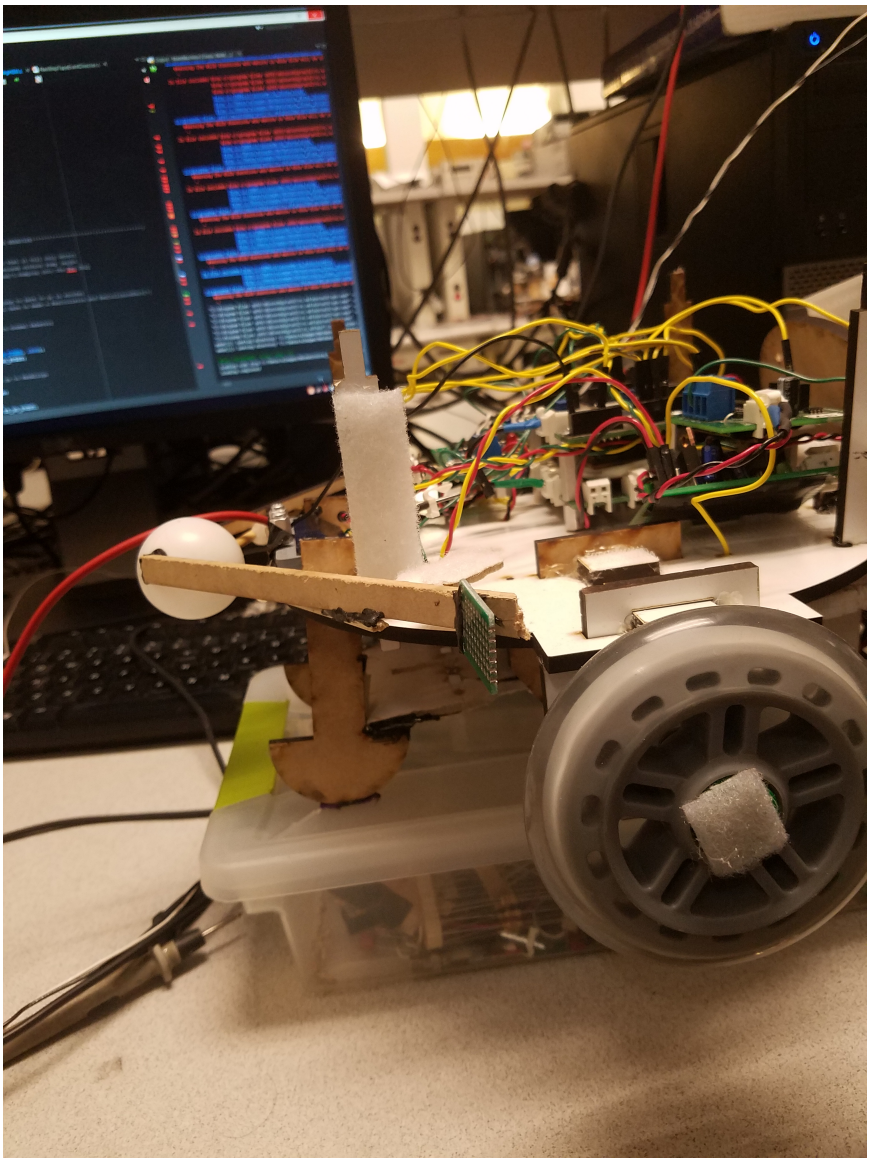

Figure 4: Robot Side View

## Materials and Implementation

Admittedly, our implementation and materials could have been chosen wiser and designed better, but by Monday of competition week, we were getting increasingly desperate.

Our original mock-up in SolidWorks was our final design. We knew that the body of the robot would have to be made from MDF in order to maintain structural integrity, so that is what we crafted the two platforms, motor mounts, platform supports, and skids from. Even if we decided to choose something else to make the skids out of, or buy better motor mounts, we would at the very least have the MDF style to run tests on. However, we felt no need to change any part of the body since it was compact and got the job done.

Again, starting from the bottom layer, the skids were made from MDF, cushioned by a piece of tape so as not to scratch up the field. The battery holder was made from a paper towel roll and wire hot glued to Velcro to secure the battery into place. We also mounted Velcro to the USB cable and the underside of the bot in order to secure it in a place that wasn't under the UNO. The tape sensors were mounted right above the ground using a foam-core sheet that attached to the skids and Velcro. The motors that we used were rated for 150 RPM and mounted pretty nicely to the MDF mounts we made (aside from the fact that the wrong dimensions were shown on Amazon. The motor shaft was attached to the wheels we bought using a 3D printed coupling that wasn't quite made for our size shaft. A lot of epoxy was used to keep the left wheel attached.

On top of the lower platform we mounted all of the given boards (UNO, H-Bridge, etc.) using screws and all of our own boards using Velcro. we either used laser cut holes as the alignment for the boards or had to drill out holes when our measurements were slightly off. This worked in our favor too because many of the holes that we measured incorrectly were used to feed wire into the bottom layer for tape sensors. The opposite outcome arose when fitting our bumpers. In measuring wheel width incorrectly, we had very little room to make our bumpers. The bumpers ended up being a ping-pong ball (ours) with holes bored into it to support wooden dowels which supported MDF strips. Given more time, we would have loved to make better bumpers, but we did what we could with the space we had in the shortest amount of time possible. The bumpers turned out to work surprisingly well.

On the top platform, we mounted our hastily-made launching mechanisms. After a few failed attempts, we were able to perfect the design, but not quite the implementation. Or, at least, it wasn't very aesthetically pleasing. The accelerator wheel was mounted using hand-cut MDF, the hopper was made with re-purposed PVC and perfboards, and the limiter for the slapper arm was a heatsink on one end and a puddle of hot glue on the other. Granted, everything worked beautifully, it just would have been nice if we focused more energy into getting a solid design from the get-go instead of scrambling. Our slapper arm itself, however, turned out really nice with

only a strip of MDF, PVC, and tape. The bill of materials for our robot is detailed below:

| Part | Quantity | Supplier | Price | ETA |
|---|---|---|---|---|
| DC Motor (150 RPM) | 2 | Amazon Prime | $30.00 | 11/14/2017 |
| DC Motor (500 RPM) | 1 | Amazon Prime | $15.00 | 11/14/2017 |
| Bump Sensors | 10 | Amazon Prime | $10.00 | 11/14/2017 |
| PVC (1.5"x3') | 1 | SL Lumber | $7.00 | 11/12/2017 |
| Razor Wheels | 2 | Big 5 | $15.00 | 11/13/2017 |
| MDF | 2 | BELS | $7.00 | 11/14/2017 |
| mounting hardware | 1 | SL Lumber | $10.00 | 11/15/2017 |
| RC Servo | 2 | Amazon Prime | $5.00 | 11/16/2017 |
| Velcro | 4 | SL Lumber | $20.00 | 11/20/2017 |
| | | total: | $119.00 | |

Figure 5: Cost Breakdown

# Mechanical Setbacks

The mechanical setbacks that we had were actually pretty significant. As previously mentioned, we had some trouble with the wheels. Finding a decent coupling was not easy and we had to settle for a hand-me-down since it was going to be produced the quickest. We drilled out the original inner hole since it was too small and relied on epoxy and hot glue to keep the wheels on the bot. Even so, the left wheel constantly fell off until we pilled on the epoxy and let it sit overnight. Another issue we had with the wheels was that they ended up being taller and wider than we were expecting. Fixing the height was easy; we just had to raise the skids and not build the launcher as tall (by a dozen or so millimeters. However, width wise our robot was longer than eleven inches, but it fit diagonally in the dimensions. That really put a constraint on

our bumpers more than anything. In order to stay within our eleven inch cube, since everything was already super compact, we ended up with bumpers that couldn't extend to in front of the wheels resulting in numerous head-on collisions to the wheels. That's honestly probably why we had so much trouble keeping the left wheel attached.

Another fun issue we had was the near-combustion of our robot. At the very least, we weren't the only group to run into this problem and we weren't the last. With the power switch in the OFF position, we figured we had nothing to worry about when it came to electrical components. Boy were we wrong. The USB cable got tucked under the UNO and shorted two very specific nodes on the power distribution board. This turned the inductor orange-hot and resulted in a very panicked "UNPLUG THE F***ING BATTERY." It was terrifying to almost see our 3 week old bot almost burst into flames. There was an unreasonable amount of magic smoke as well. This set us back the entire day essentially, having to replace the power distribution board and rewire everything three times to account for loose fuses on the next one.

Another multi-day setback was the launching mechanism. We sadly realized that our original plan of having an accelerator wheel on an adjustable angle was slightly beyond our mechanical prowess. So we tried to settle for a single angle launcher and adjust the speed depending on the target. This was highly inaccurate and our launch wheel was not fast enough to even propel a ball high enough to reach the Ren target. After a few time-consuming iterations, we settled on two separate mechanisms: a side mounted launcher for low targets and a slapper arm for the final target. This worked great once we actually created guide rails for the arm so it wouldn't pop off.

Our final and biggest set-back was a two-day-long-fuse-blowing-fiasco. The problem was that the accelerator wheel was drawing almost an amp of power on startup. Combined with moving wheels, RC servos, and powering the sensors, we blew a few too many fuses. It took a while to figure this out though: we started by adding capacitors to the 3.3V/5V regulator board, then stopping the drive wheels before turning on the launch wheel, then connecting the launch wheel to the opposite power rail as the drive wheels and powering it directly off of 9.9V at a very low PWM (as opposed to powering off 3.3V). It admittedly took two too many fuses to figure out what was going wrong, unfortunately.

# Electrical

## Track Wire Detection Circuit

The track wire detection circuit was an essential part of the robot. It was the primary sensor used to help the robot complete its task of shooting down the three AT-M6 targets. Each AT-M6 target was 8 inches wide and 12 inches tall with a 4 inch diameter target hole centered 8 inches above the field. The target had a vertical wire running from the field to the target hole. The wire carried a $24 - 26\ kHz$ signal through it. This current-carrying wire generates a magnetic field around it. Through the use of an inductor, we were able to sense the change in magnetic flux within the coil caused by the track wire's magnetic field. This change in magnetic flux induces a voltage within the inductor and provides us with a sinusoidal signal we can use.

The inductor was used as our "sensor" and was mounted on the side of our robot such that it would detect the track wire as the robot navigated alongside the tape and around the field. When the robot was on the tape, it was around 6inches away from the AT-M6's track wire which resulted in the voltage induced within the coil to be relatively small-signal.

Thus, we were faced with the task of designing a circuit to amplify the signal detected by the coil to a reasonable range and convert it to a DC signal which we could then read using the ADC pins on the PIC32. In addition to that, we had to account for noise introduced in the circuit. The first block of the circuit was an LC tank oscillator. This LC tank is a parallel resonance circuit composed of an inductor in parallel with a capacitor. The resonant frequency was set to be $f_C = 27.5\ kHz$. The output of this signal was amplified with a noninverting amplifier with a gain of 10. Following this first gain stage was a passive high pass filter with a cutoff frequency of $f_C = 10\ kHz$. This filter was intended to prevent low-frequency noise from interfering with the circuit. We found it unnecessary to implement this using an active filter, as we did not feel we would be subjected to much low-frequency noise in the first place.

A second passive high-pass filter was also added in the later stages of the circuit centered at the same cutoff frequency. With both these filters in place, it was more than sufficient to eliminate the possibility of any low-frequency noise to interfere with the signal we wanted to pick up. We learned that the motors we were using to drive our robot generated noise at around $30\ kHz$. Through our mechanical design, our ball launcher was to fire off to the side. Therefore, our trackwire "sensor" logically had to be side-mounted, which left it very close to our driving motors and could potentially succumb to the noise generated by the motors. In an attempt to eliminate the noise

from the motors, we introduced a second-order Chebyshev low-pass filter with a cutoff frequency of $f_C = 28\ kHz$. Given the tolerances of the components we had access to, the filter did not attenuate frequencies until at around 29 $kHz$, which was sufficient for our purposes. Along with the active low pass filter, the circuit included additional gain stages and ended with a peak detector which gave us a DC output. The final schematic for the circuit is shown in figure 6 below. The actual implementation of the trackwire circuit, on the perfboard, is also shown in figures 8 and 7 below.

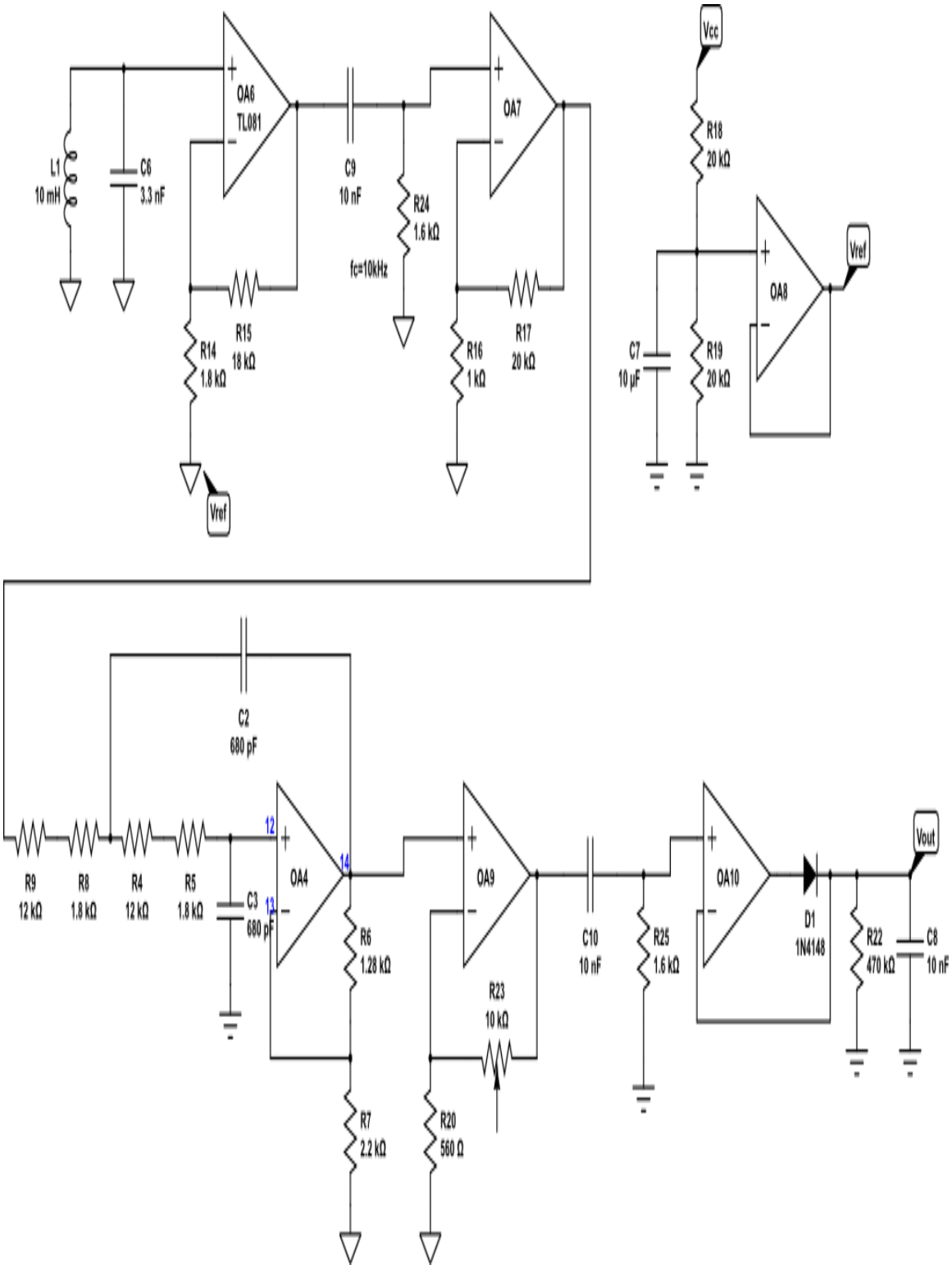


Figure 6: Final schematic for the trackwire detection circuit

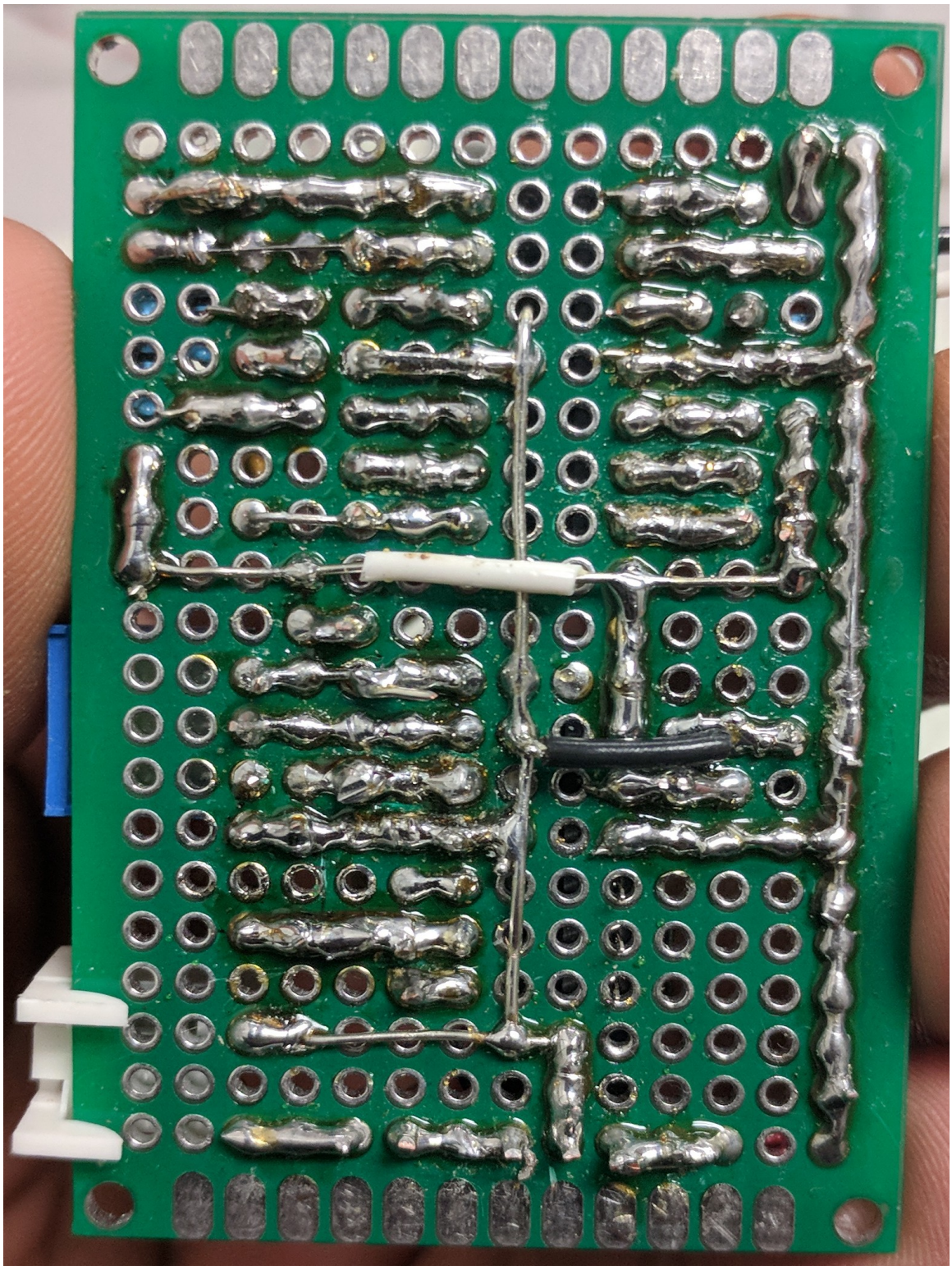

Figure 7: Trackwire Bottom View

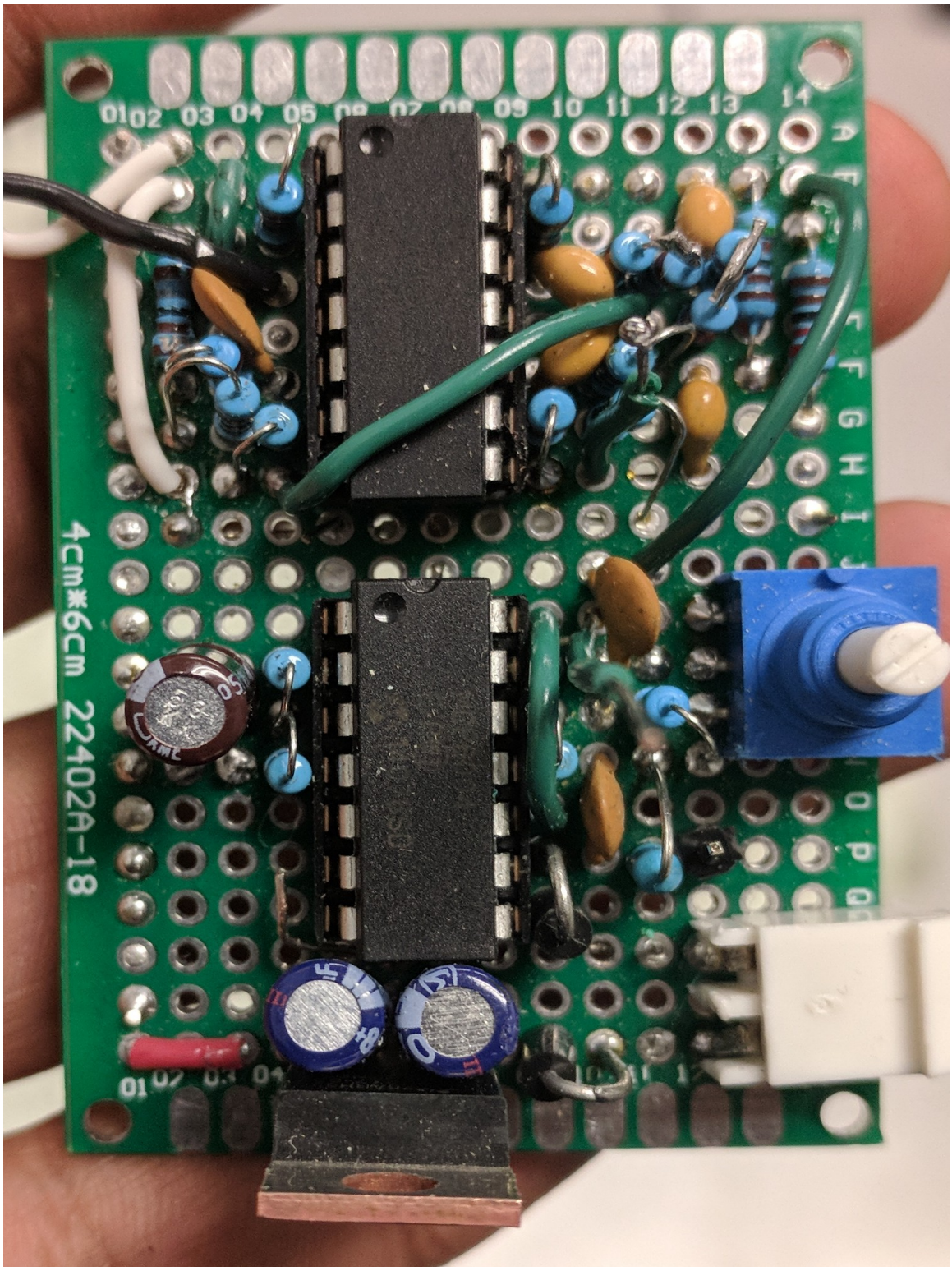


Figure 8: Trackwire Top View

## Beacon Detector

The beacon detector did not differ from the beacon detector we designed and prototyped for the prior labs. The specifications for it were the same as before. These specification are as follows: the circuit should be able to detect the beacon at 2 $kHz$ with a range of 1 to 6 feet and it should reject the beacon at 1.5 $kHz$ and 2.5 $kHz$ within the same range. The beacon detector was not used much in our attempt to complete the task at hand. We only used it in our initial state to adequately orient ourselves and navigate to the tape efficiently. As such, the design of the beacon detector was not as crucial as the one for the trackwire.

The circuit consisted of a transresistive block, which consisted of an IR sensor win a sinking configuration. This was passed to three successive gain stages. This was then sent to a second-order bandpass filter. This was then sent to a peak detector, then a comparator and sent to a buffer and LED to indicate whether we had detected the beacon at 2kHz or not. The final beacon detector, on the perfboard is shown below in figure 9 (top view) and in figure 10 (bottom view).

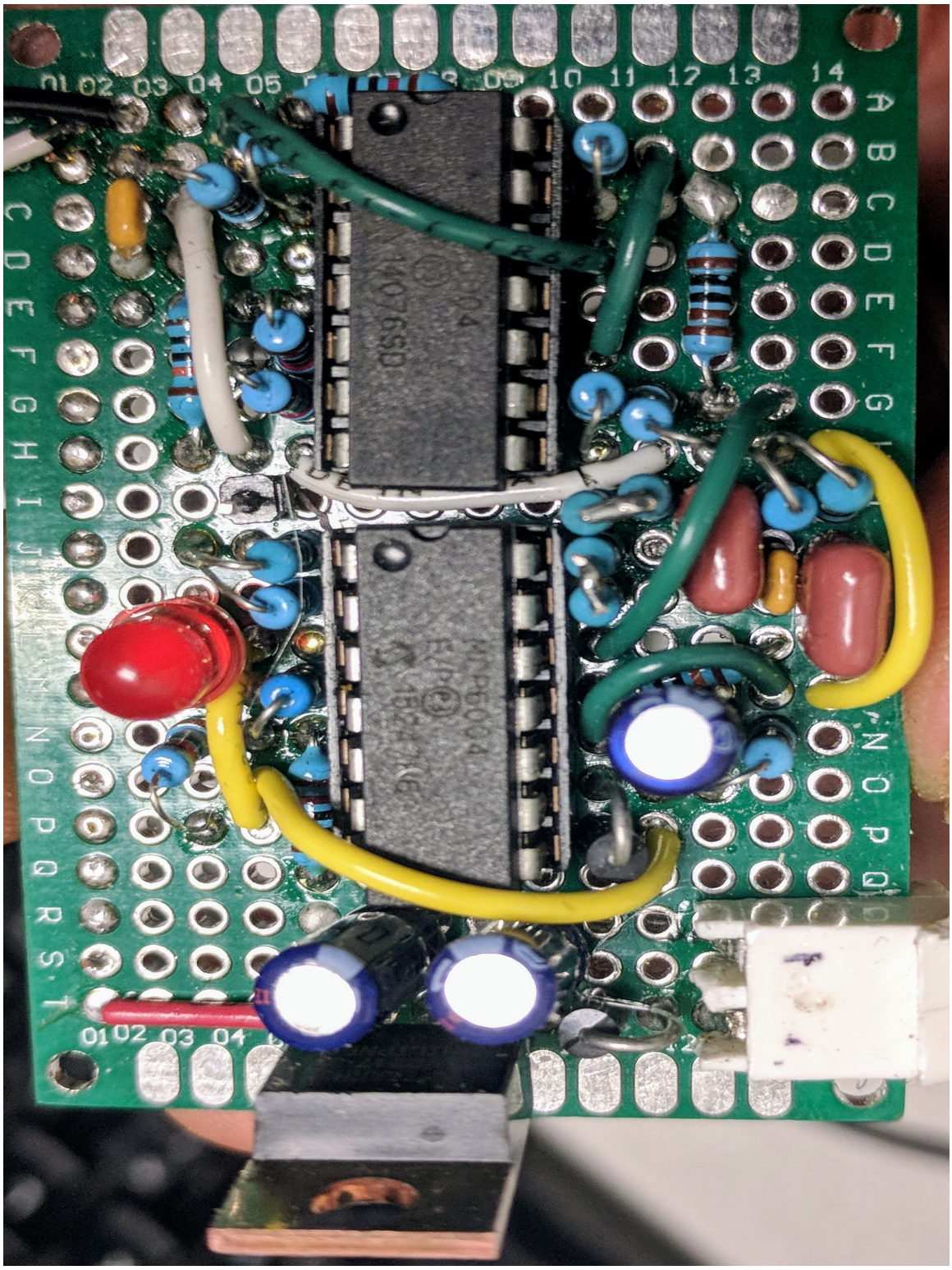


Figure 9: Beacon Detector Top View

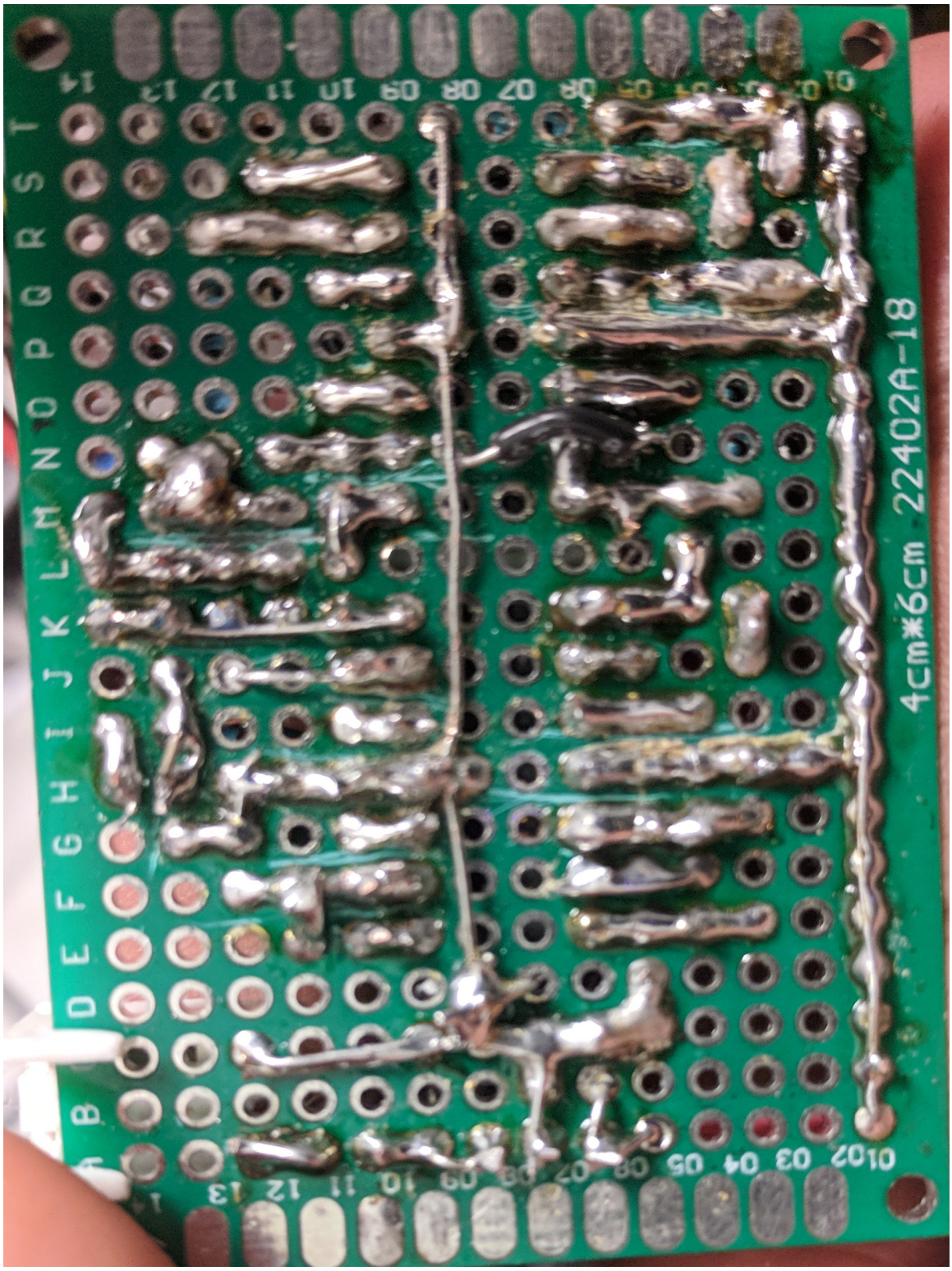


Figure 10: Beacon Detector Bottom View

### Power Distribution Board

Another crucial part of hardware that was essential for the completion of the robot was a power distribution board. This power distribution board took the output from the UNO's distribution board, the voltage from the battery and stepped it down to use for our motors. This power distribution board consisted of a 9.9V input (from the battery) which led into a 5V linear voltage regulator. Output of the 5-volt regulator was then sent into the input of a 3.3V linear voltage regulator. The 3.3-volt outputs were used to power our tape sensors and our motors and the 5-volt output was used for the limit switches we used for our bumpers.

In regards to soldering our circuits on the perfboard, we heavily emphasized incremental design implementation. After soldering every single component, we ran a continuity test in reference to all components currently populating the board. We did this for three reasons: to ensure nothing had been shorted, to verify the correct components were connected and to make sure a good soldering joint had been made (no cold joints). Once the connection was verified, we tested the stage. We sent in a generic signal from the signal generator (when necessary) and verified that the circuit operated as intended. Furthermore, we soldered and tested every single stage of each circuit individually. Thus, we made sure every single block of the circuit was working standalone before connecting it with multiple stages. As such, we followed this process throughout the implementation of all the circuits. Upon completion of the soldered circuit, the circuit operated as intended and did not pose any major issues.

## Electrical Setbacks

The first iteration of the power distribution board caused us some problems. The motors drew too much current and caused us to overload the fuses on the UNO's power distribution board and caused us to blow a couple of them. When it came to the trackwire detection circuit, the active low-pass filter did not attenuate the signal enough at 30kHz, so we were unable to completely eliminate the noise generated from the motors. Thus, we faced issues when our trackwire detected random noise, which interfered with our signals and our robot's behavior. We had to get creative in addressing these issues as we did not want to redesign nor resolder a whole new trackwire circuit. In addition to the trackwire, we faced a myriad of issues when trying to implement our tape sensors. We went through a few iterations of trying to implement our tape sensors, and they all failed. We made a total of 3 different tape sensor distribution boards and modified each of them a couple times to no avail. Trying to get our tape sensors to work set us back multiple days and incredibly

halted our progress. We ended up using discrete tape sensors we acquired from a fellow classmate.

# Software

The bulk of our coding in this project is implementing the Events and Services Framework in a clear, effective way. Through our experience with Lab 0 and programming the Roach, we had a solid understanding from which to base our new state machine. Before we wrote the state machine, we incrementally tested out all of our components individually via test harnesses. Using this method allowed us to fix problems one by one instead of having to debug the entire state machine all at once. Once all of the components were proven to be working as desired, we moved on to create our state machine for the project. We chose to implement a hierarchal state machine so that we can break down our code into easy to manage portions and have an organized, professional layout.

## Top Level

The first step in creating our hierarchal state machine was choosing the top level that we will be implementing. We decided that an appropriate amount of top level states would be the initial orientation, searching for the 3 targets, and engaging the final Ren ship.

## Orientation State

Before we started implementing any of the state machine, we decided to make our lives easier by having a master pin configuration header file as well as a master motor driver header file that we would include where necessary and can easily change. Below is the master pin configuration and motor driver headers:

```
 * File: pin_configuration.h
 * Author: Kagan
 *
 * Setting the pin configuration for the final project.
   Total number of AD pins used:  9 (4 on V port, 5 on W port)
   Total number of I/O Pins used: 2
   Total number of PWM pins used: 2
 */

#ifndef PIN_CONFIGURATION_H
#define PIN_CONFIGURATION_H

#include "PWM.h"
#include "IO_Ports.h"
#include "AD.h"

// Beacon Detector Pin Configuration
#define BEACON_DETECTOR         AD_PORTW6   //uint16_t value

// Bump Pin Configuration
#define LEFT_SWITCH             AD_PORTW8   //uint16_t value
#define RIGHT_SWITCH            AD_PORTW5   //uint16_t value

// Motor Pin Configuration
#define LEFT_MOTOR              PWM_PORTY12 //PWM signal
#define RIGHT_MOTOR             PWM_PORTZ06 //PWM signal
#define LEFT_MOTOR_DIRECTION    PIN12       //Port X output pin
#define RIGHT_MOTOR_DIRECTION   PIN11       //Port X output pin
#define ALL_3_DESTROYED         PIN9        //Port X output pin
#define LAUNCHER_MOTOR          PWM_PORTY10 //PWM signal

// Tape Sensor Pin Configuration
#define TAPE_CORNER             AD_PORTV4   //uint16_t value
#define TAPE_FOLLOWER           AD_PORTV3   //uint16_t value
#define TAPE_REN                AD_PORTV5   //uint16_t value

// Tape Parameters
#define TAPE_TOP_PARAM          0x03
#define TAPE_CORNER_PARAM       0x04
#define TAPE_REN_PARAM          0x05

// Track Wire Pin Configuration
#define TRACK_WIRE              AD_PORTV6   //uint16_t value

// RC Servo Pins
#define INDEXER                 RC_PORTY06  //Servo
#define PUSHER                  RC_PORTY07  //Servo

#endif
```

Figure 11: Master Pin Configuration

```
#include "PWM.h"
#include "IO_Ports.h"
#include "RC_Servo.h"

void onwards_NOS(void);

void onwards_TYO(void);

void onwards(void);

void snails_pace(void);

void stop_everything(void);

void reverse(void);

void rotate_clockwise(void);

void rotate_clockwise_REN(void);

void rotate_counter_clockwise(void);

void rotate_counter_clockwise_REN(void);

void turn_right(void);

void turn_right_slower(void);

void turn_left(void);

void turn_left_REN(void);

void turn_left_slower(void);

void attack_ATM6(void);

void stop_attack_ATM6(void);

void ping_pong_dispenser_low(void);

void ping_pong_dispenser_med(void);

void ping_pong_dispenser_high(void);

void final_attack_low(void);

void final_attack_high(void);

#endif /* MOTORS_DRIVERS_H */
```

Figure 12: Master Motor Drivers

To start off, we rotate our bot until a beacon is detected. We have a timer built in that if an obstacle is blocking the path of the beacon after 5 seconds the robot will travel forwards until tape is detected and go from there. Once tape is detected, we rotate until our one tape following sensor goes low again. If the obstacle is in the way of robot's path to the tape, we have our robot reverse, slightly rotate, and continue trying to find tape. All of this happens in our sub-orientation state. Once the event has been consumed, an `ES_NO_EVENT` is passed up to the top level, indicating that the subroutine has successfully dealt with the event that has occurred. On our top level, we have a tape detected event transition into our next top level state, which is the searching state.

## Searching State

Once our initial orientation is complete, our algorithm to traverse the course is quite simple - once it's on the tape we immediately go off the tape. Similarly, once we are on the tape we immediately go off the tape. This is how we navigate the entire course in a smooth, simple, and efficient matter. The tape navigation sensor is located towards the left side of the robot so that more than half of it is inside the course at all times. We also have a tape sensor specifically for corner detections. This is place about 2 inches to the right of the tape sensor used to navigate the course. If an obstacle is detected during this time, such as in the initial orientation state, we reverse, rotate, and continue our tape following algorithm. If another bump is detected during this time then the same response occurs. Below is a snippet of our code for the tape following:

```
158         case SubTapeDetected:
159             switch (ThisEvent.EventType) {
160                 case ES_ENTRY:
161                     ES_Timer_InitTimer(OH_NO_TIMER, OH_NO_TIMER_LENGTH);
162                     turn_right();
163                     break;
164 
165                 case TAPE_NOT_DETECTED:
166                     nextState = SubWhiteDetected;
167                     makeTransition = TRUE;
168                     ThisEvent.EventType = ES_NO_EVENT;
169                     break;
170 
171                 case BUMP_PRESSED:
172                     nextState = SubCollision;
173                     makeTransition = TRUE;
174                     ThisEvent.EventType = ES_NO_EVENT;
175                     break;
176 
177                 case CORNER_TAPE_DETECTED:
178                     nextState = SubCornerDetected;
179                     makeTransition = TRUE;
180                     ThisEvent.EventType = ES_NO_EVENT;
181                     break;
182 
183                 case TRACKWIRE_DETECTED:
184 
185                     nextState = SubTrackWireDetectedState;
186                     makeTransition = TRUE;
187                     ThisEvent.EventType = ES_NO_EVENT;
188                     break;
189 
190                 case ES_TIMEOUT:
191                     nextState = SubCollision;
192                     makeTransition = TRUE;
193                     ThisEvent.EventType = ES_NO_EVENT;
194 
195                     break;
196 
197                 case ES_EXIT:
198                     stop_everything();
199                     break;
200 
201                 default:
202                     break;
```

Figure 13: Tape Detection Sub-State

```
206        case SubWhiteDetected:
207            switch (ThisEvent.EventType) {
208                case ES_ENTRY:
209                    turn_left();
210                    ES_Timer_InitTimer(OH_NO_TIMER, OH_NO_TIMER_LENGTH);
211                    break;
212
213                case TAPE_DETECTED:
214                    nextState = SubTapeDetected;
215                    makeTransition = TRUE;
216                    ThisEvent.EventType = ES_NO_EVENT;
217                    break;
218
219                case BUMP_PRESSED:
220                    nextState = SubCollision;
221                    makeTransition = TRUE;
222                    ThisEvent.EventType = ES_NO_EVENT;
223                    break;
224
225                case CORNER_TAPE_DETECTED:
226                    nextState = SubCornerDetected;
227                    makeTransition = TRUE;
228                    ThisEvent.EventType = ES_NO_EVENT;
229                    break;
230
231                case TRACKWIRE_DETECTED:
232                    nextState = SubTrackWireDetectedState;
233                    makeTransition = TRUE;
234                    ThisEvent.EventType = ES_NO_EVENT;
235                    break;
236
237                case ES_TIMEOUT:
238                    nextState = SubCollision;
239                    makeTransition = TRUE;
240                    ThisEvent.EventType = ES_NO_EVENT;
241                    break;
242
243                case ES_EXIT:
244                    stop_everything();
245                    break;
246
247                default:
248                    break;
249            }
250            break;
```

Figure 14: White Detection Sub-State

Once an ATM6 is detecting via our track wire detection circuit, our robot stops moving, takes a second to get our shooting wheel up to speed, then our servo pushes one ball into our launcher. Assuming the track wire is successfully shot, we increment our global track wire kill counter and continue searching for the remaining targets. If the track wire is still on after one shot, indicating a miss, we inch ever so slightly forward and shoot again. Once all 3 targets have been destroyed, which is checked

by a global incrementer that we have, we pass an event up to the top level to signal our state machine to transition into the third and final state, the Engaging State.

### Engaging State

Our last state has practically the same code as our searching state with only a few key differences. We have a tape sensor placed such that it detects the white Ren ship target. Once this event is passed through the state machine, within our sub-engaging state we transition into our Ren attack mode. From here, we reverse, rotate clockwise, and drive forward. We then realign ourselves with the target, drive towards it until we get a bump event, and then use our servo to raise a lever with a ping-pong ball at the end of it directly into the target. If we somehow overshoot the target, we have our bot slowly rotate counterclockwise which ideally gets the ball in on the second attempt.

## State Machine Setbacks

The state machine, as described above, is how it is supposed to work in theory. However, we saw that in practice the robot did not always behave as we expected it to for a wide variety of reasons.

Before we had our final robot laid out, we initially tried implementing 4 tape sensors for navigating around the course. However, when creating our event checker passing the tape parameters and reacting based off that proved to be needlessly complicated. We came to realize that a simpler method could be done by just having a single tape sensor traversing the tape using the method we described earlier. This way we had an event checker with just one parameter that we could simply pass to our service routine without any additional comparisons. In both our state machine and overall bot design, we desired to go as simple as possible, since that ideally leads to fewer errors and a simpler state machine.

Certain course layouts caused our bot to either drive off the field or make two loops and not meet the time requirement for the course. As these scenarios became apparent during our testing phase, we adjusted our state machine accordingly. We implemented a timer for when there has not been a transition for more than 4 seconds. Similarly, we added transitions when there has been a tape event in any state so that we do not accidentally drive off the map.

# Conclusion

Overall, this project has been a tremendous learning experience for every team member involved. Some concepts that may not have been fully understood from the previous labs were solidified into our knowledge. We all now understand the difficulty of implementing a design into practice, as it will almost always work differently than how you expect it to be.

Working in a group of 3 was a valuable experience in and of itself - coordinating times out of our busy schedules to overlap the time spent in lab, being responsive to messages nearly 24/7, and setting out clear goals throughout a 4 week period. Understanding the strengths and weaknesses of each partner and working in a fluid way to maximize productivity was something that we were able to achieve quite quickly. The team spirit was high throughout the project and we had a great time whenever we were working on the robot. We prioritized having a very friendly working environment, which is not to say we did not stay on track. We did end up running into multiple roadblocks that set us back pretty heavily - such as having our tape sensors malfunction for 3 straight days, our UNO catching fire, and fuses being popped left and right (sorry). However, regardless of all of these setbacks our team still had an amazing time grinding through the project, all the way from the very beginning deciding the layout of everything to pulling 5 days of consecutive all-nighters together. The project was a fun, crazy, and worthwhile experience that we will all cherish and look back on with pride and ask ourselves how we were able to sacrifice our entire social lives, sleep, and sanity to create a (marvelous?) feat of engineering.

Thank you to Professor Elkaim, Max Dunne, Pavlo, Kyle, and all the tutors who made this class a wonderful experience.