TurboBlaster9000

ECE 118 Final Project Report

David Calman, Haitham Alsaade, Jacob McClellan

University of California, Santa Cruz

Professor Gabriel Elkaim

Fall 2021

Abstract

Over the course of five weeks, our team conceptualized, designed, prototyped, built, programmed and tested a robot that could complete a challenge field. This field included three towers, each with nine holes, of which one is active. The field included boundaries on the edge and in the middle in which our robot could not pass. The task required our robot to autonomously navigate between towers, identify the correct hole, and deposit a ping pong ball into that hole before moving onto the next tower. Our robot accomplished this feat using a combination of sensors, including an IR 25kHz beacon detector, a 2kHz track wire detector, an ultrasonic ping sensor, four bumpers and seven tape sensors. There sensors were designed or chosen by our team, several created on perfboards, and incorporated into our robot design. Our robot deposited the ping pong balls into the towers using a launcher-like mechanism controlled which used a servo to load a tube equipped with a flywheel. In time, our robot was able to successfully navigate the field, depositing ping pong balls in the correct hole of two towers without any out of bounds issues.

Introduction

Class Background

This robot constitutes the final project of ECE 118 - Mechatronics at UCSC. Mechatronics is a class about the development of mechatronic systems. These systems include mechanical structures controlled by output devices - such as motors. These structures and output devices are controlled by software, which in turn is informed by data from sensors. The scope of this class includes circuit design, hierarchical state machine design, soldering, Analog/Digital converters, event-driven programming, sensor design, motor control and more.

The first few weeks of the class included a combination of laboratory assignments, focused on developing the skills necessary to complete the final project, which constitutes the class's lifespan.

Class Technology

The microcontroller utilized throughout the duration of the class is the Uno32. The Uno32 is a development board built around a PIC32MX microcontroller. We are using it with the professor's custom IO shield which exposes five ports.

The software used for programming in the class is based upon The Events and Services Framework (ES Framework), developed by professors at UCSC. ES Framework is a framework for event-driven applications. We do not call functions from its API, instead, main() and interrupt handlers call our service(s) and event checkers. Services should avoid interacting with sensors directly. Instead, event checkers should interact with the sensors and post events to services.

This class utilizes Solidworks, a 3D Solid CAD software. It is used for creating 3D Models which can be created using the Laser Cutter. The Lasercutter is a large machine used for cutting 2D dimensional shapes into materials such as MDF and Foamcore. The 3D models that we create within Solidworks need to be created around this 2D constraint.

Final Project

In the final project, we are tasked with building a robot that can deposit ping pong balls into the correct hole on different towers. It must be able to navigate to the towers (located with IR beacons), find the correct hole (marked by black tape) on the correct side (marked by a track wire), and shoot a ping-pong ball into the hole.

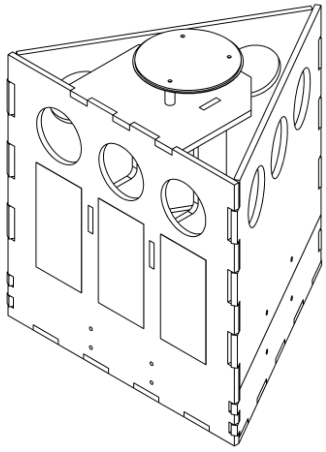


Figure: Illustration of a VAT Tower

The playing field is a large square, whose boundaries are marked with black tape, and also contains a ‘pit’ marked by a black tape diamond in the middle. The robot must not fall off the playing field nor enter more than 50% into the pit.

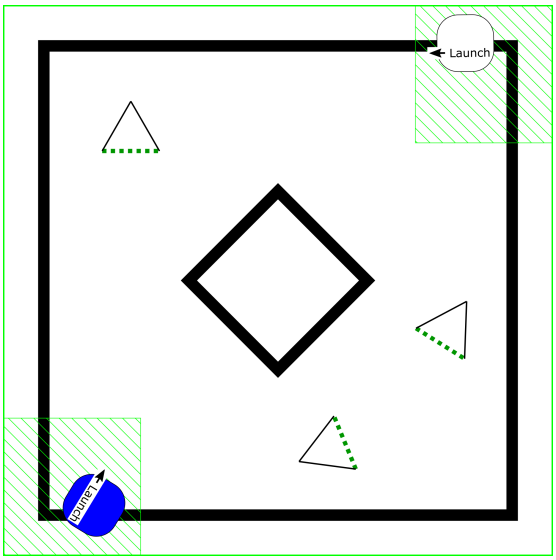


Figure: Illustration of an Example Field

The robot must be able to complete this task within two minutes. Two robots are competing simultaneously on the same field, however, for the final check off the second robot is replaced by a black box to mimic a nonfunctional robot. The robots will begin in one of the four corners of the field in a random orientation. The location and orientation of both the VAT tower and competing robots are set by one of 1000 randomly generated fields.

Methodology

We are using the ES Framework to create an event-driven application for our microcontroller. On the electrical side, the inputs from all the sensors except the bumpers are going to be analog to allow for more sophisticated filtering and hysteresis done by the microcontroller; the amplification is still analog, as are some of the filter stages. The microcontroller’s output will be sent through a driver module to drive the motors of our robot’s wheels and ball launcher, which uses a flywheel. The electrical circuits are built on perfboards that are connected by solder

joints. The chassis of our robot and many of its other parts are made out of laser-cut medium-density fiberboard. The laser cutter operates on DXF files exported from SolidWorks.

Materials

Going into the robot:

MDF

Fasteners

Hot glue

Solder

7 tape sensors and modules

Beacon detector

- Beacon detector shield
- Phototransistor
- Conditioning board
  - MCP6004 op-amp

Track wire detector (integrated with its conditioning board)

- 10mH inductor
- MCP6004 op-amp

PVC pipe

- 90-degree bend

3 x DC motor

- 3 x L296 H-bridge

Prototyping

We began our project by coming up with possible prototypes for our robot. Our main focus revolved around the chassis of the robot. We considered omni wheel, two heel drive, tank drive and mechanical wheels, and a chassis to support the wheel choice. We decided pretty early to use a launcher-like mechanism for depositing the ping pong balls. We discussed the choice of

either utilizing a strong solenoid or a fly wheel within the design. Below are some drawings created during this planning phase.

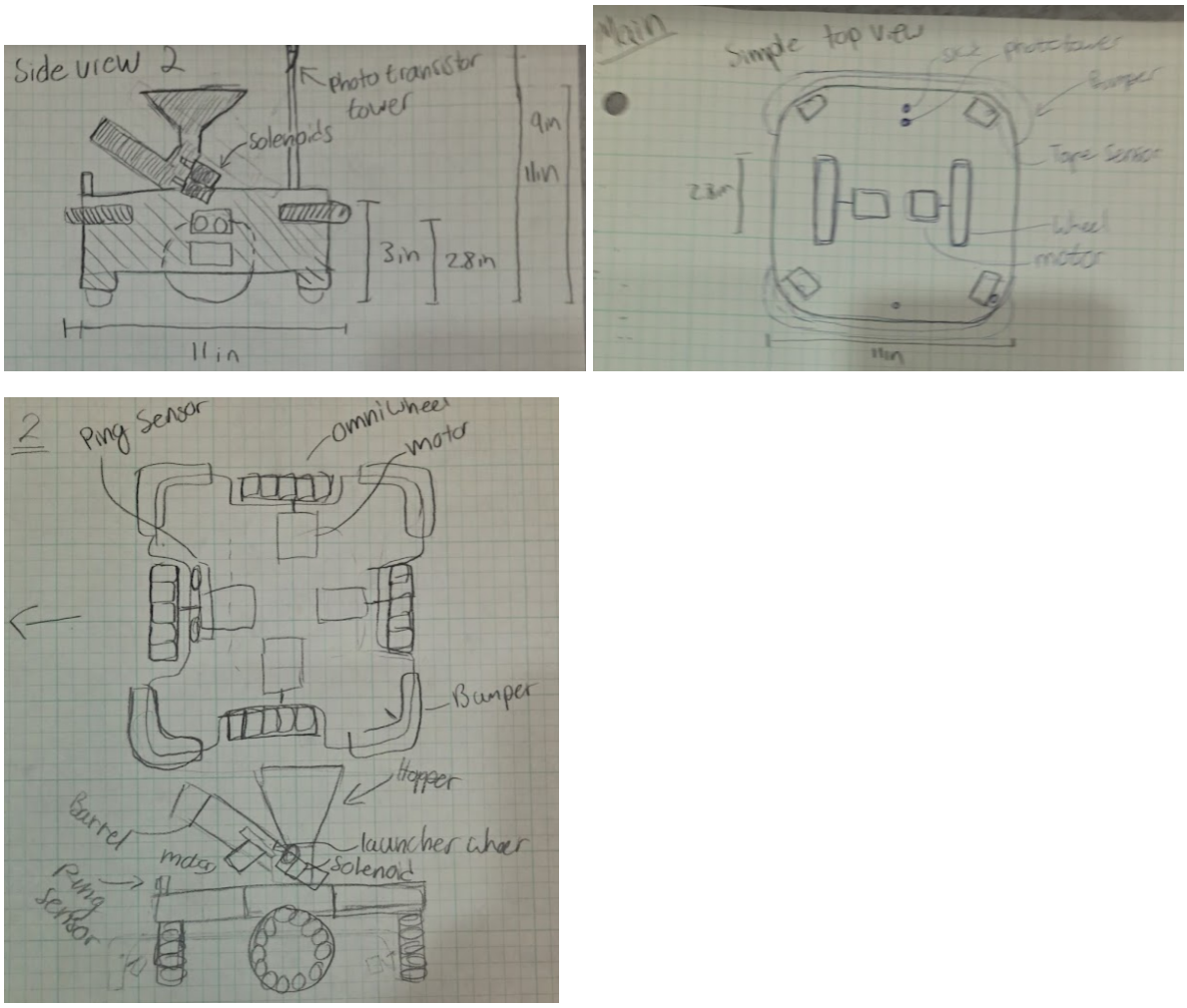


Figure: Different Types of Chassis Drawn

For our final design, we decided to go with a simple two wheel design with skids. The robot would utilize a launcher with an attached fly-wheel. We decided on the two wheel design as it was the simplest and easiest to code for, and also decided to use the fly-wheel design because we were not sure the solenoid would be strong enough to push the balls up the tube and into the hole given our angled design. In the beginning we did not consider other designs aside from the angled launcher, as we were concerned with having room for a hopper mechanism to store excess balls.

Solidworks and Physical Prototype

Solidworks

One of the initial tasks of our project, once we decided on a basic design, was to create it within Solidworks. This would allow us to fully flesh out our vision for the robot. The process also highlighted potential issues with our original design, and allowed us to do a rudimentary prototyping of our mechanical design.

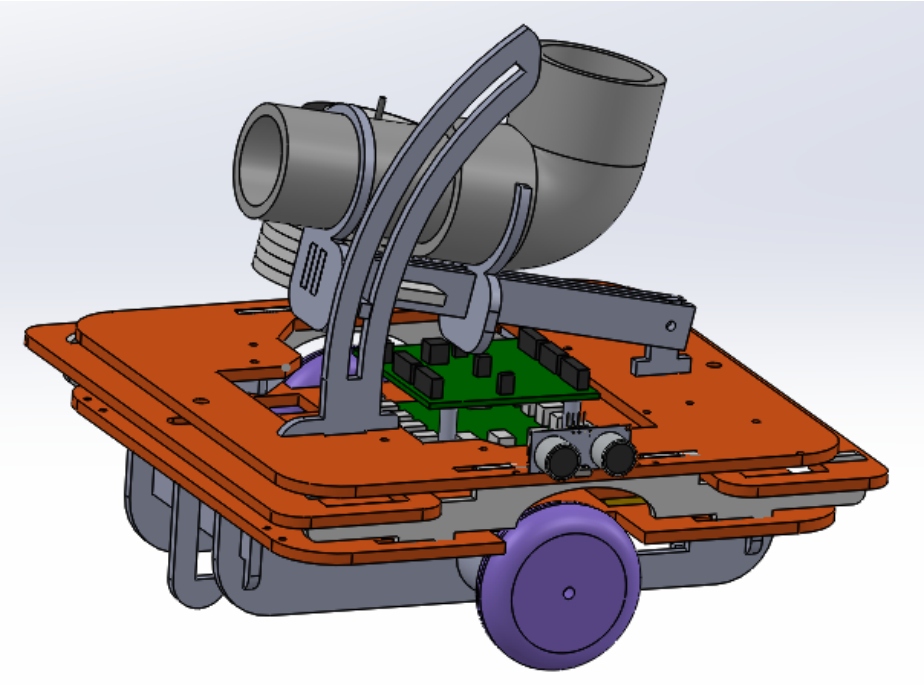


Figure: Full Solidworks Design

The Solidworks modeling also made us consider where our perfboards, sensors and other components need to go, how they would be attached, and how we could make room for all of them. Any further adjustments to perfboard placement could be made by drilling additional holes.

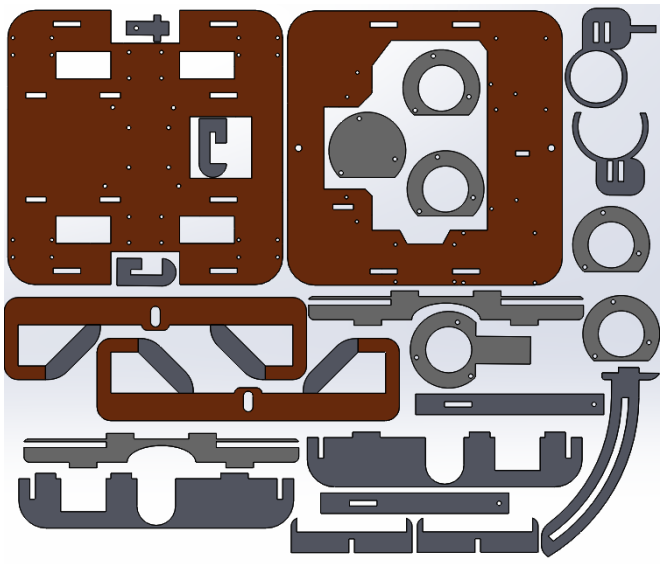


Figure: Flat Layout of Cuttable Components

Above is a display of the cuttable components laid out flat. This assembly was converted to a DXF file and cut using the lasercutter on .2in thick MDF.

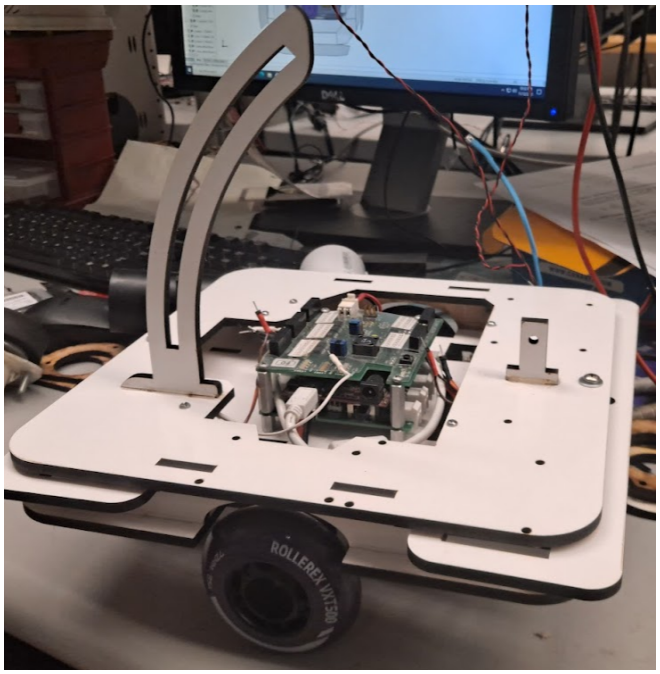


Figure: Cut and Assembled Robot with Additional Components

After cutting, the robot was assembled using the tab and slot construction designed in solidworks, and a combination of screws and hot glue. Screws were used where we would like the robot to be detachable or moveable (such as the top layer and gun hinge) and hot glue where construction was permanent (most of the chassis).

Physical Design

Motorized Platform - Top Layer

Instead of being mounted on panels, the bumpers and tower tape sensors are mounted on an upper deck of the platform, which is spaced from the lower deck so as to be 3” off the ground. The track wire sensor, integrated with its conditioning board, and the beacon detector conditioning board are mounted on the sides of the top layer. They consist of stacks of boards.

Motorized Platform - Bottom Layer

Here you find the wheels in the middle of the sides and the tape sensors on the corners of the underside. In the center of the underside you will find the battery cavity and, under the centers of the front and back edges, a skid each.

Ball Launcher

We have this mounted on a vertical swivel, even though every hole is at the same height, but it is held in its position on the swivel with hot glue. The body of the launcher is made of polyvinyl chloride (PVC) tubing, with two bends. It uses a rubber wheel to launch the ball, and a stepper motor to get the ball to the rubber wheel.

Electronics

Electronics Overview

Our robot consists of two wheels on either side, a platform to which the motors that spin the wheels are mounted, and two panels on the front and left that extend upward.

There are seven tape sensors: two on the front panel, spaced 1.5” apart, one on the left panel at the same height as the ones on the front panel, and four on the underside of the platform, at the corners. The tape sensors on the front and left panels are used to detect the tape that marks the correct hole into which to launch a ping-pong ball in conjunction with a track wire. The tape sensors under the motorized platform are used to detect the tape that marks the edges and the central keep-out zone of the playfield.

We also have a track wire detector mounted on the left panel, an ultrasonic ping sensor mounted on the left panel, a beacon detector mounted in the center of the front panel, and bumpers mounted somewhat low (3” from the ground) on the sides of the front panel.

The track wire detector uses an inductor, and the beacon detector uses a phototransistor. They are both noisy and have a small signal amplitude, so they need to be conditioned. First we amplify the signal, then we filter it.

Beacon Detector

Each VAT Tower was equipped with a 25kHz IR beacon that the robots could use to identify the direction of that tower. A phototransistor can be used to detect that signal. The output of the phototransistor needs to be linear, filtered and amplified before it can be used by the robot’s microcontroller.

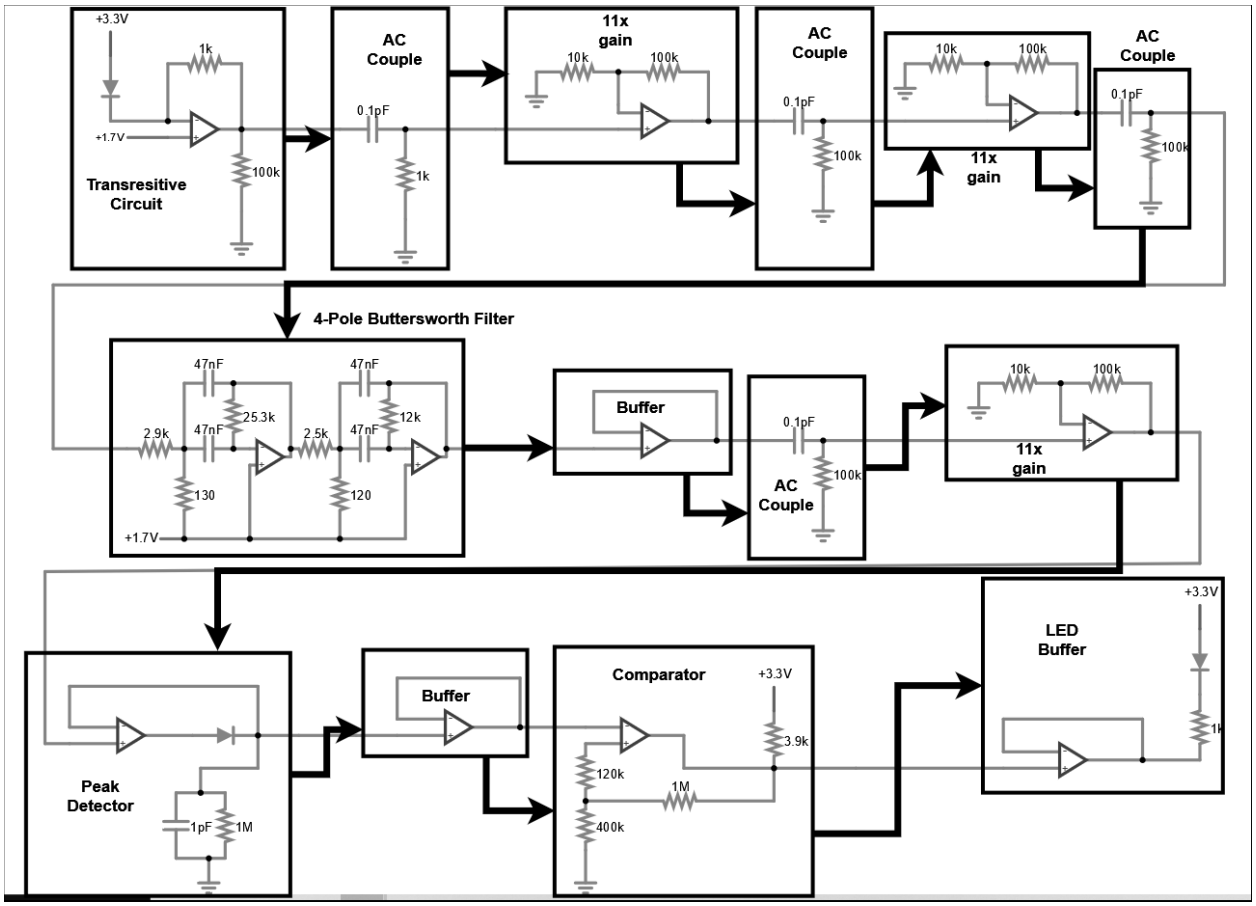


Figure: Beacon Detector, Amplifier, and Filter Circuit Schematic

Our final circuit also included a peak detector and comparator. The peak detector was used to create a nice analog signal proportional to the intensity of the beacon signal. The comparator was used to create a digital signal with hysteresis, which also controlled an LED. In our final implementation of the robot, only the analog signal was used, as digital hysteresis is far easier to affect.

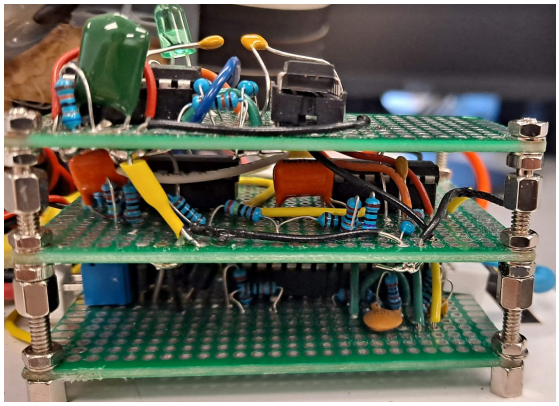


Figure: Triple Decker Beacon Detector

Track Wire Sensor

The track wire is a long wired run across the inside of the active VAT Tower face, which carries an AC 2kHz signal. The alternating current creates a magnetic field which can be detector by a sensor. Our track wire sensor utilizes an LC Tank Circuit, whose resonant frequency is equal to that of the AC current. The magnetic field will induce current in the tank circuit, which can be amplified, filtered and turned into an analog signal.

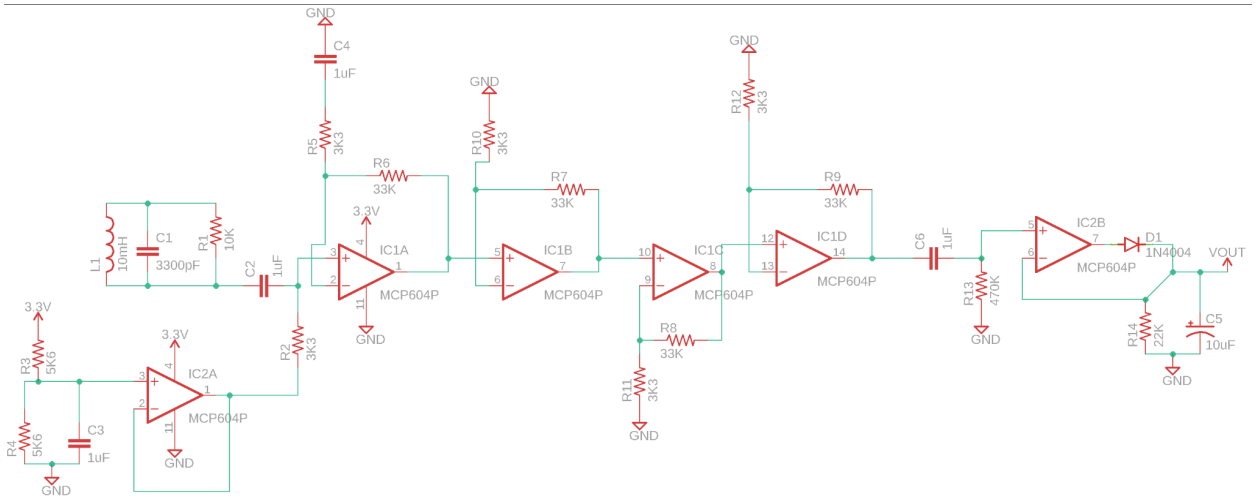


Figure: Track wire sensor schematic

The sensor again utilizes a peak detector to create the desired analog output signal, which is passed into the microcontroller.

Bumper

Our robot utilized four limit switches for its bumpers. The limit switches could create a closed or open circuit, creating a high or low signal on its output pin, which was measured by a digital pin on the microcontroller.

Ping Sensor

In order to gauge distance from the VAT Tower, we used the HC-SR04 Ping Sensor. The ping sensor could be activated by sending a pulse on its trigger pin. Once it receives a pulse, it omits an ultrasonic signal and raises its echo pin high. That pin remains high until the sensor receives the signal back. The high time of the echo pin can be used to infer a distance to an object. The sensor uses 5V logic. Since our microcontroller works on 3.3V, the echo signal needed to be run through a voltage divider before being passed onto the microcontroller.

Figure: Ultrasonic Ping Sensor

Tape Sensor

Due to the need to detect black tape on white backgrounds, a type of photoreflective sensor is needed. We used the TCRT500. The tape sensors we used came on commercially available modules. These Modules require 3.3V power and output both a digital and analog signal. For our project, only the analog was used.

Figure: Commercial Model of the TCRT5000

Software and State Machine

Behavior Overview

Our robot begins in a corner facing a random direction. After it initializes, it will spin clockwise in place, until it gets a beacon found event. This event occurs when the analog signal from the beacon detector goes above a software hysteresis threshold. At that point, the robot will begin approaching the tower. Once it hits the tower, indicated by a bumper press, it will begin to search for the hole. It does this by traversing the tower, maintaining a consistent distance using the ping sensor mounted on its left side, and stopping when both the side tape sensor and track wire sensors are active. It will then pivot in place, and approach the tower head on. Using the front tape sensors, it will repeatedly back up and adjust its position until the launcher tube fits snugly in the hole, allowing both front bumpers to be pressed. The robot will then activate its flywheel, let it spin for a time, and deposit a ball into the tube using the servo motor. Once the

launch is complete, the servo and flywheel are reset, and the robot will back out of the hole. It then rotates and uses timers to drive around the tower, maintaining a roughly parallel position, until the beacon detector finds a new tower and the process begins over again. If in the process of finding a tower the robot collides with pit tape, it will attempt to back up and go around the tape. If the robot mistakes the other field bot for a tower, it will timeout after some time looking for the hole, allowing it to search for a new tower.

Top Level State Machine

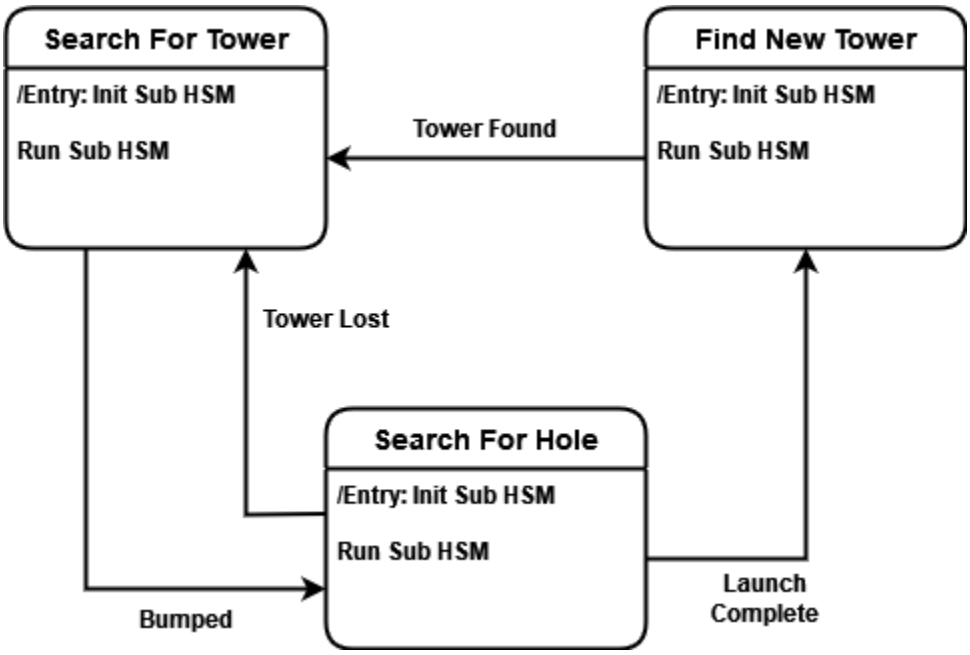
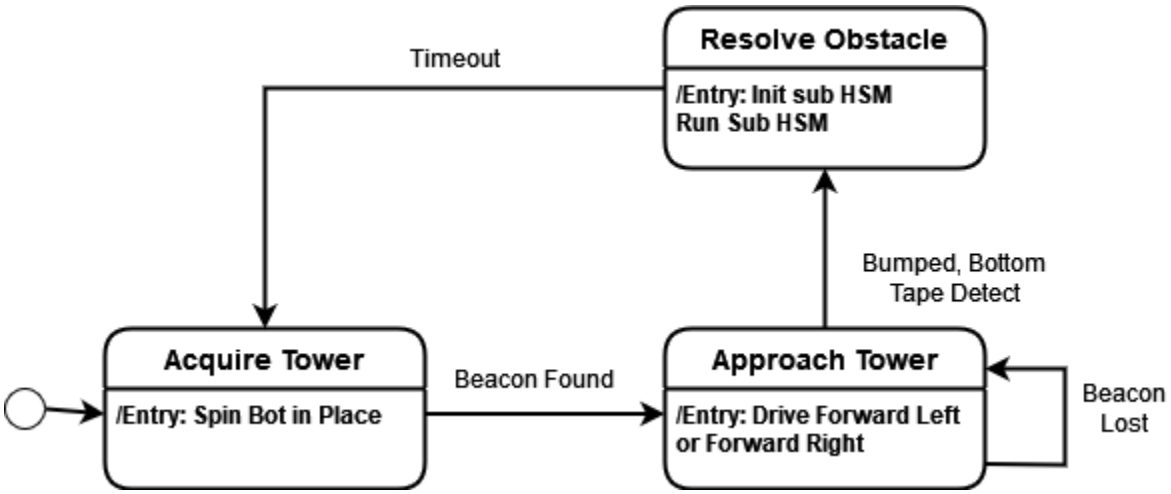


Figure 1: The top level of our final state machine

The top level of our state machine is also the most basic. It consists of three states: Search For Tower (Which could be more appropriately named Approach Tower), Search For Hole and Find New Tower. Search For Tower is the initial state. From here, the robot will lock onto a tower and approach it. If it bumps an object in this state, the state machine will transition to Search For Hole. This sub state machine deals with navigating the tower, aligning with the hole and depositing a ball. If at anypoint the tower is lost while within this state, it will transition back to Search For Tower. However, if a ball is successfully deposited, it will transition instead to Find

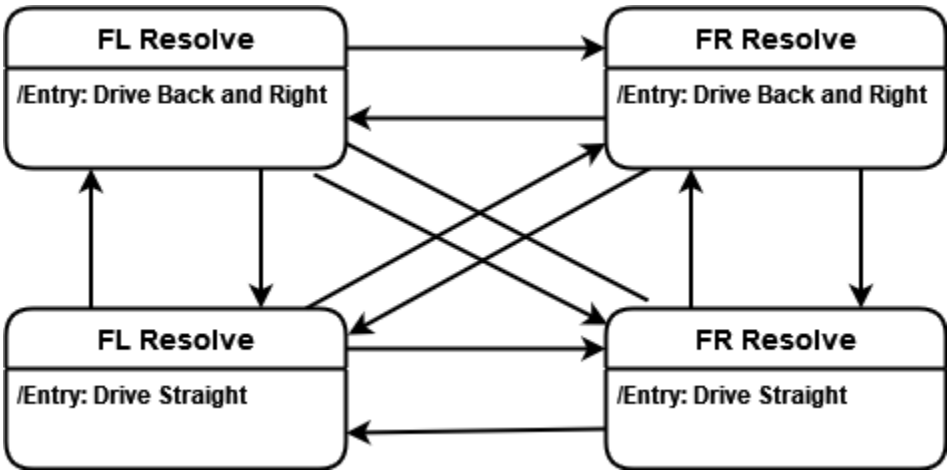
New Tower, whose purpose is to allow the robot to find a new state machine without using the rotate and avoiding reacquiring the same tower again.

Search For Tower Sub State Machine



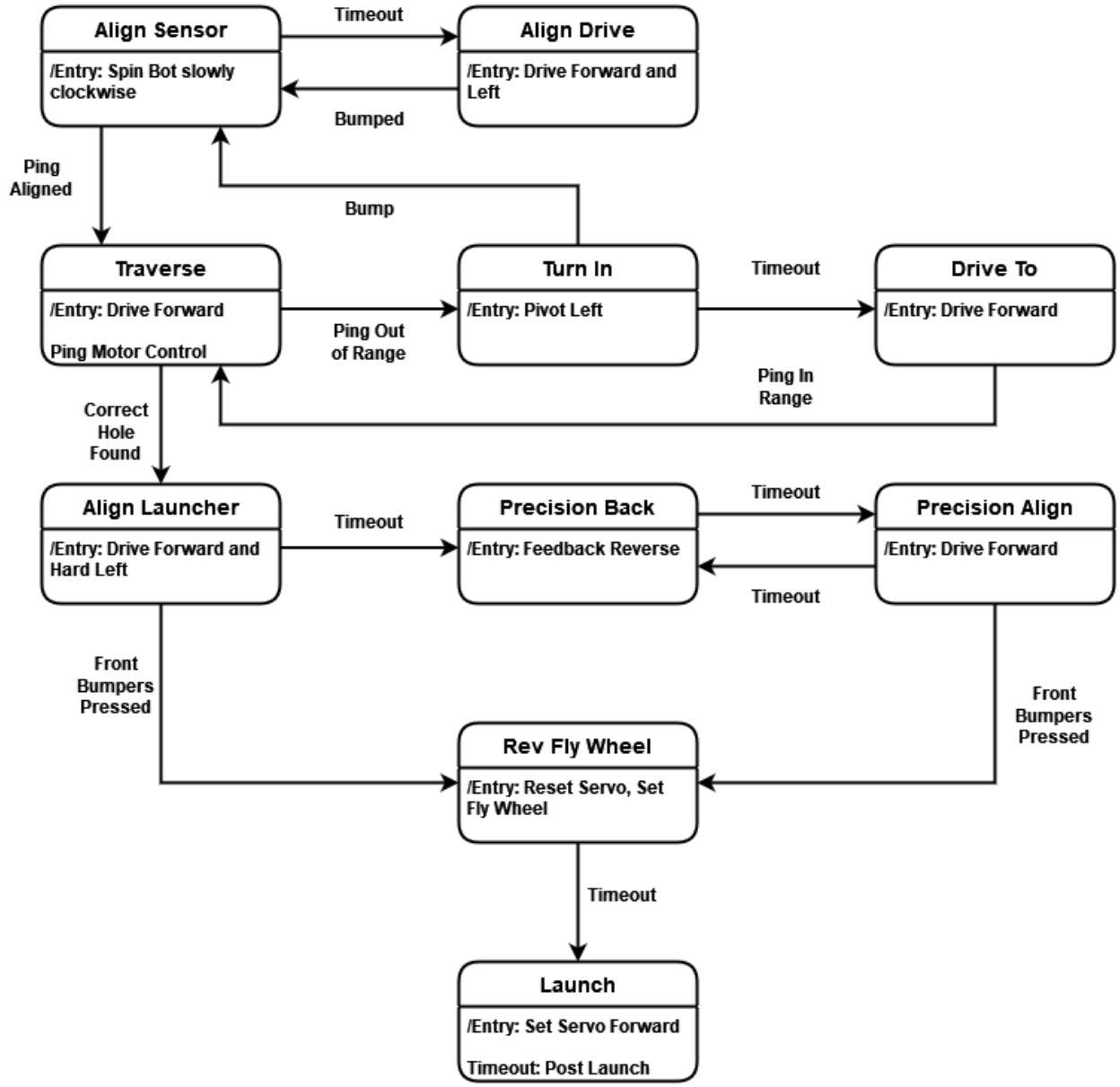
This is the state machine for finding the first tower. The robot spins around until the beacon detector is high enough, then drives directly to the tower, until it finds an obstacle. If it's a tape, or it's a bump far away from a tower, it goes to the collision resolution state machine. It relies on the fact that, if the beacon signal is not strong enough and the robot bumps into something, what it bumped into is not a vat tower. While the current state is Approach Tower, the robot will have a slight left or right bias. The robot will eventually have a Beacon Lost event, in which it will begin to bias in the other direction. There is a timer set up, which is reset each time this occurs. This behavior allows the robot to accurately snake towards the tower. If the timer ever expires, it means the robot was never able to reacquire the tower while approaching, and causes the robot to go back to the acquire tower state.

Resolve Obstacle Sub State Machine



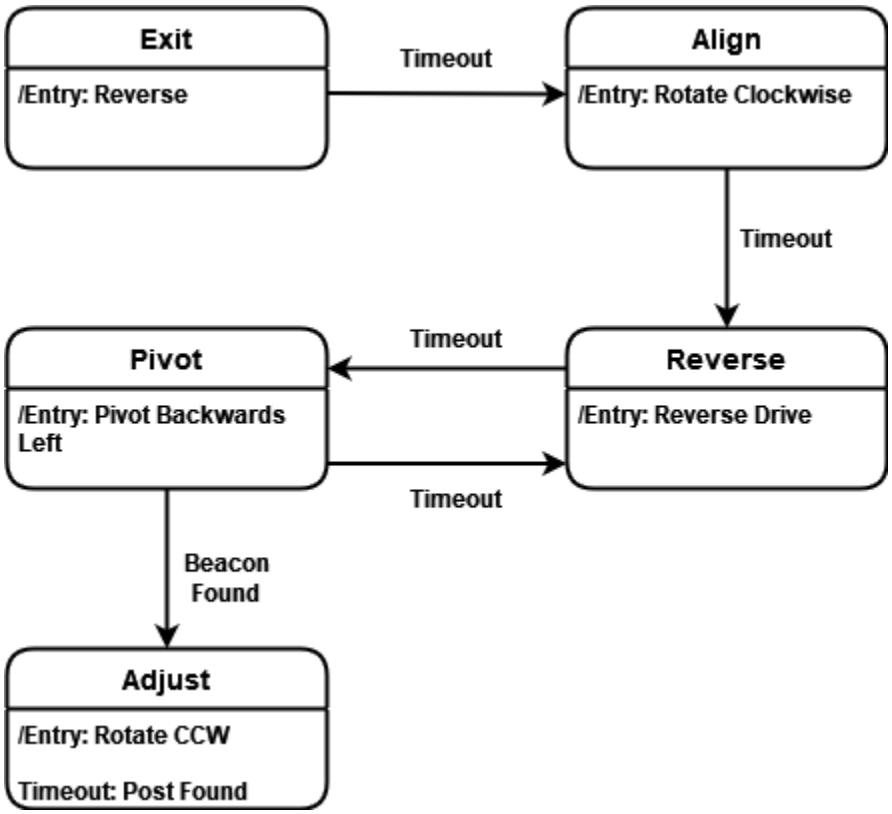
The resolve obstacle state machine handles how the robot behaves when it believes it has acquired an obstacle. The robot will try to go around all objects by going around it on the right. It does this by resolving front collisions back backing up backwards and to the right, and back collisions by driving forward. For example, when the robot encounters the pit,it will back up and to the right, and after some time, go back to the acquired tower state, rotate clockwise and drive forward. This creates a loop where the robot will do driving menuevere when it will go around the pit while maintaining vision with the desired tower.

Search For Hole Sub State Machine



The state machine for going around the tower, finding the correct side and hole, and then aligning the launch mechanism with the hole uses the fact that, since the final PVC pipe section is horizontal and protrudes past the bumpers, both bumpers can only be pressed when the pipe is inserted into the hole.

Find New Tower Sub State Machine



Testing and Adjustments

Throughout the course of our project we ran into many obstacles and hurdles that required us to change up our approach in one manner or another. The first of these obstacles came with our state machine logic's ability to handle obstacle resolution. It took us a while to develop a method for going around an obstacle and not losing which tower the robot was locked onto. Several methods were tried until we settled on the incremental backup to the left and reacquire method.

One of the major issues needing readjustment was our launcher design. Originally, the design required the ball to launch about an inch from the barrel to get into the hole. While this was certainly possible, the precision required was hard to come by using the tape beneath the hole alone. The tape was not consistent between all three towers, and after many hours of adjusting, the best shooting percentage we could get was around 50%. To alleviate the precision alignment problem, we added a barrel extension to the gun whose purpose was to be inserted directly into the hole. It was designed with a length so that if it was not in the hole, both of the front bumpers could not be pressed simultaneously. This allowed for additional logic so that the robot could continuously readjust its position until the barrel was perfectly into the hole. This meant our shooting accuracy went up to 100%.

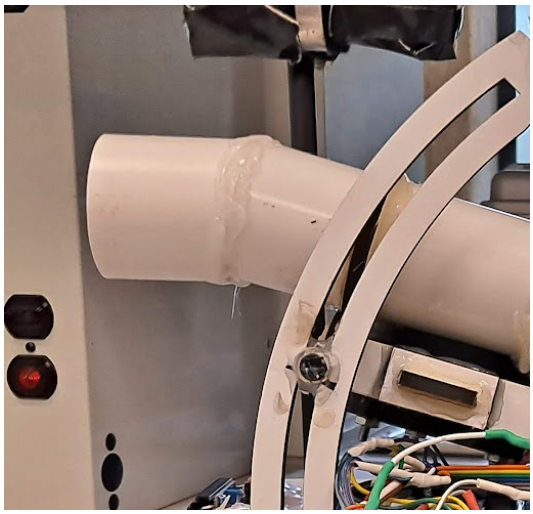


Figure: Barrel Extension

There were many other minor fixes that deviated from our initial design plans, especially within software. One of the primary themes of our software development was incremental development. We Implemented our design step by step, and fixed behavior as we went along. This came mostly in the form of changing macros, adding additional states and covering edge cases. It also had its drawbacks: often our progress would stagnate and we could not deliver our deliverables *quite* on time.

Outcomes

Our robot was able to pass the minimum specifications within time. Its behavior was never perfect, and many issues are still withstanding. However, it was able to complete the course with a dead bot, under favorable circumstances. What the robot was able to do extremely well, was the entire Search For Hole sub state machine. Once the robot correctly aligned with a tower, it could very reliably navigate the tower, find the correct hole and deposit a ball (sometimes 2!).

Another thing our robot was able to do reliably well was once it acquired a tower, the snaking behavior worked very well.

Our robot did have its shortcomings. Its ability to navigate around the pit was and sometimes unreliable. The robot always performed best when it did not have to deal with the pit. It sometimes accidentally lost the tower and acquired a different tower unintentionally when doing this.

If the VAT Tower was too close to the edge of the playing field, our robot had an unfortunate habit of driving off the edge. This is because we never covered that case within software. While not impossible to account for, we decided it was not a good use of our time as we could just give preference to fields where that wasn't the case.

One final and substantial issue was the occasional unresolved software bugs. There were a couple states which could have the robot bug-out and stop responding entirely and be stuck in a specific drive state. There was also the unresolved issue where the robot;s resolve obstacle state wouldnt properly transition and the robot would just drive straight into the pit for no apparent reason.

Discussion

So part of what went wrong is that there was this one part of the state machine that we originally envisioned where it would find the second-highest peak by first using the Extreme Value Theorem to find peaks and then sorting them to find the second-highest.

Another thing was that, since we were all split up and doing what we were good at, nothing went very well. Of course, this went against the advice of the professor, who suggested we all do what we are bad at, with supervision of our teammates.

In hindsight, there were also many major design changes that would be made to the robot if we were to do it all over again. One such design change would be to use a type of omni wheel to allow for two axis movement of the robot. Another design change would be to use a ramping design rather than a launcher to allow for less headache when depositing the ball, also saving time lost on activating the flywheel. Perhaps then we'd get by with just a servo motor or solenoid.

Conclusion

This project was one of the largest of our academic career. It required many hours of designing, prototyping, constructing coding, tweaking, soldering, testing and adjusting to complete. One of the constant themes of our project was that the robot and its components never behaved quite how they were supposed to. Sensor circuits needed to be doubled or tripled checked before possibly needing a do-over. CAD designs highlighted impracticalities of our design intentions. Once put together, the robot's on-field behavior displays the many edge cases that were overlooked, if the robot was to even behave how it was intended under ideal circumstances. The only easy way to navigate through all of this was to identify an issue one at a time and fix them one at a time. What might seem like a mountain of bugs can we whittled away slowly, clearing a path for the finish. Sometimes one bug fix will reveal, or worse, create more in its place, but being able to persist though will allow the robot to slowly and incrementally improve. Eventually the robot will get to a place where it can deposit a ball during the tournament and pass min-spec, but it takes time. Working as a team is the only way to make such a great task manageable.

Our robot had its fair share of issues. In software, on the perfboards, in its mechanical design and more. If we could start over again, there are certainly things we would do differently, and maybe even approach the project from a completely different angle. But with enough work, we were able to get our robot to be successful.

References

Carryer, J. Edward; Ohline, R. Matthew; and Kenny, Thomas W.. *Introduction to Mechatronic Design*.

Uno32 datasheet from Microchip

I/O Board datasheet for ECE118 by either Maxwell Dunne or Gabriel Elkaim

The ECE118 library by Maxwell Dunne

ES_Framework by J. Edward Carrier

https://www.openimpulse.com/blog/wp-content/uploads/wpsc/downloadables/TCRT5000-Sensor-Module-Circuit-Diagram.pdf. Document should be rotated counterclockwise for best viewing, and is in Chinese.