Final Project Report

ECE 118: Introduction to Mechatronics

Scott Oslund
Jaden Allbritton
John Madden

September 19, 2022

Contents

1 Introduction

The culmination of ECE 118: Introduction to Mechatronics is a four week long project to design a bot capable of meeting a given challenge. This challenge incorporates aspects of all of the previous labs in this class, including, software and state machine design (lab 0), sensors and electrical design (lab 1), Mechanical design and perfboarding (lab 2), and actuator control (lab 3). All of these skills are needed to complete the final challenge, making this final project the perfect capstone to the mechatronics class.

This document in detail the challenge we were given and our strategy for completing the challenge. It will go into great detail on our bots mechanical and electrical design as well as our software for controlling the bot. Lastly, it will cover our process of testing the bot and the results our bot achieved. In short, it will provide an overview of our team's (team 15) results for the final mechatronics project.

1.1 Project Specifications

The task the robot had to complete was autonomously navigating a field to score ping pong balls into towers. The field consists of a square outlined by black tape. In the center is a cross of tape. The bot must stay within the outer bounds of the field. If the bot is more than half way off the field it is disqualified.

Inside the field, are towers with three faces each. Each face has three holes cut out for depositing ping-pong balls. One hole with have back tape underneath it on each side. One of the three sides will have a track wire running along it with an alternating current with a frequency of 25 Kilohertz. The hole in the tower with the black tape underneath and with the track wire along that side is the hole the bot should try and score in. Each tower is equipped with a IR beacon at the top to allow the bot to find the towers. Lastly, also on the field is a "dead-bot" which is simply a black cube to emulate a broken down bot.

To achieve minimum specifications, our bot had to navigate through the field and score in the correct hole on two separate towers for at least two out of three randomly generated field layouts. For beer checkoff, we had to prove our bot capable of scoring on on three towers for a randomly generated field a week before the final project deadline.

1.2 Team Philosophy and Project Strategy

In designing our bot, we took a strategy of prioritizing simplicity in our bot. Our thinking was that if we designed a extremely complicated bot and ran into a design flaw or found out a system did not function as well as we expected, we would be in trouble. We would have to make major modifications to the bot and potentially not finish the competition in time. For this reason, we wanted to design a simple bot capable of meeting min-spec requirements first. Then, we could potentially make upgrades to get beer checkoff.

2 Mechanical Design

Mechanical design was the first part designed as everything else built upon this base. The software design portion was determined to be the most time consuming task, therefore a working mechanical prototype was needed as quick as possible. All of the mechanical design of the robot was done in Solidworks and designed such that it could be lasercut on MDF.

It was determined that the biggest challenge would be following the side of a tower which drove the early design. The robot was created with a circular base powered by differential drive which would allow the robot to tank turn without any translation. This way once the robot triggers the from bumper it can tank turn in place to align perpendicular to the face of the tower. The robot was broken into multiple layers to allow modularity and ease of construction. The layers were fixed thickness as constrained by $0.2in$ MDF sheets. The bottom layer was considered `layer0` and the layer that the bumpers are towers are mounted on was `layer1`. Both `layer0` and `layer1` were made to be the same shape. The Uno32 stack, L298N H-Bridge, DC motors, and battery were all mounted on `layer0`. For the drivetrain, generic DC 12V 40 RPM gear reduction motors were initially used. Based on the CAD model of the motors, mounting holes were made in the motor mounts that could be cut with the laser cutter. For the wheels, $76mm$ inline skate wheels were chosen for their price and prior experience. Multilayered mounts made of MDF were made to mount the wheels to the motor shaft. The remaining components also had CAD models created to verify that the components would physically fit in the robot but mounting holes were not added to allow for flexible placement. Similarly slots were cut in the at the front, middle, and rear of the robot for tape sensors to be mounted in a range of positions.

The beacon detector, track wire detector, bumpers, tape sensor and launcher were mounted on `Layer1`. The sensors had bounding box models to ensure they would physically fit. The bumpers were a challenge as they comprised of many different interdependent parts. The four bumpers were split between the horizontal and vertical axis of the robot for a total of 4 bumpers. Each bumper had two individual switches resulting in eight individual switches that needed to be wired. The moving part of the bumpers had a slot that allowed for two degrees of freedom, a single axis of rotation and single axis of translation. The degrees of freedom allowed for three types of bumps to detected: left/right switches triggered individually and both triggered simultaneously. The dimensions for clearances were decided when the bumpers are depressed. A model of the switch in the triggered state was create specifically for designing the bumpers. Despite the complicated design, the bumpers were a overall success in their mechanical functionality.

Unlike the bumpers, where there was careful planning in CAD, the launcher was almost all prototyped with trial and error using foamcore. The original design had planned on using foam core unlike MDF motorized base, therefore could be rapidly prototyped as the material was cheap and easy to manipulate. Due to the time constraints, a gravity fed launcher was chosen for its simplicity. A cardboard paper towel roll was used as the ramp that the ping pong balls rolls down into the tower. The release of ping pong balls was controlled by a MG966 servo with a foamcore claw. The launcher was mounted on a tower constructed for a previous project.

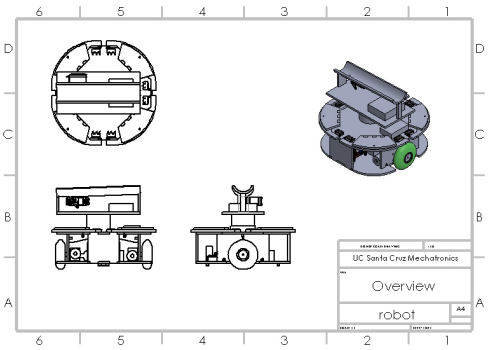
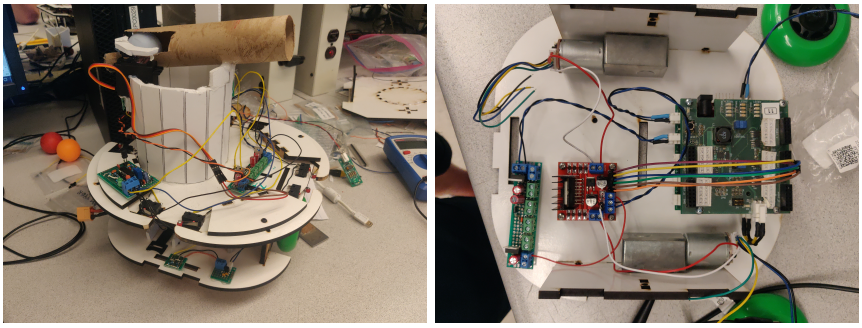


Figure 1: CAD drawings of the first iteration of the robot.

There was several ways that the physical construction of the robot did not match what was designed in CAD. The biggest difference was how the motors were mounted. In Solidworks, the mirror tool was used for the motor mounts resulting in the motors having the same orientation. When the robot was being physically assembled, one of the motors had to be flipped 180° along the drive shaft with respect to the other motor. The difference is shown between in Figure 1 and Figure 2. Because of this, the electronics on `layer0` had to deviate from the planned placement. A drill press was use to create mounting holes for the updated positions. On `layer1` the electronics boards were placed ad-hoc and hot glued in place as it was a quick and relatively non-permanent way of mounting components. The MDF wheel mounts had too loose of tolerances, where the torque of the motors would cause the driveshaft to freely rotate without rotating the wheel. Instead, mounts were 3D printed allowing for greater range of designs over the fixed thickness MDF.



(a) Layout of components on `layer0`. (b) Fully built mechanical design.

Figure 2: Pictures of the first iteration of the robot.

After starting the software design, it became apparent that a few parts needed to be redesigned to match the needs of our state machine. The decision of having a circular base with all circular bumpers did not allow for effective "wall hugging". The physical switches

for the bumpers were too close together not giving enough accuracy to drive parallel to the wall. To help with this the left back bumper was redesigned to have a flatter face towards the wall. The modularity of the design allowed for an easy replacement of the bumper. On the launcher mechanism the paper towel roll was being worn down through testing, so flexible tubing was placed to extend the ramp and allow for flexibility when wall hugging around towers. This iteration of the design was the ones used to achieve both minspec and beer checkoff. Functionality was prioritized more than anything else, with the looks being able to be adapted after proof of concept. These design changes were reflected in an updated CAD model shown in Figure 3.

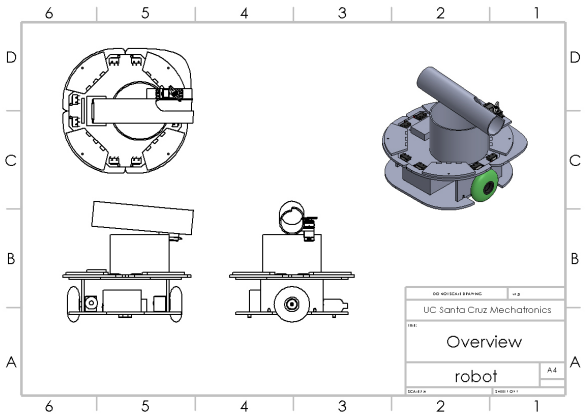
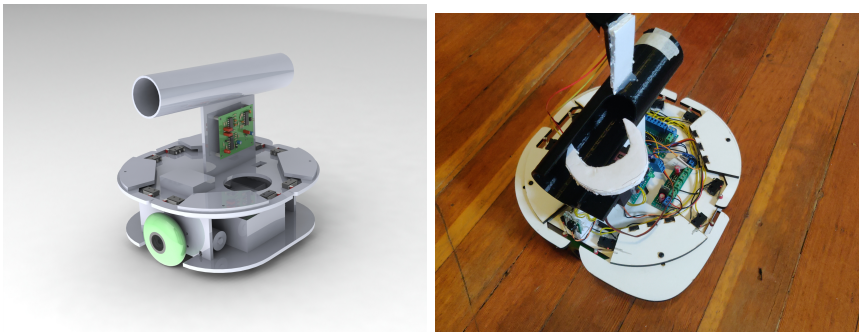


Figure 3: Updated CAD drawings with foamcore tower and redesigned left rear bumper.

In the final iteration only included cosmetic changes to the tower and launcher mechanism. The tower was replaced by a single support cut out of MDF. On top of the support the ramp was replaced by 3D printed tube with a mounting bracket for the servo as a more permanent design. A hole was also cut in `layer1` to allow for better wiring management between the Uno32 stack and the sensor board.



(a) CAD render of the robot. (b) Fully assembled robot.

Figure 4: Final designs of the robot.

3 Electrical Design

3.1 Beacon Detector

The beacon detector was used to detect the direction of the tower. It had to detect a $2kHz$ IR signal from the beacons mounted on top of the tower. The beacon detector needed to detect the signal from $< 6ft$ while rejecting $1.5kHz$ and $2.5kHz$ signals from any range. The block diagram of the beacon detector circuit is shown in Figure 5.

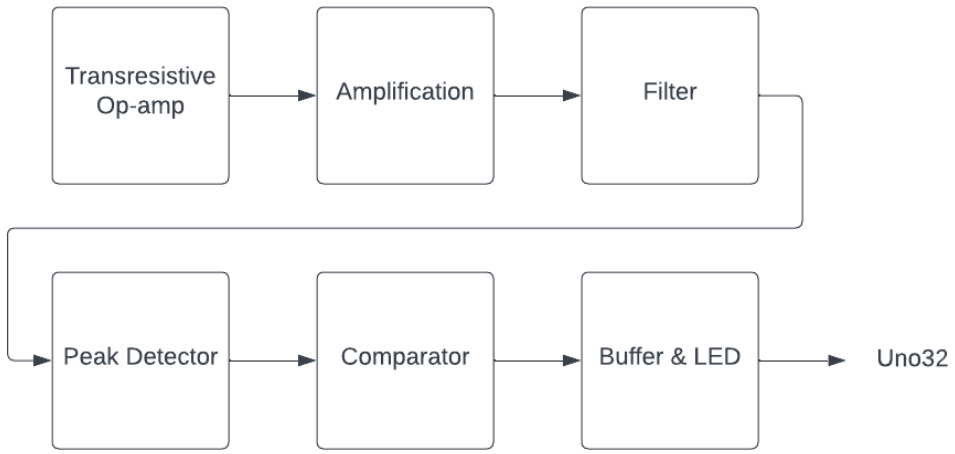


Figure 5: Block diagram of the beacon detector circuit.

The transresistive op-amp stage converted the IR signal to an analog signal. The IR phototransistor was current biased to allow voltage to vary with the signal. Both $1k\Omega$ and $100k\Omega$ resistors were tried, and $1k\Omega$ was chosen as it resulted in a greater magnitude on the signal. The input voltage was biased at $1.65V$ to put the offset in the middle of op-amp supply rails. This allowed the maximum peak-to-peak voltage swing for the signal. This was applied to all the stages to get the most gain out of the op-amps.

The amplification stage was made of 4 stages of 21 gain non-inverting op-amps resulting in a total gain of 194481. The gain could not be all combined into a single op-amp due to open loop gain limitation of the `MCP6004` quad op-amps. High pass filters were applied to each stage to prevent DC noise from being amplified and ruining our signal. The signal needed to amplified to a suitable to the input voltage of the Uno32.

The filter stage used a bandpass Sallen-Key filter attenuating signals other than the desired $2kHz$ sine wave. The attenuation of the signals at frequencies $2kHz$, $1.5kHz$, $2.5kHz$ was $-0.1dB$, $-9.92dB$, $-10.1dB$ respectively. The circuit was designed within KiCAD and spice simulation of the circuit was used to generate a bodeplot shown in Figure 6.

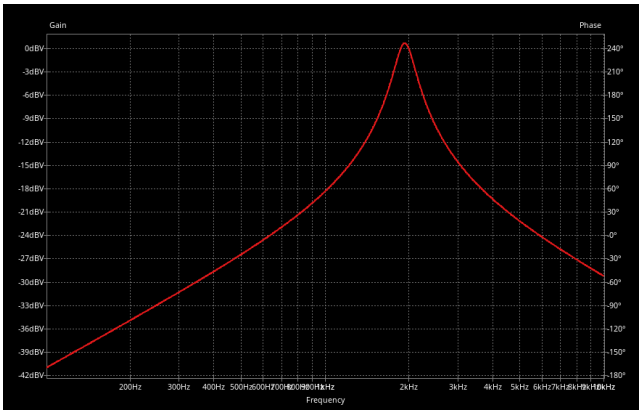


Figure 6: Bodeplot of the filter stage.

The peak detector and comparator converted the analog signal to a pure digital signal. The peak detector "holds" the signal at the maximum voltage. During short time frames, the signal was stable, but over large time the peak detector would leak current over time to allow the peak detector to adjust whatever the maximum voltage in the current time frame. The comparator was a Schmitt trigger that allowed for the adjustment of hysteresis and converted the analog signal to digital signal. The signal was converted to allow for use with Uno32 digital IO pins. At the comparator output, a pull-down resistor in series with a LED was used to give a visual indication of the signal.

The circuit went through various design phases. This circuit was first prototyped on a breadboard before being transferred to a soldered. For the final design a PCB was ordered based on the protoboard design. The soldered board is shown in Figure 7. Only one iteration of the PCB design was needed as the beacon detector worked the first time it was soldered.

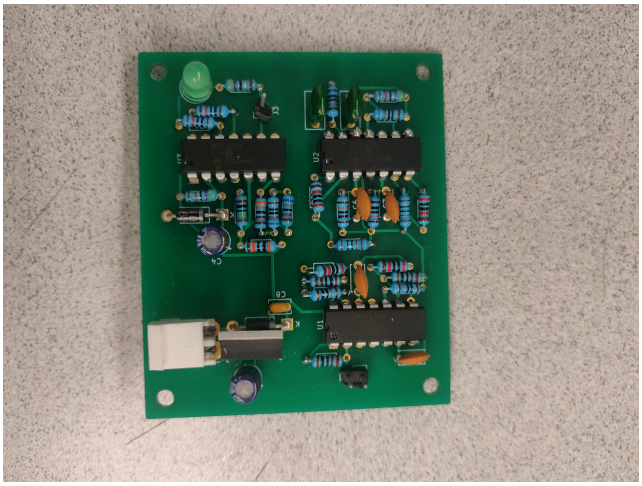


Figure 7: Soldered beacon detector PCB.

3.2 Track Wire Detector

The track wire detector followed a similar structure to the beacon detector, with the process of filtering, gaining, and rectifying an input signal. A block diagram of each stage

can be seen below, and the full schematic can be found in the appendix. The circuit is designed to detect the changing magnetic fields produced by an alternating current flowing through a wire.

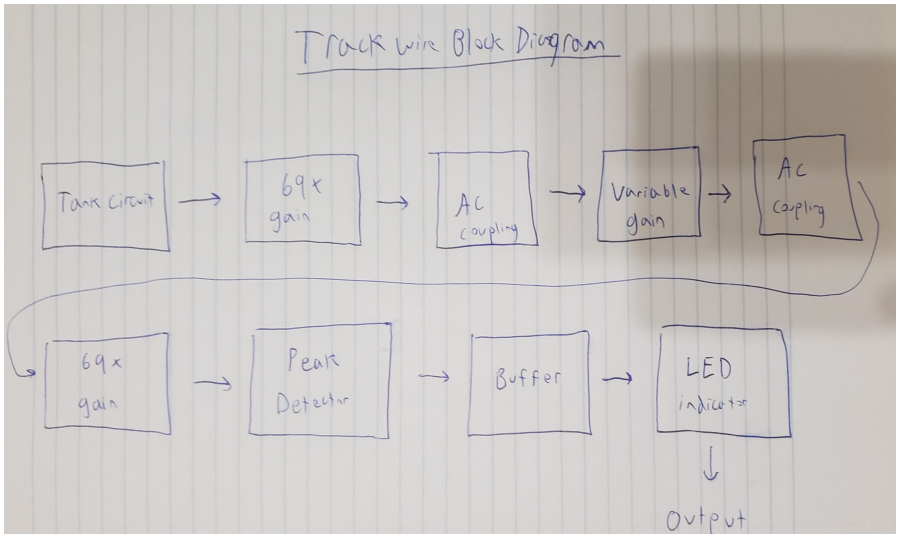


Figure 8: Block Diagram of the track wire detector circuit

The first stage consists of an RLC tank circuit. This module serves the dual purpose of receiving the input signal through an induced emf in the inductor and filtering out noise. By altering the resistor value, the quality factor of the circuit changes, altering the range of frequencies allowed through. We experimentally found that using a resistance of around 20k ohms provided a good quality factor.

The next section of the circuit consists of a gain stage with AC coupling. Using a non-inverting amplifier configured for a gain of 69, the input signal which is likely to be very small at first is amplified. However, this amplification can also lead to amplification of DC offset as well. So AC coupling is introduced after the gain to remove any offset. The AC coupling is essentially just a high pass filter.

Next, is once again another gain stage followed by AC coupling. However, this gain stage is a bit different in that it uses variable gain. Once again, an op-amp is set up in a non-inverting configuration. However, by using a potentiometer configured as a variable resistor in our design, there is a range of gains the stage can have from 2 to 21. This allows the user to easily adjust the sensitivity of the circuit by turning the potentiometer, without the need to desolder or resolder any components. This feature would prove to be very helpful in testing since it allowed for the easy adjustments to make the circuit have the correct range. Once again, AC coupling after this stage in the form of a high-pass filter removes any DC offset. Following this modules is another final 69x gain stage.

With the input signal having being amplified to a appropriate magnitude, it next needs to be rectified to a DC value. To do this, a peak detector stage is added on. This consists of a diode, only allowing current to flow into the peak detector and not back, a capacitor that charges up in response to the input signal, and a resistor to slowly discharge the capacitor in case their is a change in the input signal's amplitude. All together, this module takes the

input signal and outputs a rectified DC signal held high near the peak of the input.

The final part of this circuit is a buffer, the allow the signal from the peak detector to be strong enough to drive components, and an LED in series with a current limiting resistor. The LED is used to visually see when a track wire is detected. The output of the buffer is then the output that is attached to one of the microcontroller's analog to digital pins.

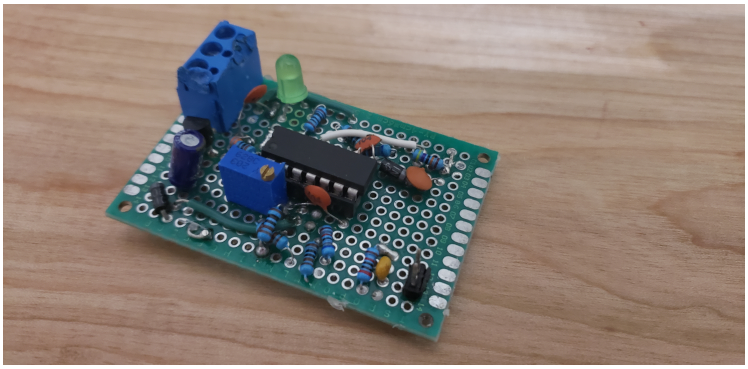


Figure 9: Soldered track wire detector

3.3 Tape Detector

Our tape circuit circuits were fairly simple to design. They consist of an IR emitter LED attached to power and ground with a current limiting resistor. We found a resistance of 47 ohms to be sufficient for this purpose. In parallel to this, we wired a 5.1 kohm resistor and a IR receiver transistor in series connected to power and ground. The output of the sensor is the node connecting the transistor and the resistor.

In our configuration, the transistor functions as a pull down transistor. When it receives the IR signal emitted from the LED, the transistor opens, pulling down output to ground. Therefore, reflective surfaces that appear white, cause the sensor to read out a zero. However, when the sensor is over a less reflective black surface, it less of the signal from the LED bounces back to hit the transistor. Therefore the transistor remains mostly closed causing the output to be pulled to a high value by the 5.1k resistor. In this way, our tape sensors were capable of detecting the black tape on the sides of tower or on the floor of the field.

3.4 Bumpers

Our bumper setup was extremely simple. Using limit switches bought from Amazon, we attached on pin of the switch to ground and the other to a digital input of the microcontroller. By default, when the limit switch was up the switch acts as a short circuit connecting the microncontroller's input to ground, causing it to read a 0. However, when the limit switch is pressed down, the switch behaves as an open circuit. In this case, the microntroller pulls the input pin to a 1 using an internal pull up resistor. Therefore, with this configuration we read a 0 when the bumpers were not pressed down and a 1 when they were pressed down.

Figure 10: Sample limit swtich

3.5 Actuators

The final part of the electrical design are the actuators to cause our robot to complete the given task. One of these actuators is a servo, responsible for releasing the ping-pong ball into the tower. The wiring is simple. It is powered with 5 volts and ground pins. Then, a third pin is given a PWM signal for controlling the position of the servo. This is done by directly connecting a PWM pin from the microcontroller to the servo.

The remaining actuators on the bot are two motors. These are 100 rpm motors each using worm gears. They are controlled through the use of an H-bridge. The H-bridge has an in1, in2 and enable input port for each motor. Then, two outputs for each motor to set the motor speeds. Into the in1 and in2 ports, are digital outputs from the microcontroller. These inputs control the direction the wheels spin in. The enable port receives a PWM signal from microcontroller. This allows us to control the motors speed by setting the PWM's duty cycle. With this setup, each motor can have its direction and speed controlled.

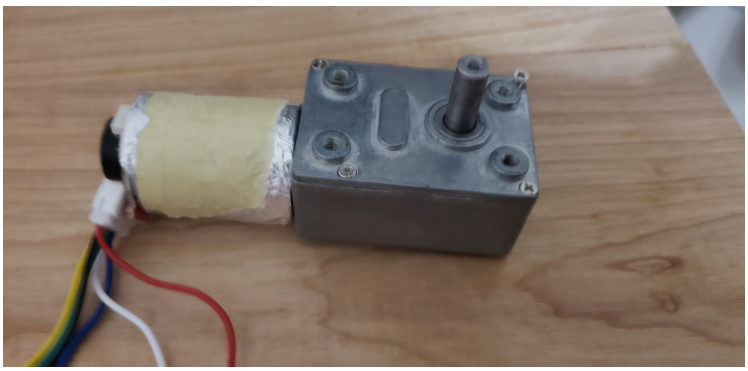


Figure 11: One of the motors used in our bot

For a more complete view of the wiring of the actuators and the other electronics discussed, see the wiring diagram included in the appendix of this report.

4 Software Design

Creating the software for the project was first separated into two tasks. First, creating a sensor and actuator interface for checking for events from sensors while filtering noise and to

drive the actuators with function calls. The second task in terms of software was to utilize the interface to create a full state machine to complete the given task.

4.1 Sensor and Actuator Interface

For each sensor system on the robot, the event checker follows a similar structure. First, read one of the microcontroller input pins to get the current reading of the sensor. If the current reading varies significantly to the previous reading (stored in a static variable) generate an event and post it to the HSM. The sensor reading is then passed through as the parameter of the passed event and the previous reading is updated to the current reading.

The track wire sensors and tape sensors operate by feeding an analog voltage into the microcontroller which is then converted into a 10 bit number using an analog to digital converter. In order to interpret the analog voltage as a digital high or low while accounting for some noise, a software hysteresis is used. The hysteresis has a defined upper and lower bound with a deadzone between the two. When the upper bound is crossed, if the previous state was a low, a high event is generated and if the lower bound is crossed while the previous state was a high, a low event is generated. If a reading lies in the deadzone between the high and low threshold, no event is generated. This method prevents noise from causing frequent event generation.

The track wire detector event checker serves as a good example of how the standard event checker functions as well as the use of a software hysteresis. The basic structure in pseudocode is shown below, where Prev is defined as the previous reading of the track wire, U_Thresh is defined as the upper threshold of the software hysterisis and L_Thresh is defined as the lower threshold of the software hysterisis, where the precise values of these thresholds are experimentally determined.

Algorithm 1 Track Wire Event Checker Structure

```
1:  Curr ← Prev;
2:  Reading = ReadTrackWireOutput();
3:  if Reading > U_Thresh then
4:      Curr ← 1
5:  end if
6:  if Reading < L_Thresh then
7:      Curr ← 0
8:  end if
9:  if Curr != Prev then
10:     PostTrackWireEvent();
11: end if
```

The one event checker that slightly varied from this structure was the bumper event checking. The bumpers of our bot are noisy, and pressing them down once can generate a few up and down bumper events. Therefore, we had to devise a way to debounce the noise from bumper presses. One method we learned in class was to check the event bumper event with a service operating on a timer. However, this method would lead to worse response time and would require the extra overhead of using another service and timer.

Instead, we devised a better method for denouncing. We kept inside of our bumper

event checker a static counter variable. Everytime the event checker is entered this counter increments by one. Using this counter, the function only runs if the counter is above a certain value, otherwise, the function returns immediately. Upon an event being posted, the counter resets to zero. The counter is initialized to a value high enough so it is over the threshold.

With this configuration, while the counter is over the threshold the event checker can immediately react to bumper changes and post events without waiting for a timer to go off to check for a new event. However, after posting an event the event checker must wait for the counter to reach a high enough value before it can post events again. This serves as the debouncing, ignoring bumper changes that are likely noise.

Finally, the actuator software design was relatively simple. For the motors, a function was designed to intake two values and set the motor speed depending on each. All the function has to do is set a PWM signal duty cycle based on the value that was input. If the value entered in is negative, the function also switches the voltages input to the motor to allow it to turn in reverse. For the servo controlling the ball release, similarly, a simple function to set pwm duty cycle was all that was needed. For convience we also defined two macros LOCK and RELEASE to define the duty cycles to put the servo in the right position to hold the balls and the right position to move to let the ball go.

4.2 State Machine

With the robot mechanically and electrically assembled, and the sensor and actuator interface done, the next step was to design the robot's state machine so it could intelligently combine all aspects of its design into a working robot capable of meeting project specifications. To do this, we designed a hierarchical state machine, with a top level state machine that calls sub-state machines to respond to events.

4.2.1 Top Level State Machine

Our first, top level state machine provides a good overview of the robot behavior and demonstrates the manner in which each sub-state machine is called. The top level consists of four states, each being their own sub-state machine, responsible for part of the robot's actions. These are the find tower, circle tower, tape avoid, and release sub-state machines.

Find tower is the initial state the bot enters, it is responsible for locating a beacon and driving towards it. This state can be left when a bumper press is detected, indicating the bot has hit either a tower or a dead-bot. The bot then transitions to the circle tower state. This state is responsible for the bot wall-following whatever it hit. There are three conditions for exiting this state. First, if a tape event is received indicating the robot is looking at black tape with the side tape sensor while the track wire sensor is also outputting a very high voltage, there is a transition to the release state. Next, if a tape event indicating the robot is looking at tape from the bottom of the robot, the bot enters into tape avoid to prevent it from running off the end of the field. Lastly, if a move-on event is occurs (posted from the deposit sub-state machine) the bot transitions to the find sub-state machine to look for a new target. How this event is generated and the conditions for it are discussed in the

deposit state machine section.

The avoid tape sub-state machine causes the robot to continue wall hugging the object from the deposit state, but in this case it wall-hugs in reverse, moving the opposite direction around the object since it detected tape when moving in the previous direction. There are two conditions for exiting this state. First, if another beacon is detected and the bot has already scored (remembered by a static variable) the bot transitions to the move-on state. Otherwise, if another tape event goes off and the parameter indicates that it was a bottom tape sensor, the bot moves to the circle tower state again.

The move-on sub state machine, is a simple routine for the robot to follow to get itself away from the current tower and towards the next tower. It causes the robot to drive away towards the next tower it spotted (since this state is entered when a new beacon is found) before moving back onto the find tower state. The condition for exiting this state machine is an ESCAPED event which is posted by the Move-On state machine itself when it finishes its routine. The next state is then the find tower state.

Similarly, the Release sub state machine is a short routine for releasing the ball into the tower and jiggling back and forth a little bit to make sure the ball falls in. This state is exited upon the SCORED event being received which is posted by the release state machine when it finishes its routine. The bot then moves onto the circle tower state again.

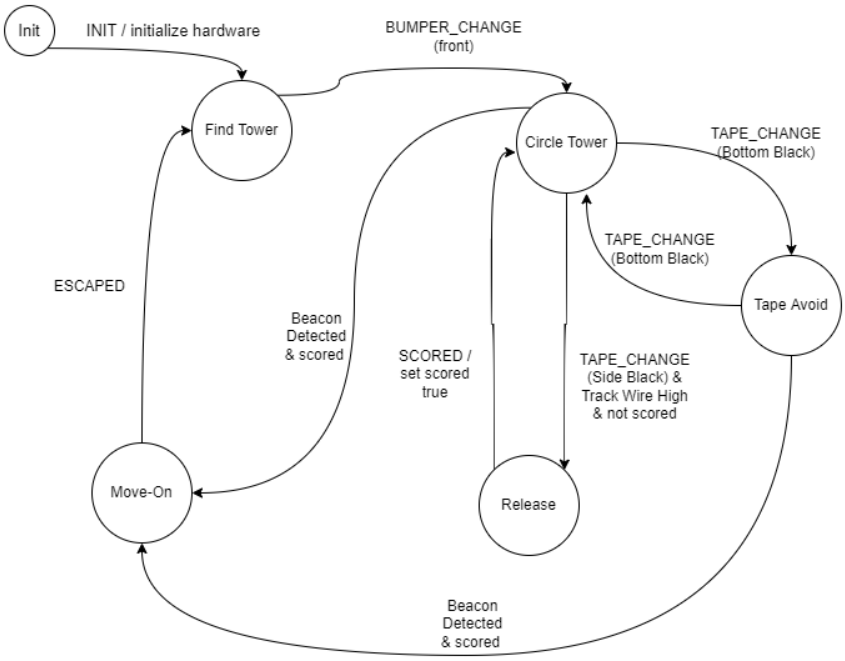


Figure 12: Top-Level Diagram of Final State Machine

4.2.2 Find Tower Sub State Machine

The find tower sub state machine is the first state entered by the bot. Here, the bot begins by spinning counter clockwise in place, scanning for a beacon. When a beacon detected event happens, the bot moves to a state in which it moves forwards while tilting to the left. When the bot no longer detects the beacon due to too much of a tilt to the left

(indicated by a beacon changed event with a parameter indicating the beacon is not found) the bot then transitions to moving forward with a tilt to the right. This process keeps the tower centered so the bot always is moving towards it.

On the top-level, this state machine is exited when there is a bumper event indicating the tower was hit.

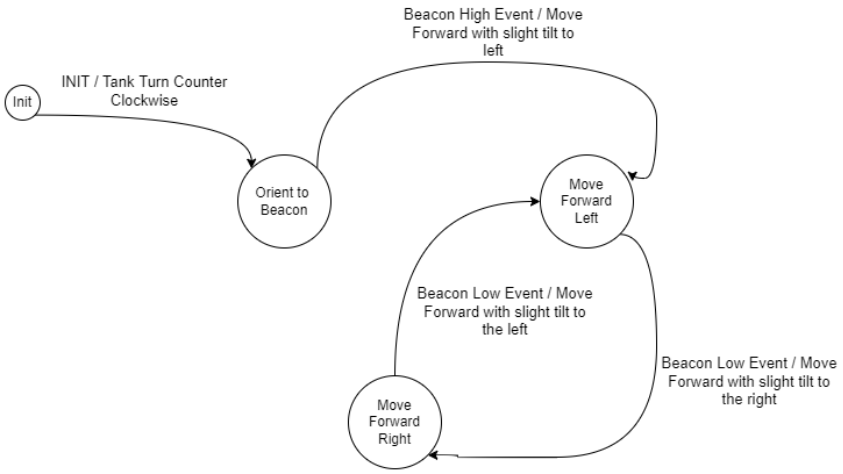


Figure 13: Diagram of the Find Tower Sub State Machine

4.2.3 Circle Tower Sub State Machine

The next and perhaps most important state machine is the Circle tower sub-state machine. This causes the bot to wall-hug around the tower while remaining close enough to the wall to be able to release the ball into the tower if all of the conditions are met.

Upon a font bumper being pressed and this state being entered, a set routine is run. The bot begins by aligning itself against the side of the wall by tank turning clockwise until a bumper on its side back is pressed. Once this is pressed, it indicates the bot is flush against the side of the wall and it can enter the wall hugging state. In this state, the bot moves forwards along the side of the tower with a tilt to the left to make sure it remains against the side of the tower.

If there is a Turn Timer timeout without a bumper being pressed, the bot assumes it has not had a bumper pressed because it has passed beyond the side of the tower. In this case, it moves to the Make Turn state where it makes a sharp turn while pivoting on one wheel. A timeout or front bumper being pressed (indicating the bot is now against the side of the tower again) causes the bot to transition back to wall hugging. Otherwise, if the Turn Timer expires and a second Realign Timer expires in the wall hug state, the bot goes to the realign state. Here, the bot tank turns clockwise until a timeout occurs or until the side back bumper is pressed. This causes the bot to realign itself to be flush to the side of the tower in the same way it was originally when first entering the wall hugging state. This routine prevents any error from propagating and growing since the bot resets itself.

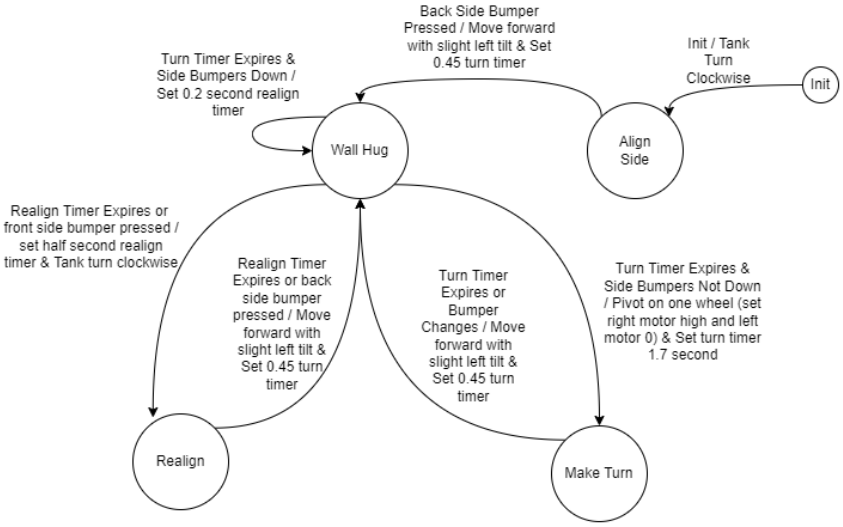


Figure 14: Diagram of the Circle Tower Sub State Machine

4.2.4 Release Sub State Machine

This next state machine takes over when the bot has detected the necessary conditions to score on the tower. These conditions are the track wire being detected and black tape being detected. First, upon entering this sub state machine, the servo opens and a timer is set. The bot stays still for 2 seconds in the Release state to allow the bot to fall into the hole. Next, once the timer expires the bot moves forwards for half a second. Then it moves backwards for half a second. This movement forwards and backwards is meant to ensure the ball will fall into the tower.

Once this short routine ends, the state machine is reset and a custom SCORED event is posted to the top level state machine to indicate the routine has finished.

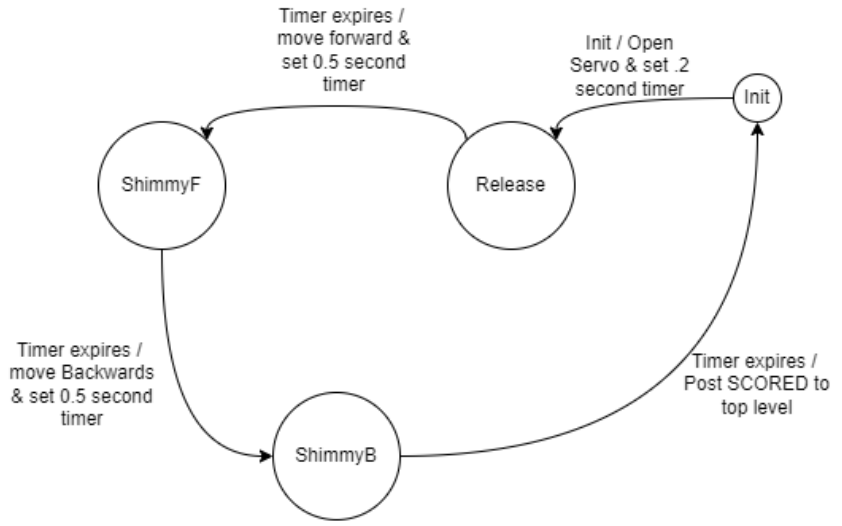


Figure 15: Diagram of the Release Sub State Machine

4.2.5 Move-On Sub State Machine

This sub state machine consists of another simple routine for escaping from the wall following of a tower so it can move on to a new tower. This simple state machine first consists of the tower backing up while slightly turning right for half a second. Next, the bot drives forwards with a slight right tilt for 1.2 seconds. The slight right tilt is meant to ensure the bot does not get stuck against the tower it was previously wall-following since it uses its left side to wall follow. After this sub state machine, a new tower is found by transitioning to the Find Tower sub state machine.

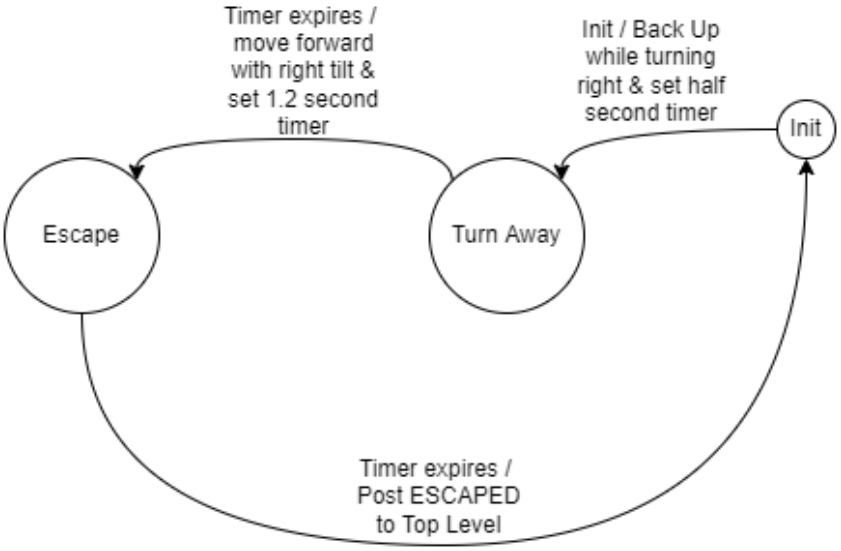


Figure 16: Diagram of the Move-On Sub State Machine

4.2.6 Tape Avoid Sub State Machine

The final state machine is for wall following backwards. This is needed for the edge case in which a tower is against the side-tape of the arena and the bot needs to avoid running off the edge. This sub-state machine causes the bot to wall-follow backwards. It is an exact replica of the Tower Circle state machine with the motor directions being reversed. For more information on each state, see the Tower Circle sub-state machine description above.

We chose to make this state machine be a simple replica of the previous state machine to keep things simple and to prevent needing to design a whole separate wall following algorithm for moving backwards. Since the bot is not symmetrical, wall following in reverse with the same algorithm leads to the bot not hugging the wall as tightly as it does when moving forwards. However, since the bot never attempts to score while moving backwards some imprecision is acceptable.

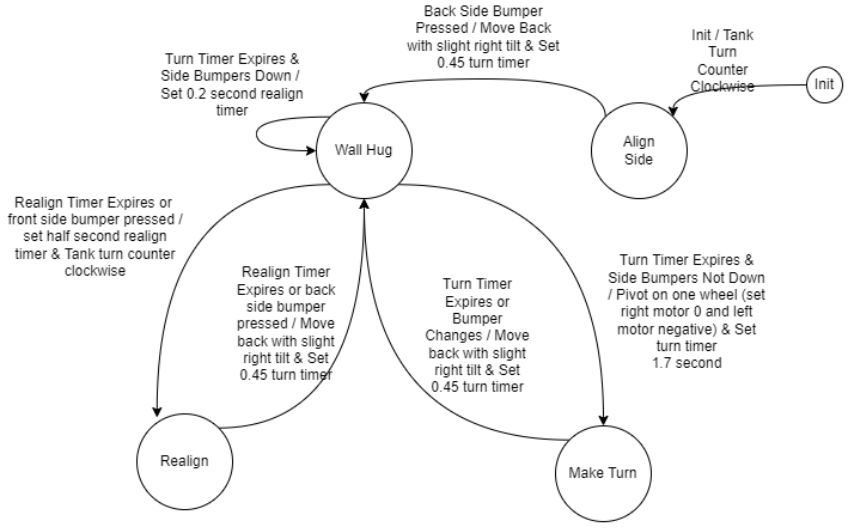


Figure 17: Diagram of the Tape Avoid Sub State Machine

5 Initial Testing Hiccups

5.1 Bumper Design

As mentioned in Section 2, our first draft of our assembled robot had curved bumpers. The issue arose with our tower circling strategy. The competition requires the bot to find the correct wall and be aligned (parallel in our case) with it in order to score. Wall hugging with circular bumpers meant that the bot would turn a greater angle forwards backwards in order to engage the bumpers to continue wall hugging. This meant our bot would never be parallel with the tower walls as it circles the tower, unless our align state was perfect. To rectify this quickly, we hot glued a piece of foamcore to the back left bumper so that it would engage when our bot was roughly parallel with the tower wall. Since the bot is changing it's angle of attack less to engage its bumpers, it is more flush with the wall. The protruding bumper also allows the bot to corner consistently without ruining the angle of attack coming out of the turn. Additionally, our bumper switches were not powered. This is relevant because when you connect nothing to an enabled analog input of the Uno32 I/O shield, the pin will "float" - or read random values from 0 to 1023. When a switch was depressed, the circuit would complete but with no power running through it. So then the pin would float. Our software reads a bumper trip as a change in value, so we still detect an event, it just can be zero even though the bumper was depressed. The result is random spamming of bumper events that was handled with denouncing to the tune of hundreds of samples.

5.2 Wire Detection Strategy

The track wire detector allows the bot to detect the correct side of the tower to score. Originally, our bot was designed to have two track wire detectors. This provides increased information for the bot to align parallel with the wall and shoot into the indicated hole

perfectly. Thus the position of the inductors on the track wire detectors matters greatly. We adjusted the variable resistors to detect the track wire at the same distance. However we encountered an interesting issue. In order for our bot to score, the tape sensor needs to detect the reflective indicator tape and also detect the track wire. Our wallhugging strategy would put one inductor in range as it turned into the tower, and move the other inductor out of range. The reverse would happen as it turned away from the tower to engage the back bumper. Increasing the gain of both detectors to circumvent this would cause the detectors to trip while not on the correct side of the wall. We overthought it for awhile until we realized we didn't need two track wire detectors. We simply removed one circuit and positioned the inductor from the remaining detector directly under the launcher barrel, and connected it back to the detector with jumper wires.

6 Results

6.1 Min-Spec

The completed robot was then put to the test as we sought checkoff. First, we needed to complete minimum specification checkoff in order to complete the class. To achieve min-spec, our bot had to be capable of scoring on different towers for two out of three randomly generated fields. Our first attempts to meet minimum specifications were on Tuesday May 24th. However, however, our bot was too inconsistent to be able to meet min-spec. It would sometimes complete a run but it failed to meet the two out of three standard. It would sometimes fail because it would be too far away from the wall when it attempted to score. To remedy this, we went back to work on our bot for the next couple days, tuning the exact timer values on our bot for our wall following algorithm to make it hug the wall more tightly.

With adjustments made, on Thursday May 26th we managed to achieve min-spec on our first attempt. We finished with one week and a day to spare before the project deadline. In addition, our bot performed well in the runs, finishing one attempt in under a minute. We were the second group to finish.

6.2 Beer Checkoff

On Friday May 27th, one week before the deadline we also achieved beer checkoff. Our bot was able to successfully score on three towers on a randomly generated field in under two minutes. Although we had faith in our bot, we found this outcome unexpected and were proud of our bot's performance.

6.3 Final Competition

Finally, on Friday May 3rd we had the final competition. In the time between beer checkoff and the final competition, we had disassembled our bot, completely redesigned the launching tower, and redid all of the wiring. While this made our bot far neater and better functioning, it left us with no time to test the altered bot before competition. This lead to a wiring mistake that we did not discover until the competition began.

Our motors turned out to be wired backwards. So when our bot attempted to drive to a tower, it instead drove backwards off the field leading to our disqualification. Despite this unfortunate outcome in the competition, we are still very proud of our bot and its achievements. Finishing min-spec well ahead of schedule and getting beer checkoff which few other teams were able to do proves the capabilities of our bot. If we had time to test our wiring before the competition, we are confident our bot would have performed well.

7 Conclusion

ECE 118: Introduction to Mechatronics is undoubtedly the most difficult course offered at UCSC. Our team spent many long hours in the lab working on our final project and completing labs. However, we have gained valuable skills and practical experience in designing mechatronics systems that cannot be gained from other courses. During labs, we learned how to design state machines, build sensor circuits, do CAD and mechanical design, and control actuators. In the final project we put all of these skills together to build our bot.

Our hard work in this class clearly paid off. Our bot succeeded in accomplishing the given challenge and we achieved both minimum specifications and the beer check-off challenge. More importantly, we all feel that we have learned a great deal from this class. We will carry the skills we learned for the rest of our lives and our engineering careers.

A Early Robot Design Sketches

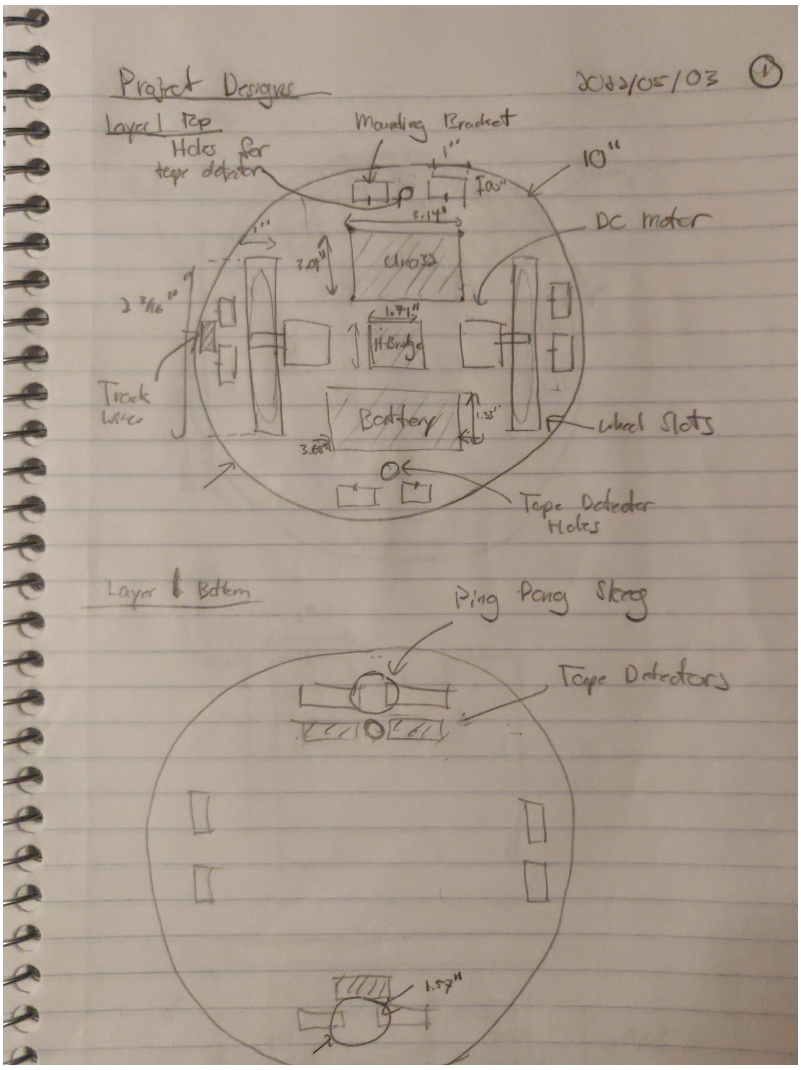


Figure 18: Initial Sketches of Bot's bottom layer

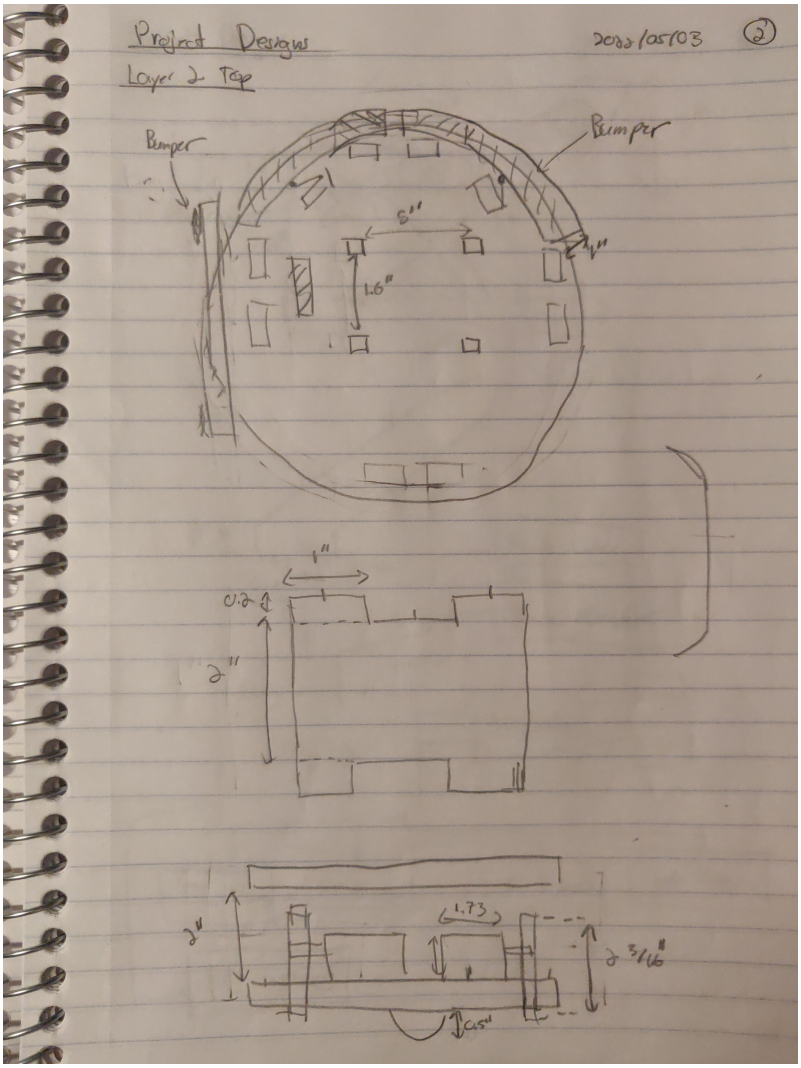


Figure 19: Initial Sketches of Bot's side and top layer

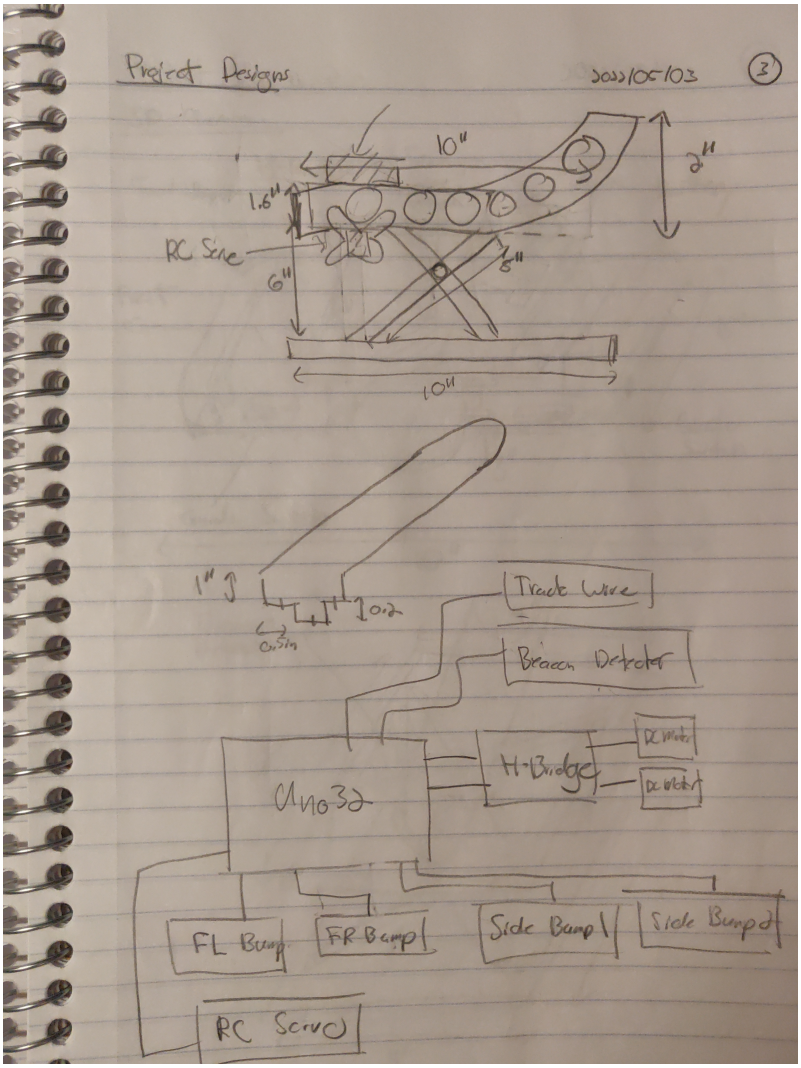


Figure 20: Initial Sketches of Launcher Mechanism

B Wiring Diagram Diagram

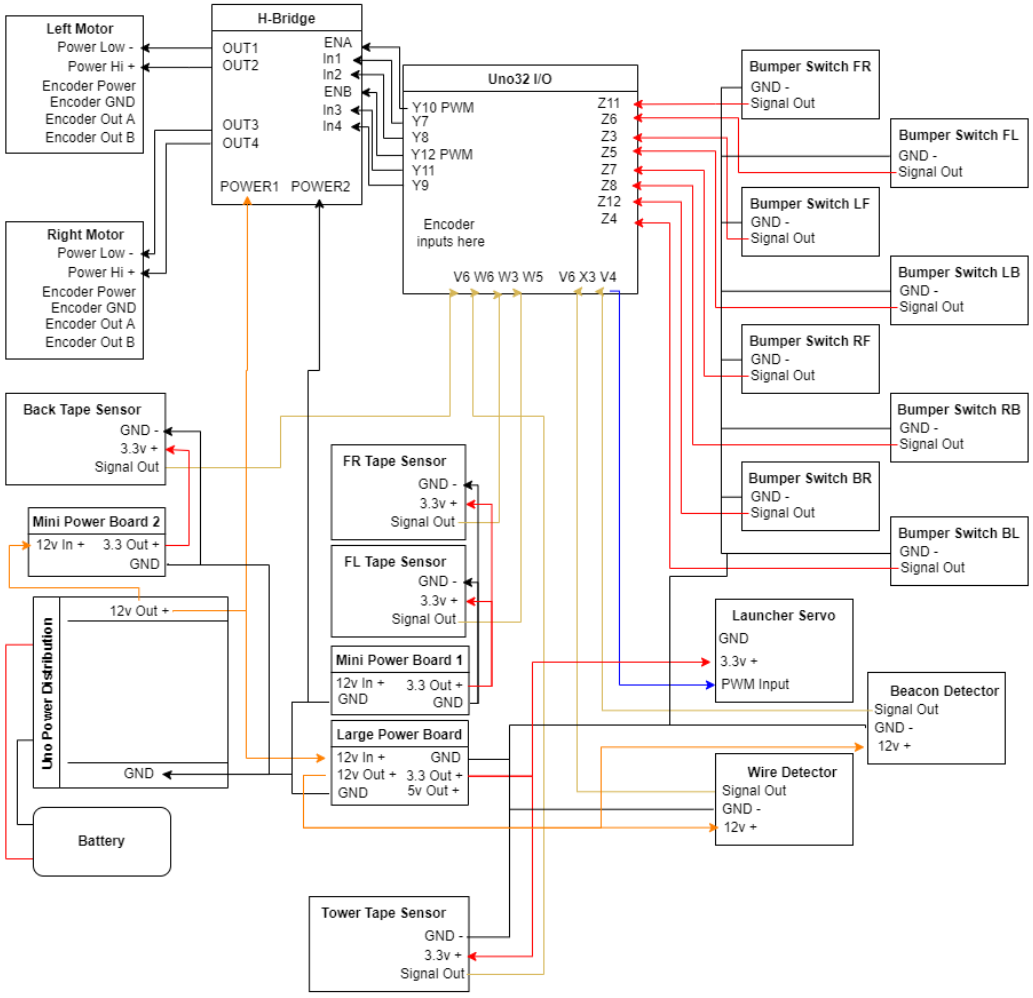


Figure 21: Wiring Diagram of our Full Robot

C Beacon Detector Schematic

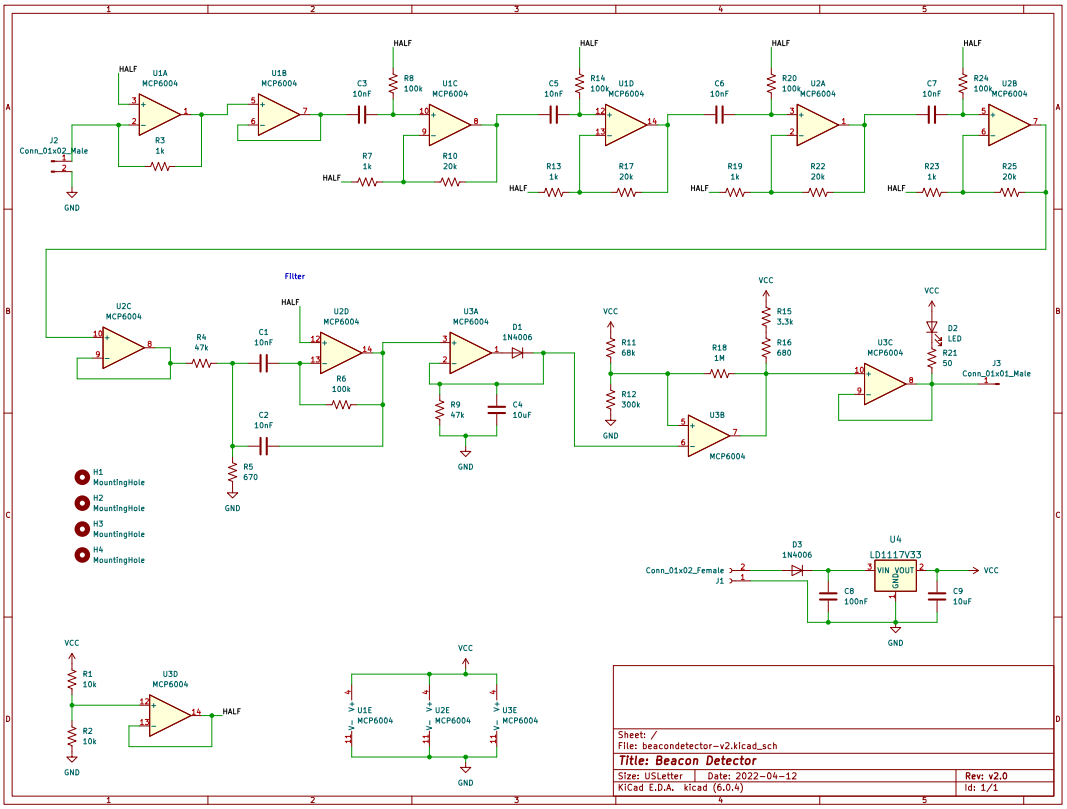


Figure 22: Schematic of beacon detector circuit

D Track Wire Detector Schematic

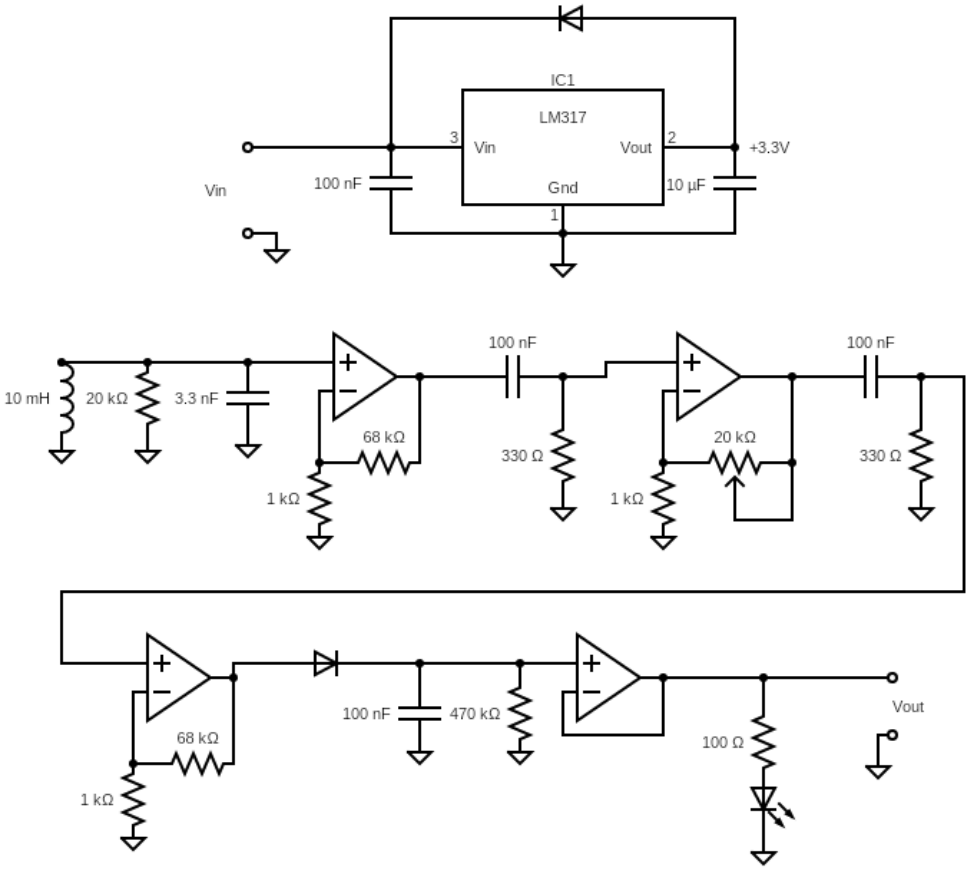


Figure 23: Schematic of track wire detector circuit

E Tape Sensor Schematic Schematic

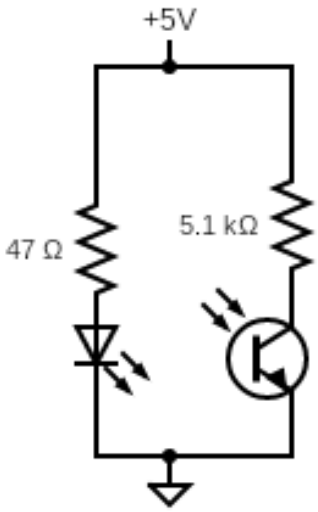


Figure 24: Schematic of the tape sensor circuit

F Bill of Materials

	A	B	C	D	E	F
1	Item	Unit Price	Quantity	Total Price	Buyer	Source
2	HiLetgo 10pcs Micro Limit Switch	$5.99	1	$5.99	John	https://www.amazon.com/gp/product/B07X142VGC/ref=ppx_yo_dt_b_asin_title_o00_s00?ie=UTF8&psc=1
3	72mm Inline Skate Wheels x4	$13.93	1	$13.93	John	https://www.amazon.com/gp/product/B01GOEJ5SI/ref=ppx_yo_dt_b_asin_title_o00_s00?ie=UTF8&psc=1
4	MDF Sheet	$6.14	1	$6.14	John	BELS
5	IC, regulator, LDO, LD1086V33, TC	$1.56	4	$6.24	John	BELS
6	Gear Reduction Motor, 100 RPM	$17.68	2	$35.36	John	https://www.amazon.com/gp/product/B08BLB34R8/ref=ppx_yo_dt_b_asin_title_o00_s00?ie=UTF8&th=1
7	screw, phillips panhead, 4-40, 3/8"	$0.12	20	$2.40	John	BELS
8	Servo	$5.00	1	$5.00	John	John
9	terminal block, solder, screw term, (	$0.50	5	$2.50	John	BELS
10	stand-off,hex,aluminum,female fem	$0.28	7	$1.96	John	BELS
11	screw, pan head slotted, 4-40, 3/8"	$0.12	4	$0.48	John	BELS
12	screw, phillips panhead, 4-40, 1-1/4	$0.12	8	$0.96	John	BELS
13	potentiometer, 15 turn, 20K	$1.50	3	$4.50	Scott	BELS
14	terminal block, solder, screw term, (	$0.50	8	$4.00	Scott	BELS
15			Total	$89.46		

Figure 25: Final Bill of Materials

G Link to sample min-spec run

https://drive.google.com/file/d/1AJXHC2hKqiLcnmvW3L6Xb09bTHp5HCQi/view?usp=sharing