ECE118 Final Project Report

Alexis Garado
Neili Hu
Matthew Kaltman

December 11, 2022

Table of Contents

Contents

1 Project Overview

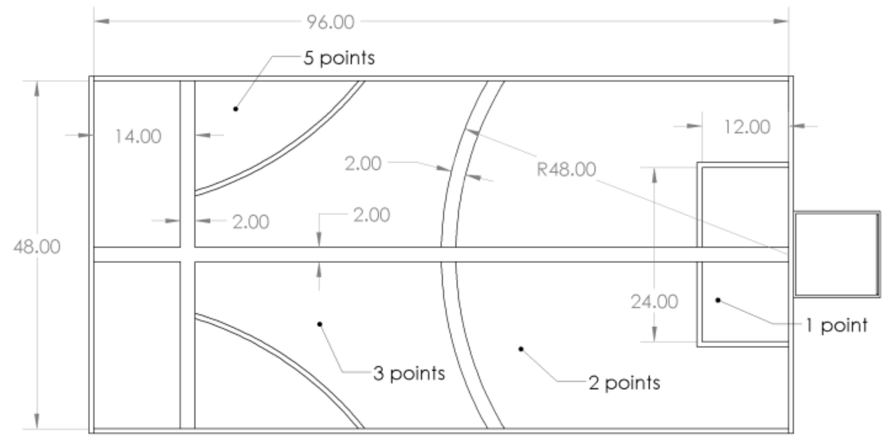

Figure 1: Scaled Drawing of the Field Including Zone Point Values and Dimensions

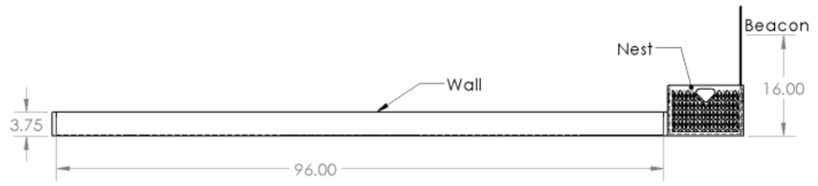

Figure 2: Scaled Side Profile of the Field, Including Arena Dimensions

1.1 Project Requirements

- Robot must fit inside an 11"x11"x11" cube
- Robot must successfully score 10 points
- Robot must successfully go through at least one reload cycle
- Robot must resolve collisions with obstacles within 5 seconds
- Team has a maximum spending limit of $150

1.2 Design Overview

For this project it was decided to go for a strategy that utilizes IR distance sensors and two mechanical bumpers in order to navigate the field and resolve collisions, two track wire detectors to notify the state machine that the robot had entered/exited the reload or 5 point zone, and a beacon detector to aim and position the robot before shooting the ping-pong balls with a PVC fly-wheel launcher. This strategy depends entirely on being in the 5 point zone as it required the least amount of movement from the reload zone as well as having the highest point scoring capacity. The beacon detector is raised on a vertically mounted rack and pinion that raises on initialization of the robot and the fly-wheel launcher is dependent on a gravity fed reload tube that pops a ball out under command from a solenoid.

2 Mechanical Design

2.1 Over-arching Design

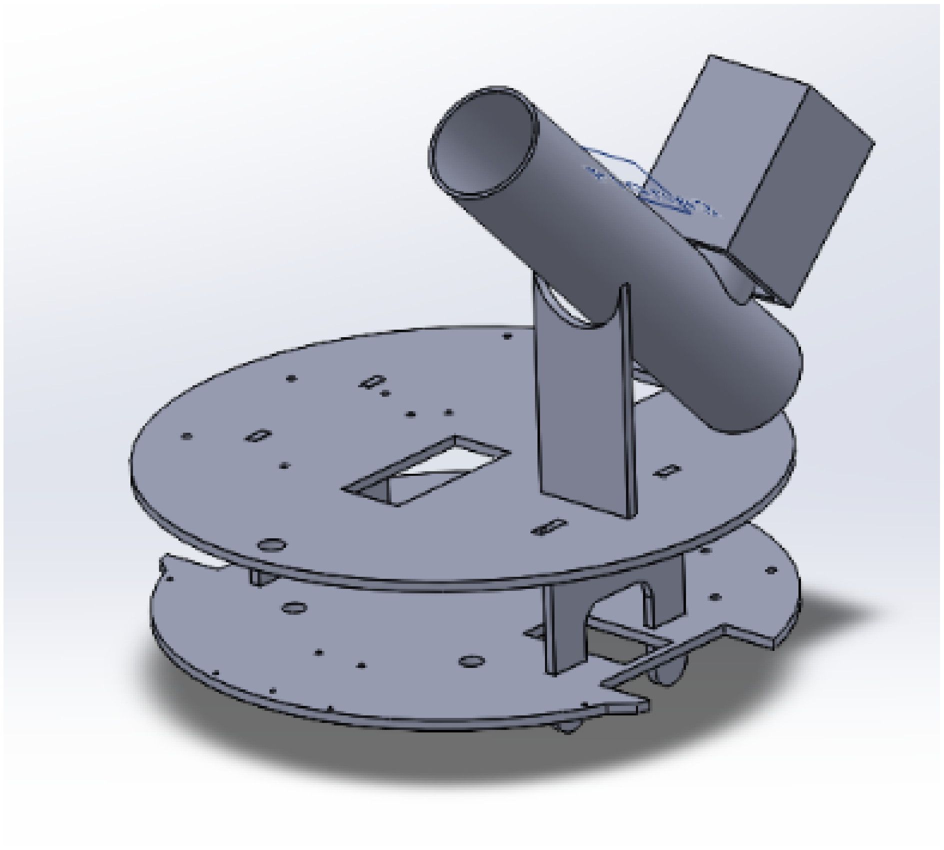

Figure 3: Initial CAD Model of the "MoneyMachine"

The initial design of the robot for this project consists of a dual level circular-based robot that utilizes two DC motors to drive as well as skids at the front and back in order to keep the bot balanced. The robot features a gravity-fed pvc pipe launcher with another DC motor used as the fly-wheel driver that transfers energy to the ping-pong ball. Large empty spaces were left on the top level as well as a few holes in the bottom level, these spaces are used to transfer signal and power lines between sensors on different parts of the robot. The robots composition has been divided into 4 sections, defining the top and bottom side of both layers as the robot is littered with sensors and boards to control the onboard sensors and actuators

2.2 Level 1 Design

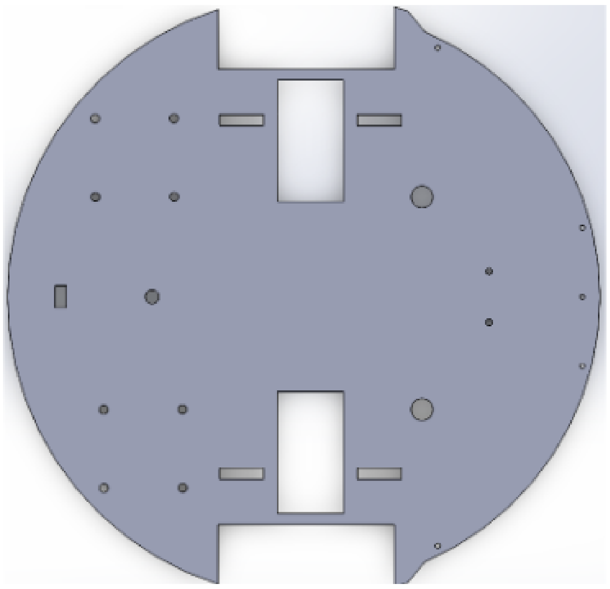

Figure 4: CAD Model of the bottom level of Robot

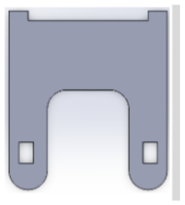

Figure 5: CAD Model of the motor mount design

Included above is a cad model of the base layer of the robot in addition to the motor mounts that connect through a press fit and mechanical stop at the bottom. In Figure 4 the right side of the bot is the front and the left side is the back of the bot. The mounting holes on the base are used for wires to transfer to and from the bumpers and track wire sensors on the bottom of the robot as well as providing power to the LED strips that were on the bottom of the bot. There are also several mounting holes for screws to place in to secure finished perf-boards in addition to the skids. The square holes in the motor mount are used to secure the motor and mount more securely when the components are together.

2.3 level 2 Design

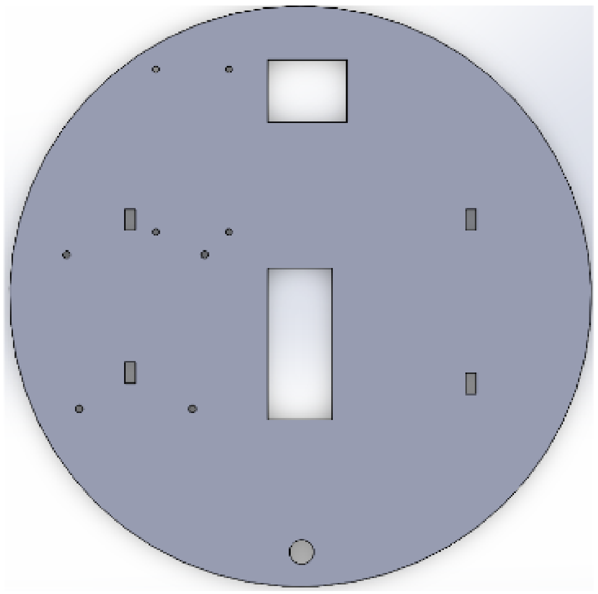

Figure 6: CAD Model of the top level of the robot

This figure of the second layer of the robot is configured so that the bottom is the front of the robot and the top is the back side of the robot. The small circular hole at the front is for the remote power switch, the large gap in the middle is for signal wires and power lines to transfer between levels and the small square gap at the back is used for the rack and pinion. The rack and pinion finds its self mounted on the first layer but protrudes up throughout the second level as seen in the real-life photo. The smaller scattered holes are for mounting the UNO stack as well as the motor mounts that act as a wall that connects layer 1 and 2. The rack and pinion raises by about 5 inches in order to receive a more consistent signal from the beacon as it is mounted significantly higher than the robot is allowed to be.

2.4 Launcher Design

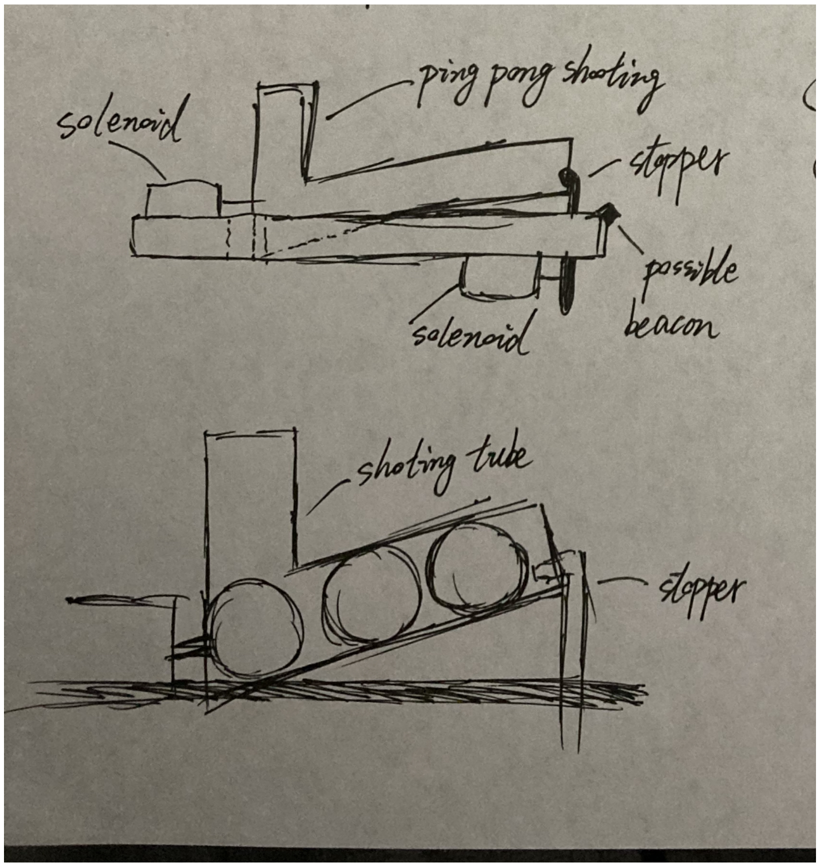

Figure 7: Initial Design Sketches of The Launching Mechanism

The launcher design is simply a PVC pipe that holds three ping-pong balls on top of each other. When given the command, a small solenoid in the back of the launcher pushes the ping pong ball into the flywheel where it gains the energy required to launch into the basket. Above, the robot is depicted and the pvc based flywheel launcher is apparent. There was alot of struggles with constructing the launcher but alot was learned about launcher composition over the course of our experiments. Major take aways for attempting a launcher like this again are

- Flywheel needs to be wider in order to grab ball well
- A more compliant material allows for a greater grabbing force on the ball
- Larger wheel will carry a greater tangential velocity
- launcher barrel needs to be long in order to reduce randomness in accuracy

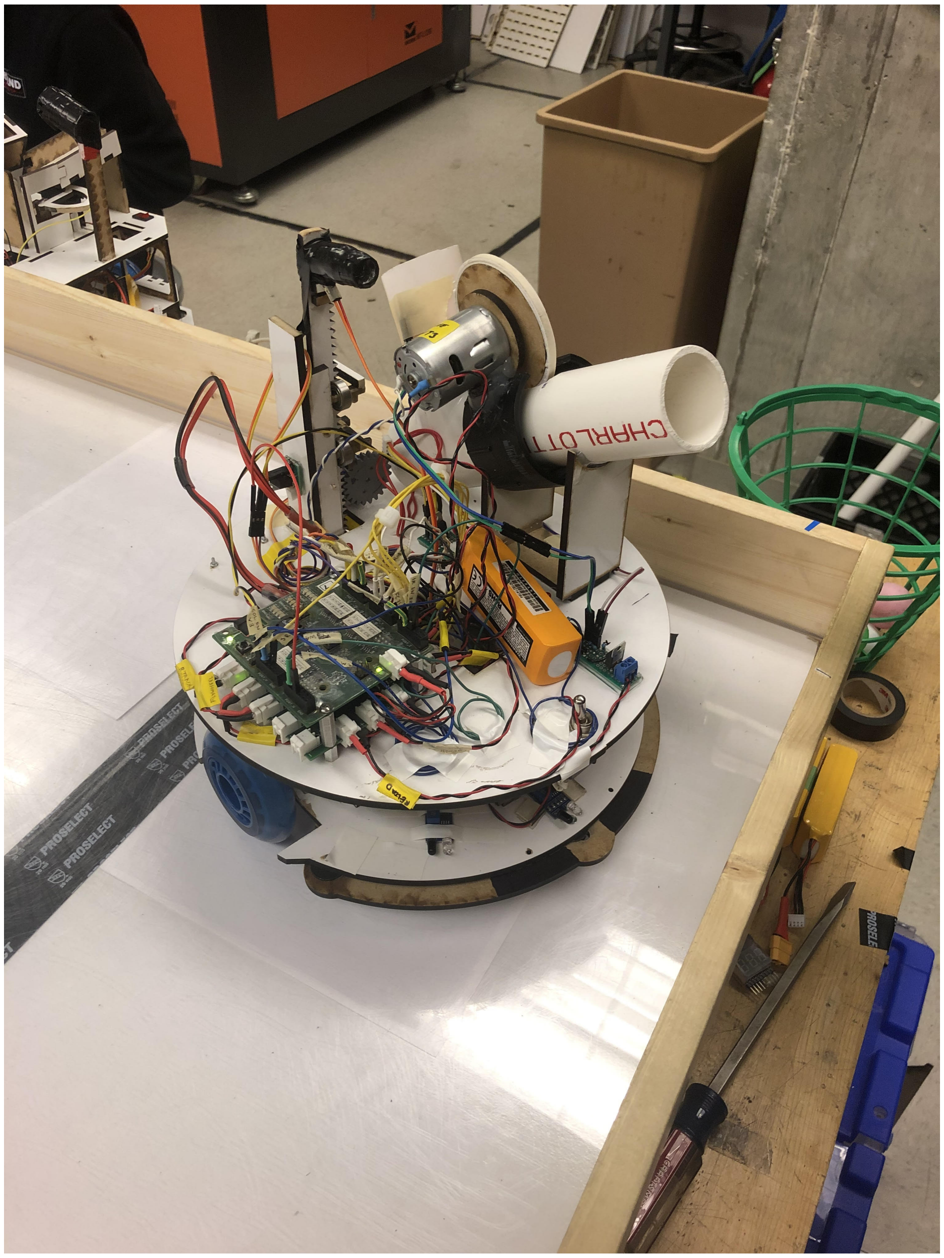

Figure 8: Picture of Robot "MoneyMachine" Two Days before MinSpec Checkoff

3 Electrical Design

3.1 Track Wire Detection

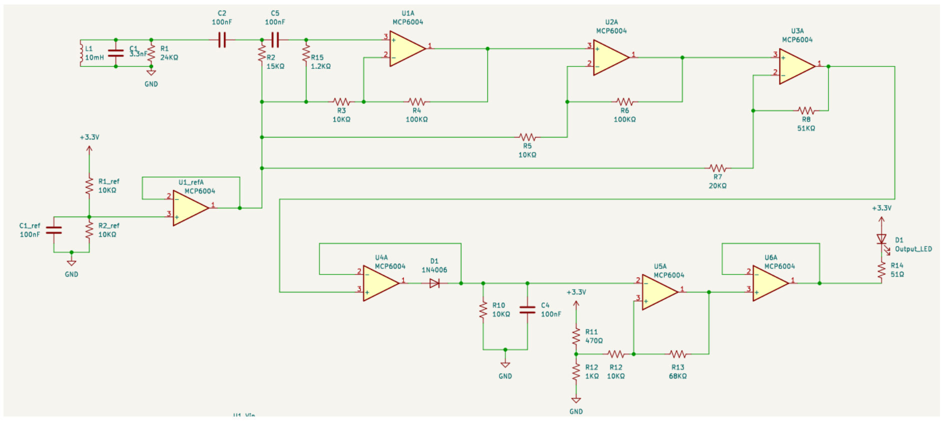

Figure 9: Track Wire Detector Schematic

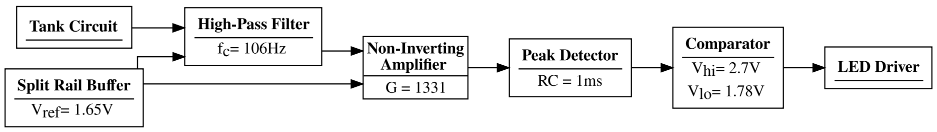

Figure 10: Track Wire Detector Block Diagram

The track wire sensor is one of the most important sensors featured on the robot and is utilized in positions at the front and back of the robot. The Sensor receives the signal from the inductor that is then passed through a passive 2nd order high pass filter with $F_c = 110Hz$. This signal then passes through a gain stage where the signal is gained by a factor of 1,331. Finally the signal runs through a peak detector and comparator constructed from an op-amp before finally being sent out. This detector performed reliably with a range of $2.5 inches$ but could use optimization in the number of op-amps used as the current design requires two MCP6004 chips.

3.2 Infrared Distance Sensor

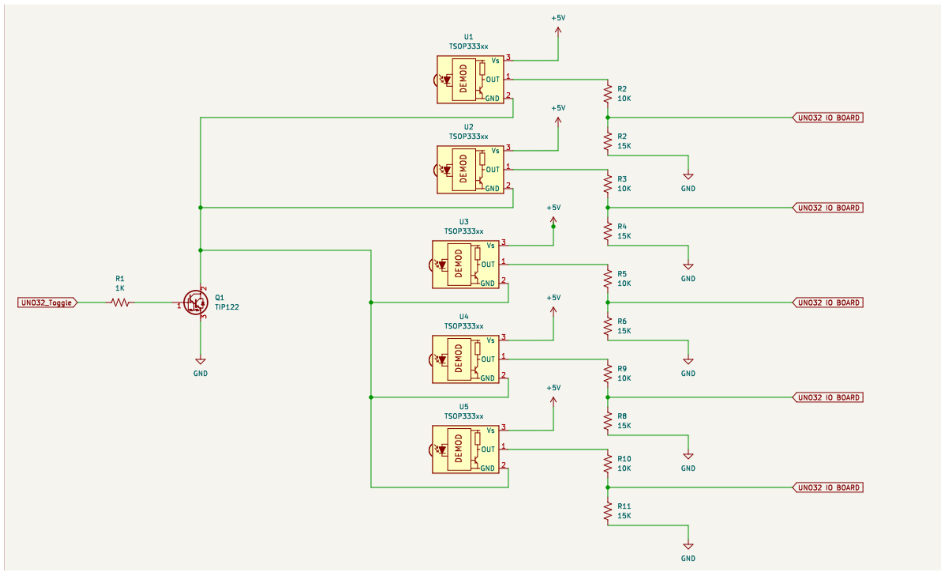

Figure 11: Infrared Light Distance Sensor Schematic

The infrared distance sensor offered a lot of room for learning during its construction. This comes in part from lack of experience designing circuits but also due to the poor quality of the sensors purchased. This circuit utilizes 5 IR sensors that are switched using a TIP122 and the UNO32 as the trigger. This allows the robot to turn off sensors that are not being utilized in order to maintain more efficient power usage. The OUT line of these sensors output at 5v and are stepped down to 3v from a voltage divider circuit in order to safely be used as inputs to the UNO32.

3.3 Reload Detection

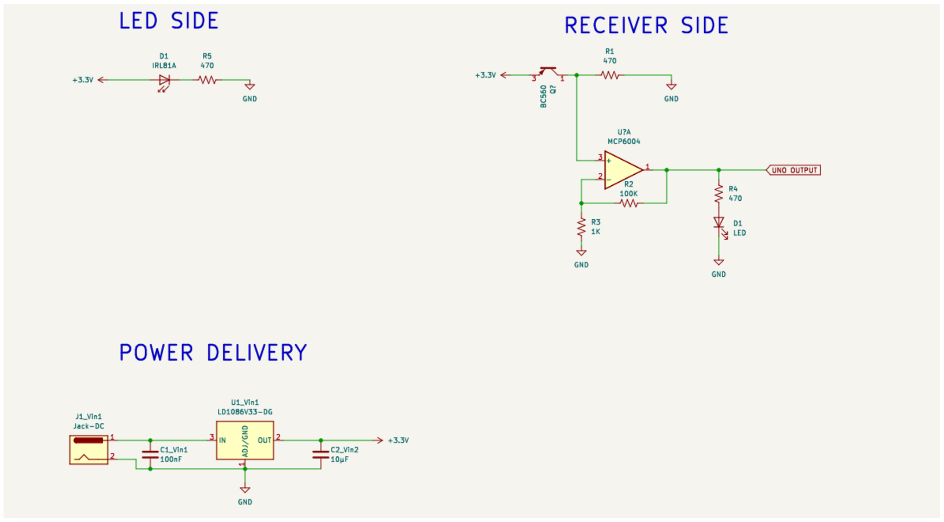

Figure 12: Reload Detector Schematic

The reload sensor utilized another two IR sensors in line on the same plane. One sensor was used as a receiver and one was used as the emitter. The reason that a beam-break sensor was used instead of a single beam sensor is that the single sensor would still detect its reflection off of the ball even when loaded. This slight complication in the electrical and mechanical design of the sensor allowed incredibly easy integration into software. In addition, this sensor allowed robust performance in the reloading area of competition as the bot is not dependent on a timer that starts when it enters the reload zone. One problem that was encountered in the construction of this sensor module is the sensitivity of the sensors to heat when being soldered. Countless sensors were destroyed before the sensors were placed into female jumper wires instead of soldering.

3.4 Beacon Detection

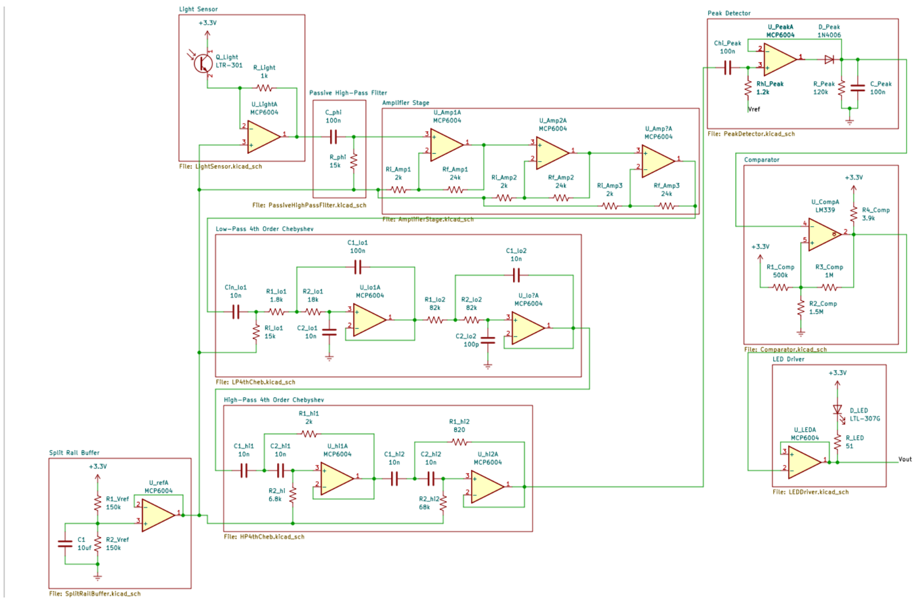

Figure 13: Beacon Detector Schematic

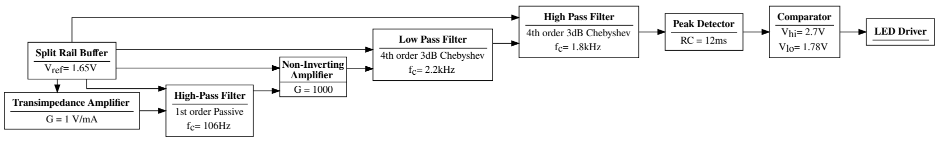

Figure 14: Beacon Detector Block Diagram

The beacon detector is the single most important sensor on the robot as it is integral to our navigation and aiming. The beacon detector was the most work intensive as this beacon was designed with the field in mind, allowing detection of the beacon at 9 feet away as well as having no response to differing frequencies even when the receiver and emitter were physically touching. This sensor was not as reliable as it could have been and this comes in part from poor mechanical shielding in the beginning of construction as well as the fact that the circuits behavior had unforeseen behavior changes in the transition from breadboard to perfboard. The block diagram above details the path of the signal as it travels from the input through filtering and gain stages, eventually reaching a peak-detector and comparator duo in order to provide the output signal.

4 Software Design

4.1 Main HSM

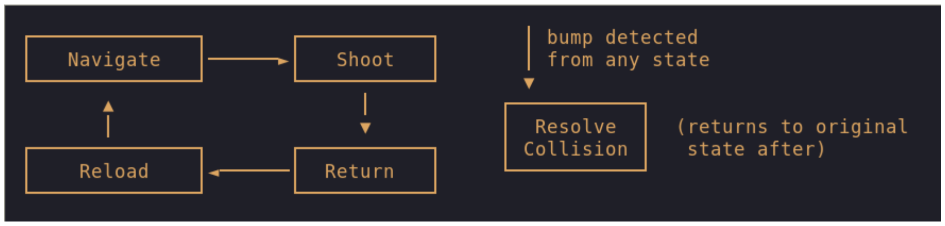
Navigate
Shoot
bump detected from any state
Resolve Collision
(returns to original state after)
Reload
Return

Figure 15: Main HSM diagram

The main hierarchical state machine contains 5 states that encapsulate the core functionality of the robot. ‘Navigate‘ moves the robot from the reload zone to the shooting position within the 5-point zone. ‘Shoot‘ shoots the 3 ping pong balls at the target. ‘Return‘ attempts to move the robot back to the reload zone, and ‘Reload‘ waits for the operator to finish reloading the robot with ping pong balls before transitioning back to the ‘Navigate‘ state. The ‘Resolve Collision‘ state handles responding to bumper events.

Initializing the main HSM. The beacon is 16 inches above the ground, but our robot is height-limited to 11 inches. To ensure that our robot sees the beacon at all times, it is mounted on top of a vertical rack and pinion controlled by an RC servo. To raise it, in the initialization function for the main HSM, the servo is activated and a short ES_TIMER is set. When the timer expires, the servo pulse is stopped.

4.1.1 MainHSM: ResolveCollision

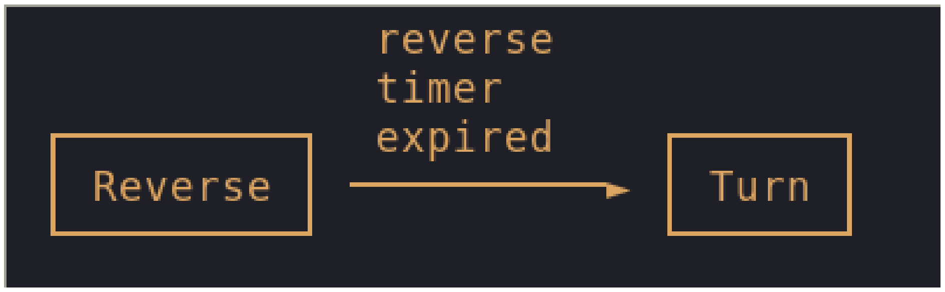

Figure 16: State machine for resolving collisions detected by the bump sensors.

If either of the bumpers are tripped, the state machine moves to the ’Resolve Collision’ state. Here, the robot reverses for a fixed amount of time, then turns away from the direction of the bump (turning 90 degrees to the right if both the front left and front right bumpers are tripped). Afterwards, it returns to the top-level state that it was in before moving to this state.

4.1.2 Main HSM: Navigate

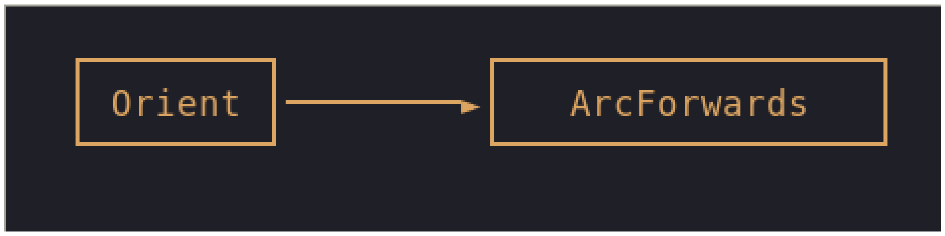

Figure 17: Navigation FSM

This state is responsible for getting the robot to the shooting position. Our robot is only designed to shoot from the 5-point zone, so we use track wire crossing events to know when we've reached our target destination.

The overall state machine shown in Fig. 17 is relatively simple. In the 'Orient' state, the bot turns to find the beacon, then moves to 'ArcForwards'. In the 'ArcForwards' state, the robot moves in an arc to the right - this is to help ensure that it will always get a track wire crossing. If we drove forwards and the robot started in the middle of the field facing the beacon, for example, then the robot would never detect a track crossing until the 1-point zone.

Obstacle avoidance with IR sensors. We use 5 IR sensors to handle avoiding collisions with the wall. When writing the software to navigate using IR sensors, I found that it was more effective to use the state of IR sensors to navigate rather than using IR sensor events (sensor tripped/sensor untripped). To do this, I poll the IR sensors every 50 milliseconds within the 'ArcForwards' machine with an ES_TIMER. Once the timer expires, the IR sensor states are polled and the turn angle necessary to avoid the obstacle is calculated. If no sensors are tripped, the robot continues to arc to the right.

The sensor being tripped	Turn angle adjustment (degrees)
Front left	+45
Front center	+90
Front right	-45
Side left	+15
Side right	-15

Once the turn angle is calculated, the left and right motor speeds are adjusted to turn the robot at the specified angle.

4.1.3 Main HSM: Shoot

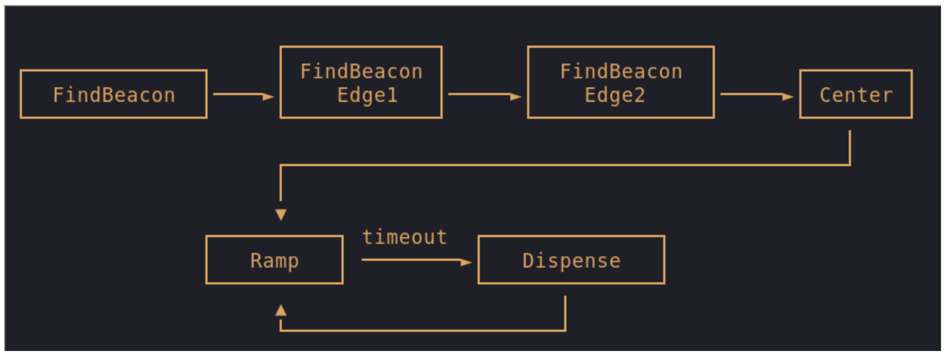

Figure 18: Sub-state machine for shooting.

This state is responsible for shooting each of the balls. To begin, the robot must first locate the target. We could not mechanically get the range of the beacon detector to be narrow enough, so it ended up detecting the beacon while pointed slightly to the left and slightly to the right of it. This caused our robot to often miss the target entirely.

To remedy this, our state machine finds the full beacon detection range, then moves to the center of it. It starts by turning right until it finds the beacon. Once found, it continues moving to the right until the beacon signal is lost. It then records the current time and turns left until the beacon is lost again, at which point it records the time once more. After this point, the robot turns to the right for $elapsedtime/2$ milliseconds, effectively centering the robot on the beacon detector range.

After the robot has been centered on the target, it begins the shooting process. Our shooting mechanism is a single-wheel flywheel. Before making a shot, we must give the motor ample time to ramp up to its actual shooting speed. To accomplish this we set a timer, and on the 'ES_TIMEOUT' event we activate the solenoid to push the ball into the flywheel. This process is repeat 3 times and then the robot transitions to the Return state.

4.1.4 Main HSM: Return

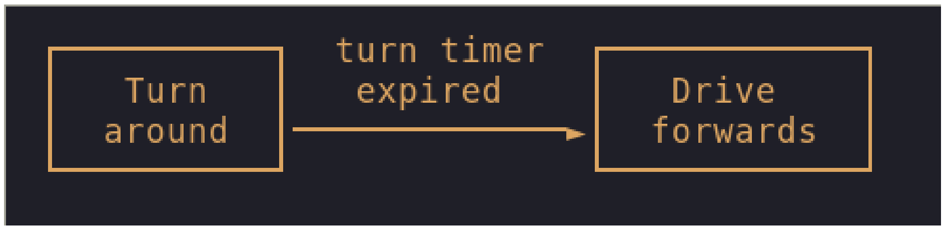

Figure 19: Sub-state machine for returning to the reload zone.

This state is responsible for returning to the reload zone. On entry, the robot turns 180 degrees and then starts freely navigating similar to how it navigated in the 'Navigate' state, except it drives forwards instead of moving in an arc. It only stops moving and enters the reload state when the reload sensor trips, which means it has made its way into the reload zone and we have started reloading the robot.

4.1.5 Main HSM: Reload

This state is responsible for reloading. When the reload sensor (located at the height of the 3rd ping pong ball within the reload tube) has been tripped for more than 1.5 seconds, the bot returns to the 'Navigate' state. We chose to do this rather than have it wait in the reload zone for a fixed amount of time because of the very likely possibility of us clumsily dropping a ping pong ball while reloading or something similar happening that would prevent us from reloading the robot in that fixed timespan.

4.1.6 Bumper FSM

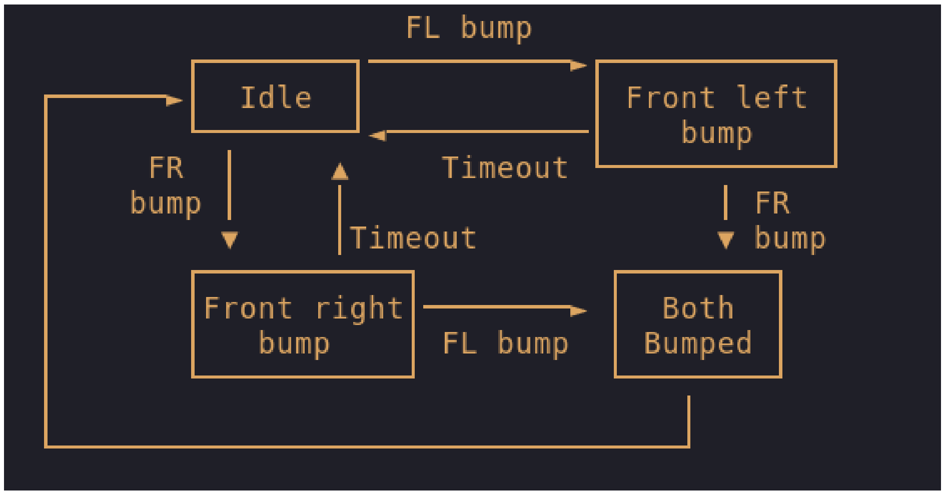

Figure 20: Bumper FSM

Our design has 2 bump sensors (microswitches) on the front right and front left side of the robot. We detect a front center bump if both FL and FR bumps are tripped. However, in that scenario, both FL and FR bumps will very rarely be tripped at exactly the same time; there are a few milliseconds of delay before the other bumper is tripped. This FSM is responsible for waiting those few milliseconds to detect a center bump.

The bumper event checker will detect for changes in bumper states and post those events (‘FL_BUMPER_TRIPPED‘ and ‘FR_BUMPER_TRIPPED‘) to this FSM. The logic in this Bumper FSM will determine if it’s actually a FL/FR trip or a FC trip, and then post the appropriate event (‘FL_BUMPER_TRIPPED‘, ‘FR_BUMPER_TRIPPED‘, or ‘FC_BUMPER_TRIPPED‘) to the main HSM. Note that I don’t really have a use for ‘UNTRIPPED‘ events so I don’t generate them in this FSM.

Design decision notes: in Lab 0, I determined if both bumpers were tripped by putting this logic directly within the main HSM, but really, the main HSM shouldn’t be responsible for that. That logic should be separate. Bumper events should be generated elsewhere and the main HSM should only respond to them.

4.1.7 Beacon detecting FSM

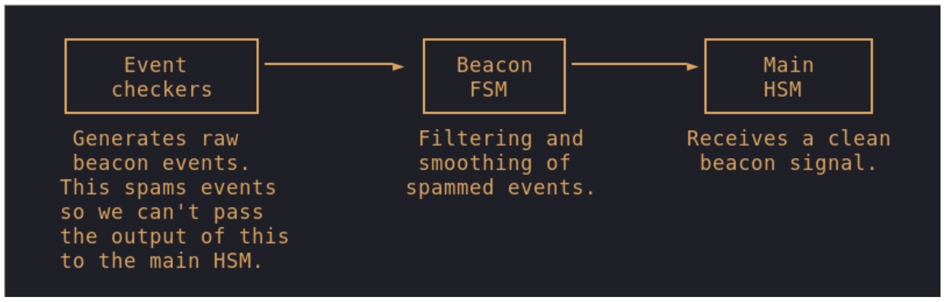
Event checkers
Beacon FSM
Main HSM
Generates raw beacon events. This spams events so we can't pass the output of this to the main HSM.
Filtering and smoothing of spammed events.
Receives a clean beacon signal.

Figure 21: How the beacon detection FSM works with the rest of the software.

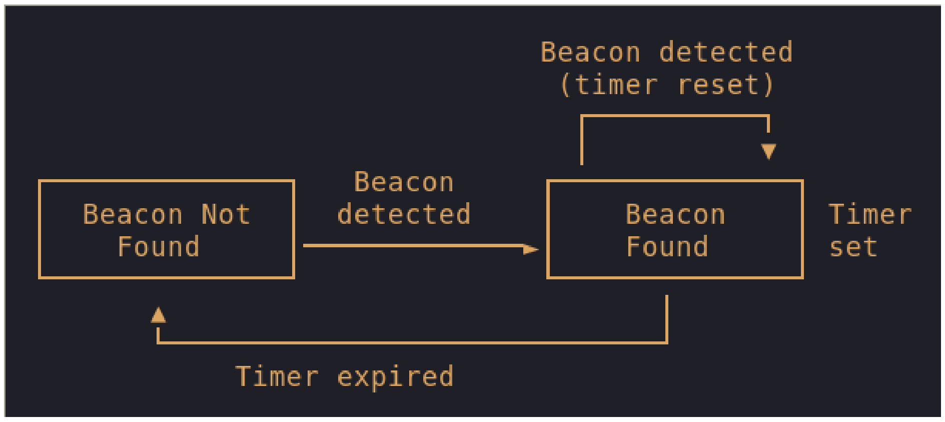

Figure 22: Beacon detection FSM.

The beacon detector had trouble producing a steady output as the transition from breadboard to perfboard slightly changed signal amplitudes and so the high level required by the comparator was slightly higher than it should have been, to counteract this a beacon detecting FSM was created.

Beacon: Event checkers. The beacon detector output is inverted (low == detected, high == not detected). Beacon detector output must be read as an analog signal because the low output 1) isn't stable and 2) doesn't meet the Uno's threshold for a low signal so it never reads low. An ADC value of less than 300 corresponds to an effective low signal. If the current state != the last state, an event is posted (either 'BEACON_DETECTOR_RISING_EDGE' or 'BEACON_DETECTOR_FALLING_EDGE'). The raw beacon detector output is so unstable that this event checker will generate dozens of these events within the span of a few seconds when facing the beacon.

Beacon: FSM. The purpose of this FSM is to turn those dozens of spammed events into a clean 2 events. This has 2 states - 'BeaconFound' and 'BeaconNotFound'. On ES_ENTRY into these 2 states, the FSM posts either a 'BeaconFound' or 'BeaconLost' event, respectively, to the main HSM. The default state is 'BeaconNotFound'. When a 'BEACON_DETECTOR_RISING_EDGE' event is detected, it transitions to 'BeaconFound'. On entry into 'BeaconFound', a 200ms timer is set. Each consecutive 'BEACON_DETECTOR_RISING_EDGE' event resets this timer. On 'ES_TIMEOUT' it transitions back to 'BeaconNotFound'.

5 Project Conclusion

This project was incredibly important to our development as students and engineers as it required problem-solving on a scale that is not typically seen in other classes or personal projects. Throughout the processes of designing,constructing, and integrating electro-mechanical sensors and actuators with software; teams learned how to effectively collaborate in groups as well as how to tackle multi-faceted problems while utilizing fundamental first principles. Looking back on the finished robot, it is very easy to decide what went right and what went wrong or was too over-complicated in implementation for all aspects of the robot whether it be mechanical, electrical, or software. This project was incredibly eye-opening to the fact that one learns much faster by counting their failures than by their successes, as it will always be much easier to see and correct what you are doing wrong as opposed to what you are doing right.