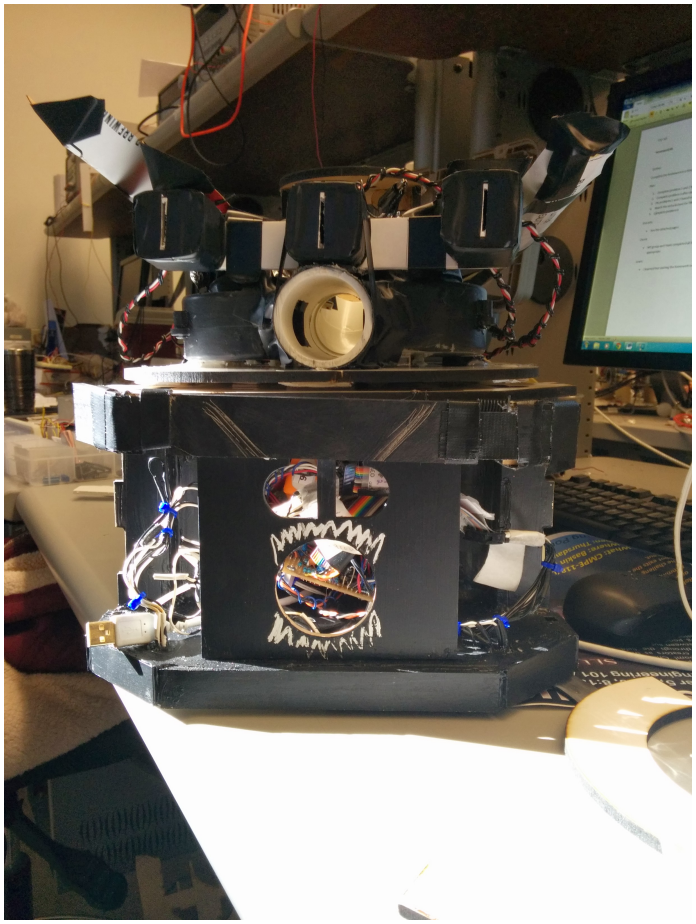
Final Project

The Debuggers:

Angel Cazarez (avcazare@ucsc.edu)
Julie T. Do (jutdo@ucsc.edu)
Russell Kirmayer (rkirmaye@ucsc.edu)
Trenton Louie (tklouie@ucsc.edu)

CMPE 118/218: Introduction to Mechatronics
Gabriel Elkaim

09 December 2013



Contents

1 Introduction

The purpose of this project is to build and implement an autonomous robot that will be capable of navigating across a field marked with black tape while avoiding obstacles. The robot should also be able to shoot the enemy robot, with the end goal of reaching the enemy gate before the opposing robot does.

This project combines technical skills we have learned in the previous labs, and challenges us to integrate it all into one cohesive project.

2 System Level Design

The general strategy for this project was to keep the robot as simple as possible. The minimum specifications were used as guidelines and any difficult-to-implement or excessive sensors were avoided. The project can be roughly divided into four parts: mechanical, electrical, software, and integration. During the first two weeks, the separate mechanical, electrical, and software parts of the project were all done in parallel, while the last week was devoted to integrating the separate parts together.

3 Mechanical Design

3.1 Overview

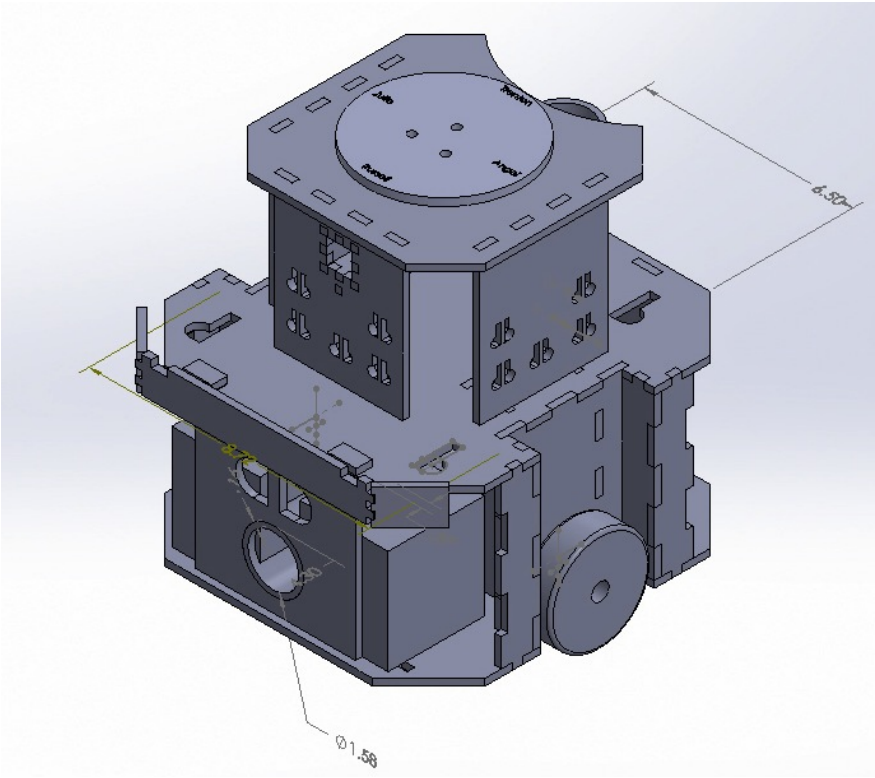


Figure 1: An isometric view of the SolidWorks model for the robot.

The mechanical design is based off of a simple, two-tiered box that would both be robust and maximize space for wiring and all other eletcrical and mechanical components. Most of the members in the team did not have much experience in mechanical design, so making our design simple helped us move quickly. The top tier is designed to hold the beacon detector and most of the wiring and circuitry, while the bottom tier houses the driving and pitching motors, H-bridge, and the tape detectors. The bump sensors consists of two simple switches and a spring attached to a piece of MDF, which is then attached to the front of the robot.

More details of the Solidworks construction are appended at the end of the report.

3.2 Driving Motors

The driving motors and wheels are slightly inset so that the base will have as much surface area as possible. The inset also provides physical protection from the wheels in case of a collision to the side of the robot. The wheels are attached to the motors using spacers cut out of MDF (and later, acrylic), and hot glued onto the DC motors. The DC motors are used with the H-Bridge driver provided, which simplifies the control (speed and direction) of the wheels. The wheels are halfway between the front and the back of the robot, and the differential driving method is used to control the direction. The middle mounting of the motors allows easy turning and locomotion as well.

3.3 Ball Feed and Pitching

The ball pitching mechanism is the most complicated part of the mechanical design. The ball feed consists of PVC pipes that had a cut out hole for a push-pull solenoid, which is timed to release up to two ping-pong balls at a time. The PVC pipe is just tall enough to hold five ping pong balls for the check-off run. Though we were advised that we would need a way to agitate the ball feed to prevent it from getting stuck, the PVC pipes worked well without any additional agitation, and the ping-pong balls rarely got stuck, as long as we used the smaller ping pong balls, as our PVC was not wide enough for the larger ping-pong balls. A small ball sorter separate from the robot was created to simplify the selection of the correct sized ping pong balls.

The actual pitching mechanism consists of two toy motors spinning in opposite directions, out of the robot. After the solenoid releases a ping-pong ball (or two), the spinning pitching motors propel the ping-pong ball out of the robot. The pitching motors have a variable speed set by controlling a PWM signal. The balls pick up enough speed falling from the solenoid release location to roll to the pitching motors near the front of the robot. The solenoid is positioned at a small height to provide this momentem, as a ping pong ball could otherwise jam the pitching motors. The motors need to spin up to speed before receiving a ball.

4 Electrical Design

4.1 Overview

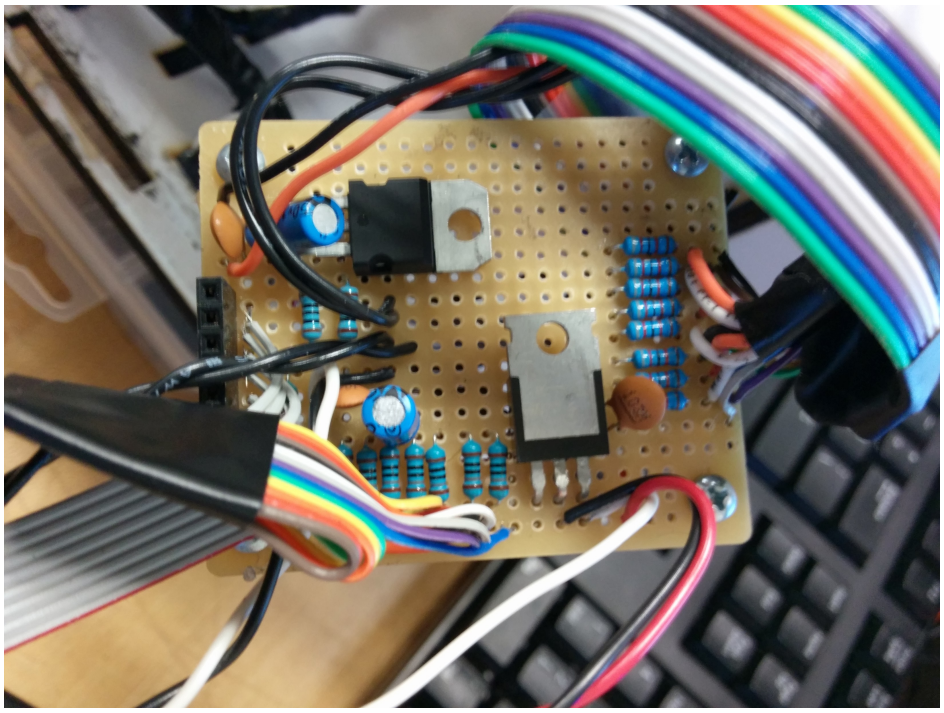


Figure 2: Perfboard circuit that included connections to most of our sensors.

Along with the PIC32 circtuitry and accompanying microcontroller, the UNO32 Stack also includes a power distribution board, which has fuses and circuit protection and distributed power from a 9.9V battery to multiple power connections.

The various electrical modules controlls the different sensors and actuators, which includes the beacon detector, tape sensors, bump sensors, pitching motors and solenoid, and driving motors. They are directly connected to the power distribution board and most use an appropriate voltage regulator (either 5V or 3.3V) to further distribute the desired voltage level. The different electrical modules are generally kept separate from each other, but are built on the same perfboard, which reduces the number of power connections. Ground wires (wires that are grounded on one end and open on the other) are wrapped around long connections to reduce electrical noise.

The schematics for the modules are appended at the end of the report.

4.2 Beacon Detector

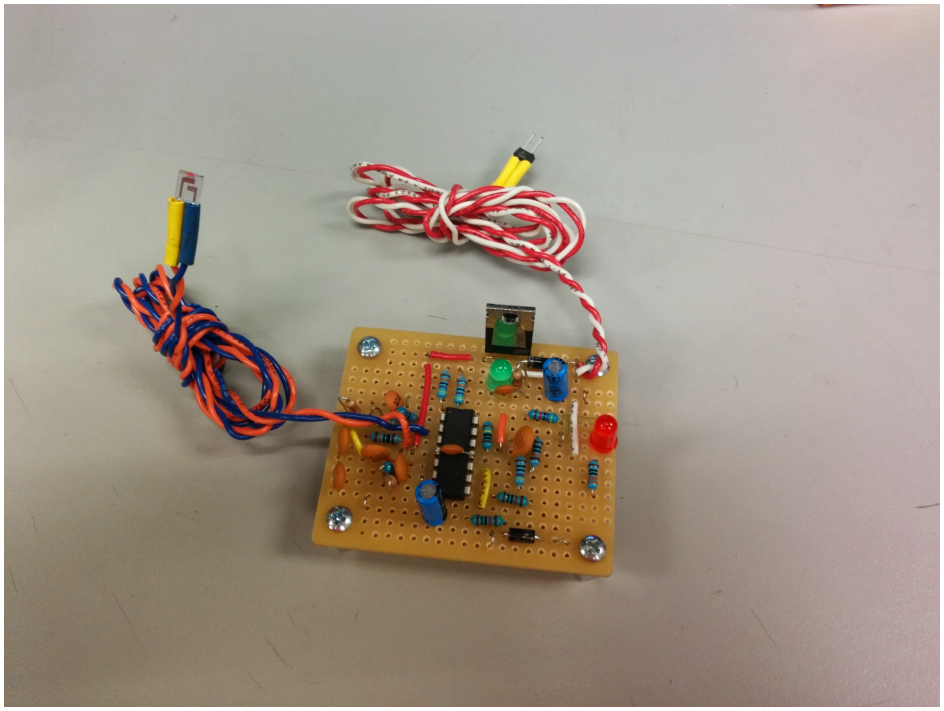


Figure 3: The beacon detector circuit on a perfboard.

Trenton Louie and Max Lichtenstein's one-chip-wonder from Lab 1 was used for our robot. The beacon works reliably up to about 8 feet, and the horizontal range is narrowed by using black tape with a vertical slit.

At some point, the beacon detector was damaged and a new circuit had to be made. Luckily, we had the circuit schematic to reference, so remaking it was time-consuming but fairly straightforward. The final circuit outputs a low signal (0V) when a beacon is detected, and a high signal (3.3V) otherwise. This output is read in as a digital value.

4.3 Tape Sensors

Each tape sensor consists of a photodiode and phototransistor pair powered by 3.3V through a voltage regulator. These were already packaged together, so using them was straightforward, and the circuit schematic is shown below. Each output is a voltage between 0V and 3.3V, which corresponds to how well the phototransistor can detect its corresponding photodiode. These outputs were read in as analog values in the software.

We started out with eight tape sensors, but ended up only using four in our software. This was due to simplifications in our state machine and realizations that not all were necessary, especially as we changed our strategy for finding the goal.

4.4 Bump Sensors

The bumper consists of two simple switches powered similarly to the tape sensors with 3.3V, with current limiting resistors to prevent shorting. When the bumper is pressed, one or both of the switches would be triggered and would output a high signal to the UNO32 stack.

4.5 Pitching Motors and Solenoid

The pitching motors are powered directly from the stack and connected in series with the Darlington, and the leads of one of the motors reversed so that the motors would spin in opposite directions. A PWM signal is fed into the base of the Darlington, and is used to control the speed of the pitching motors. A clamping diode is placed across the leads of the motors to attenuate the effects of the back-EMF.

The solenoid, like the pitching motors, is powered directly from the stack and connected in series with the Darlington. A regular digital signal is fed into the base of the Darlington, which extends the solenoid when high, and retracts the solenoid when low. Though not pictured in the schematic, a clamping diode is also placed across the leads of solenoid.

4.6 Driving Motors

The driving motors are implemented using geared DC motors with the provided Dual H-Bridge module, which allowed for us to easily control the speed and and direction of both of the motors using signals from the UNO32 Stack.

5 Software Design

5.1 Overview

The state machine is written using the Events and Services Framework provided. This framework handles the background details of event driven programming, and as a result, our team only needed to create events that would drive the state of the robot, and the functions that defines what the robot will carry out in the appropriate state.

5.2 Events

To start, the events of interest are defined and created. Each event type has an event parameter that is used to further specify the details of the event. For this robot, the main user defined events corresponds to changes in the state of the sensors. The three types of sensor-driven events used in our state machine are outlined below.

1. BEACONCHANGE:

   This event is posted if there is a change in the status of the beacon detection. The event checker checks the digital input of our beacon detector circuit. If there is a change, the parameter is set to 1 if the beacon is detected, and 0 otherwise, and the event is posted to the state machine.

2. BUMPCHANGE:

   This event is triggered if there is a change in the status of the bump sensors. The event checker checks the the bump sensors, which are read in as digital inputs and concatenated into a bit value. If there is a change, the event parameter is set to that bit value, and the event is posted. Individual bump sensors are checked by bitmasking the event parameter in the state machine.

3. TAPECHANGE:

   Like the previous events, this event is triggered if there is a change in the status of the tape sensors. The tape sensors are read in as an analog value, and a threshold, which was determined experimentally, was set to determine whether the sensor is on black tape or not. A bit value of 1 indicates that a tape sensor is detecting black tape, and 0 indicates otherwise. As with the bumpers, the bit values are concatenated into one value in which each bit value corresponds to a different tape sensor. If an event is detected, the parameter is set to the bit value, and the event is posted.

Two additional user-defined events, DONE_EVADING and DONE_FOLLOWINGTAPE are used to exit the sub HSMs for bumping and following the tape, respectively.

User-defined timers were also used in our state machine to make the robot perform specific actions, such as moving forward the length of a star, or turning approximately 45 degrees left. These timers create an ES_TIMEOUT event that also drives the state machine. The values of these timers were determined experimentally, and will be omitted from this report for the sake of conciseness.

5.3 Functions

Functions were written to control the pitching motors, solenoid, and driving motors of the robot, and were called in the appropriate states. The pitching motors can be driven at different speeds, the solenoid can be 'on' (extended) or 'off' (retracted), and the driving motors can be driven at different speeds and in different directions. The functions used are listed below.

- FullStop(): Stops the robot completely by turning both wheels off..
- MoveStraight(x): Moves the robot forward at a certain speed, specified by the variable x. Both wheels are set to move forward at the same speed.
- BackStraight(x): Same as MoveStraight, but backwards.
- SpinLeft(x): Rotates counterclockwise about the center of the robot at a certain speed, specified by the variable x. The right wheel is set to move forward, while the left wheel is set to reverse.
- SpinRight(x): Same as SpinLeft, but clockwise.
- SolenoidOn(): Extends the solenoid to release the ping pong ball.
- SolenoidOff(): Retracts the solenoid.
- PitchingOn(x): Runs the pitching motors at a certain speed, specified by the variable x.

5.4 State Machine

The final state machine for the robot is simple (much simpler than we had originally planned), and the sequence of actions the robot will take is described as follows:

1. **Orient towards the enemy beacon.** The robot initially spins to find the enemy beacon and orient itself roughly towards the gate.
2. **Turn left and move forward.** The robot then makes turn and immediately moves forward to try to find the left wall. The general goal is to use the tape to guide us to the enemy gate.
3. **Roughly follow the wall (black tape).** The robot should be able to reach the black tape on the left border of the field and follow the tape. The robot follows the tape by waiting until it detects black tape with one sensor, slowing down until it detects another tape sensor, and then spinning itself until no more tape is detected. This method allows for the bot to follow the tape by "hugging" the wall, without strictly following the tape.
4. **Avoid obstacles by arcing around it.** If the robot bumps into an object, it immediately backs up, turns slightly, and makes an arcing motion to avoid the object. This sequence of events used timers, whose times we determined experimentally. By arcing around the object, the robot should be able to reach the tape again. If the object happens to be the gate, the robot will arc right into the enemy gate.
5. **Turret control FSM.** Later a FSM was implemented to run in parallel with our HSM. The FSM was simple and only certain events were posted to it. A triple beacon detector posted the three individual sensors as events. The left sensor make the turret turn left, right sensor made the turret turn right, center made the turret stand still. While turning left if the turrets servo value reached or passed a certain threshold it would switch states to turn the opposite way and vise versa for turning right. The center beacon detector detector was the only event that was posted to both state machines.

This state machine got the robot through the gate almost every time during check off. The full details of the sub-state machines will be appended at the end of the report.

6 Integration

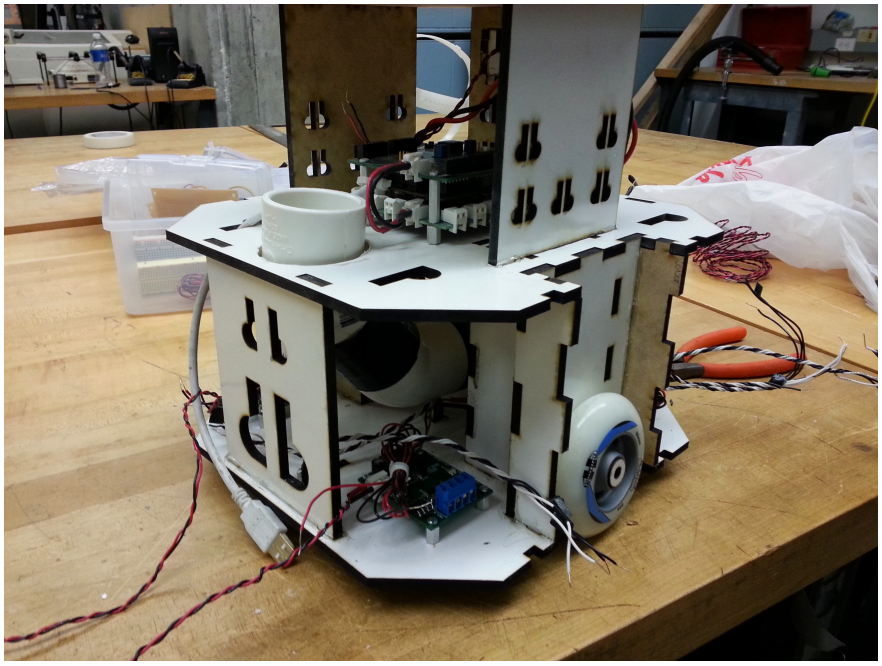


Figure 4: A partially assembled MDF model, showing the top tier, which holds the UNO Stack and the wiring, along with the bottom tier, which holds the motors and drivers.

Some of the integration was done while the mechanical and electrical parts were being designed. When setting up the ES Framework, the 3.3V rail from the ATX power source was used in conjunction with various resistors and potentiometers to simulate digital and analog inputs from the sensors. Doing this allowed for us to write and debug the software functions and events before the electrical hardware was ready. By triggering events and using TattleTale, we were also able to debug the function and transitions of the state machine.

When the physical robot was ready, we were able to test how the state machine will actually make the robot behave. At this point, most of our time was spent refining timers and speeds, and tracking down bugs to fix.

7 Results

7.1 99 Problems and We Debugged All 99 Of Them

Figure 5: Bugs. (Screencap taken from http://www.endersansible.com/.)

- Electrical noise was probably the most problematic part of our system. One of the most difficult bugs to track was when the noise from our pitching motors created enough noise to trigger the bump sensors, which would then trigger an event and force our robot into the wrong state. The delay in our test harness was long enough that we were still not able to detect the noise. We were finally able to figure out that the bump sensors were being "triggered" due to electrical noise when we noticed that state machine worked perfectly when the pitching motors were either turned off, or driven at full speed (100% duty cycle). We deduced that controlling the motors with a PWM signal was causing huge voltage spikes and creating noise in our circuit. We tried to attenuate the noise by adding clamping diodes, wrapping ground wires, shielding with foil, and adding in a small delay in the event checker that served as a simple "button bounce" for our input.

- Wiring all of our circuits on one perfboard seemed neater and more convenient in theory, but ended up making debugging difficult because it was not modular. It might have also been part of the reason why there was so much noise in our circuit, which was reduced using lots of coupling capacitors. Wiring each part separately would have made it easy for us to detach certain modules while testing individual modules. The final product was nearly impossible to debug because even with a minimal number of components, our wires were completely tangled. Using small coupling capacitors at the UNO input might have also helped with the transient noise from our sensors.

- Using service modules in the software might have helped reduce the transient noise and threshold errors we ran into. None of us looked into using a service module, as we believed that an event checker would suffice. However, having a timed service module means we would

not need to add in blocking code in the event checker to get rid of the transient noise. Our blocking code is miniscule, but is bad practice for larger projects.

- We did not create multiple test harness for the different modules of our robot, and stuck with commenting and uncommenting a define to use the test harness. This worked fine for the most part, except that most of us had trouble finding which file the test harness was in, and was thus unable to use it.

- We did not account for the fact that we would need to disassemble and reassemble our robot in order to make adjustments to our wiring. This slowed down the process of debugging, as we had to take apart the entire robot every time we needed to make a significant change to our circuitry. We did not anticipate that we would need to make changes to the wiring.

- We did not have a design for our bumper because we all assume it would be trivial. At the last minute, we hot glued a piece of scrap MDF on two switches with a spring to prevent the switch from getting stuck. Our bump sensors broke numerous times during the last couple of days while testing and debugging the system. Each time it broke, we would have to wait another couple of hours before we would start testing again, as we would need to redesign the bump sensor and lasercut the new piece. Our wheels also came loose multiple times, and would also delay our progress for a couple hours each time. Having parts of the robot break repeatly resulted in a significant amount of time wasted trying to fix things.

- The bumpers were also not very sensitive, and we would often get stuck on an object because a bump would not be detected. During the final competition, we disqualified in the first round because we were not able to resolve a collision in under five seconds. If a proper bump change had been detected, the robot would have immediately reversed away from the object, and resolved the collision.

- There were lots of disagreements on how we should tackle this project, and we finally ended up going into the project without any plan. We were unable to make concrete design desisions (number of tape sensors, number of bump sensors, etc.) because some of us thought that our design decisions would depend on our state machine, while others believed that our state machine should be based on our design decisions. Because of this circular reasoning, we were also never able to decide on state machine in the first place, because we would always try to consider a corner case. We ended up having eight tape sensors wired up, while we only used four, and the bump sensors were just glued on at the end. We also never had a set state machine to implement, and our final state machine was mainly done by trial and error.

- Our documentation was effectively non-existent. This made our team tremendously inefficient as we were not able to pick up where another teammate left off. This was especially problematic during the last couple of days when we were trying to finalize our state machine, when we were trying to fix the electrical noise in our circuit. Because we had no schematic to reference, we spent hours just trying to map out the circuit. Even when all the teammates were present, nobody would know the full details of the circuit because there was no record of it. When trying to debug and finalize our state machine, not all of us knew how to debug using the test harness or TattleTale, so we would be stuck until somebody who understood the framework was present.

7.2 Final Competition

The team met minimum specifications about a week before the final competition. Though we were technically done, Trenton and Russell made some final changes to the robot to make it look cleaner, and Angel managed to build a functional turret in just a couple days (all while Julie went M.I.A and made some futile attempts at fixing her sleep schedule). Our robot got disqualified in the first round because our bump sensor did not trigger and it was unable to resolve its collision in less than five seconds. The bump sensors got bent in a way such that it became less sensitive to physical bumps. Also, in attempting to attenuate noise from the pitching motors designing the turret, the solenoid fell out of place and did not release the ball properly. Despite this, our robot put up a good fight and our class put up a good show for everyone.

8 Conclusion

Despite all our challenges, we were able to successfully complete the project. We were fortunate enough to have a larger team of four to compensate for all the problems that we encountered. Keeping ours simple was arguably the most crucial element to our success—we managed to check off without having to implement any extra sensors aside from the bump sensors and the tape sensors. Though we were not able to complete the beer challenge as we had hoped, we were still able to check off on the same day.

In retrospect, no single element of this project was particularly difficult. The difficulty of this project comes from the implementation and integration of all the elements, during which we learned to track down a problem when there are so many possible sources of that problem. Despite all the frustration and confusion that came along with this project, we have all gained much experience from this project. We can all now officially say that we have built a robot in under five weeks, which makes all the sleepless nights worth it in the end.

A Solidworks

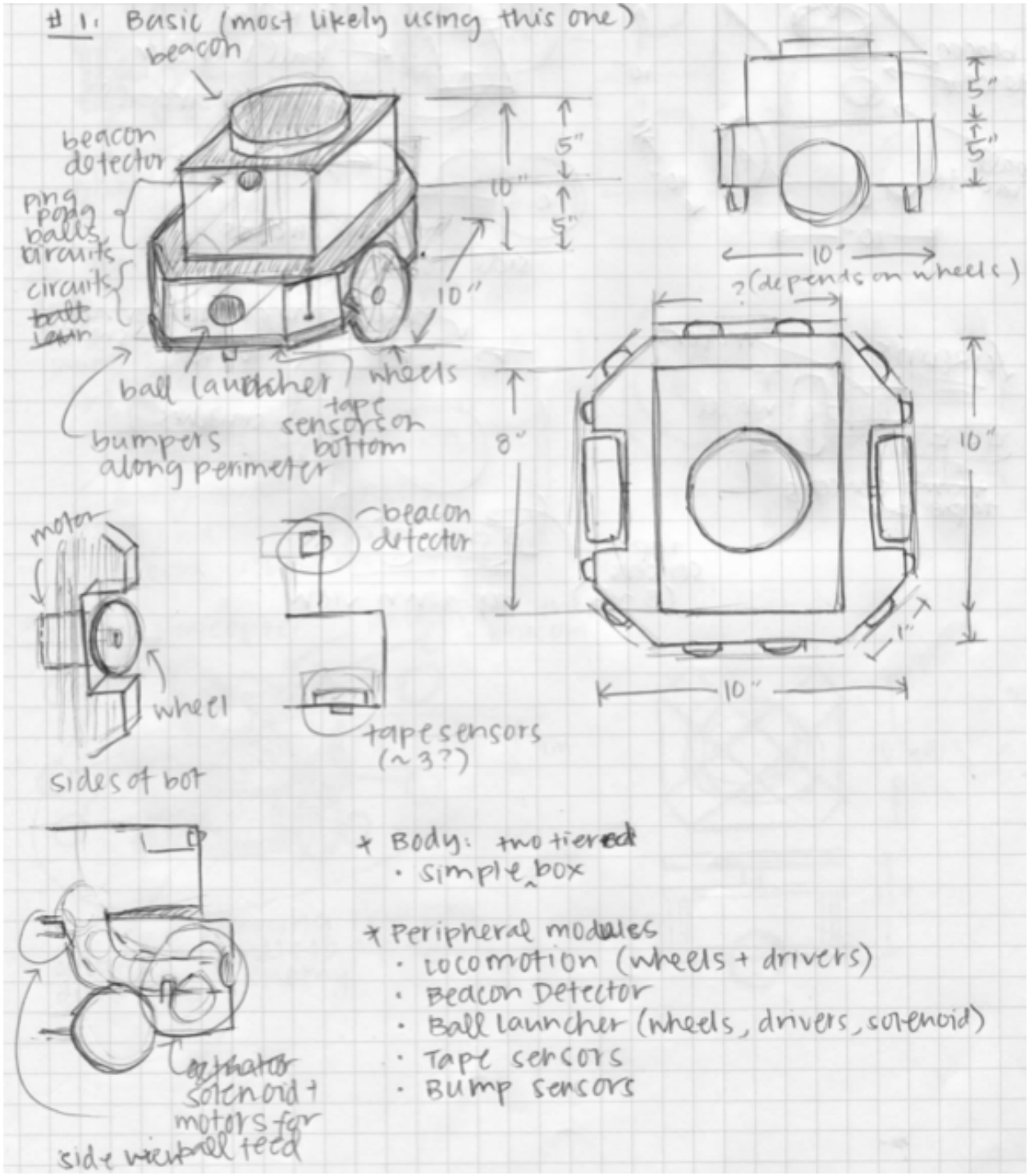


Figure 6: A first draft of the model.

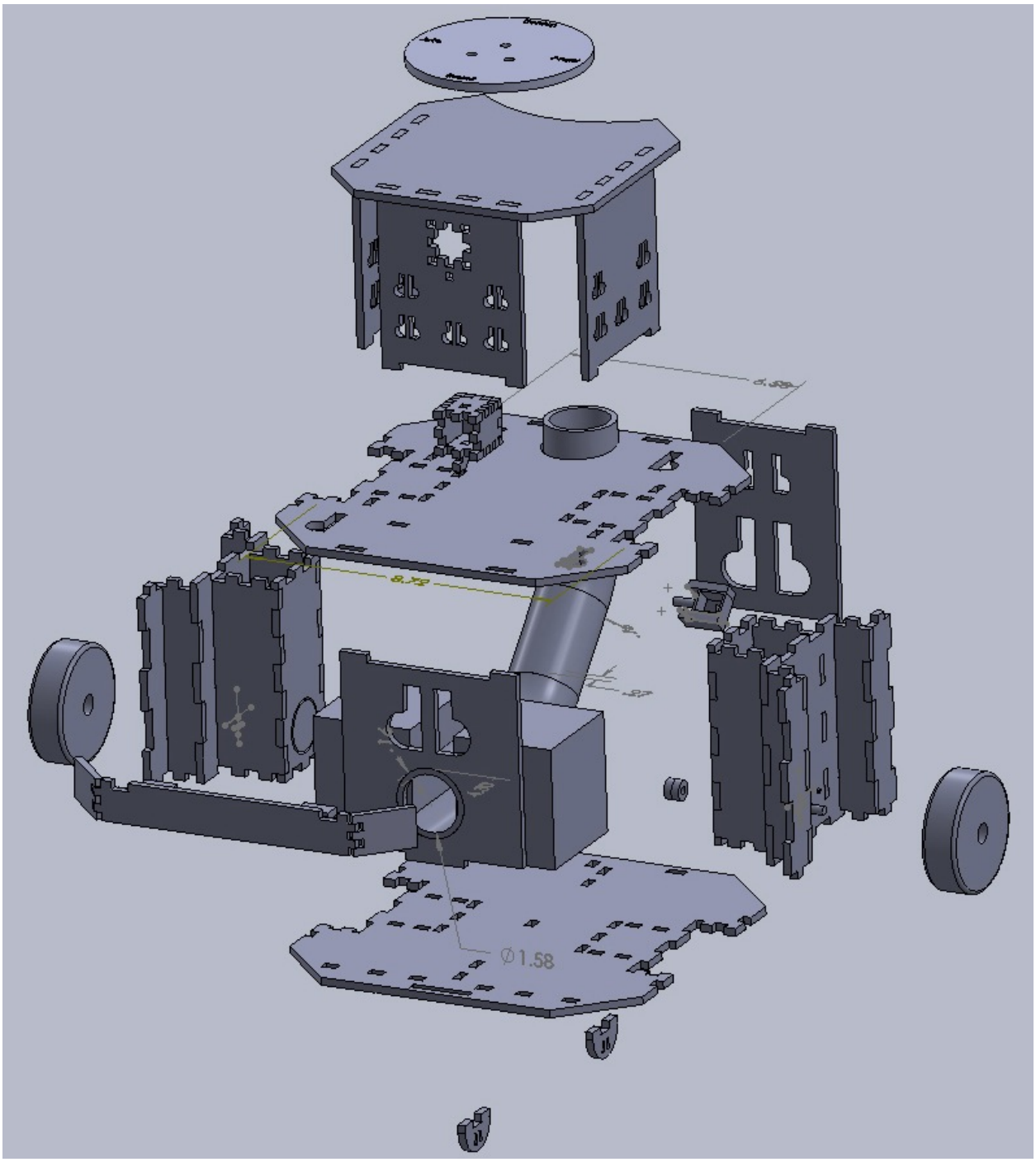


Figure 7: An exploded view of the Solidworks model, showing the different components of the robot.

B Schematics

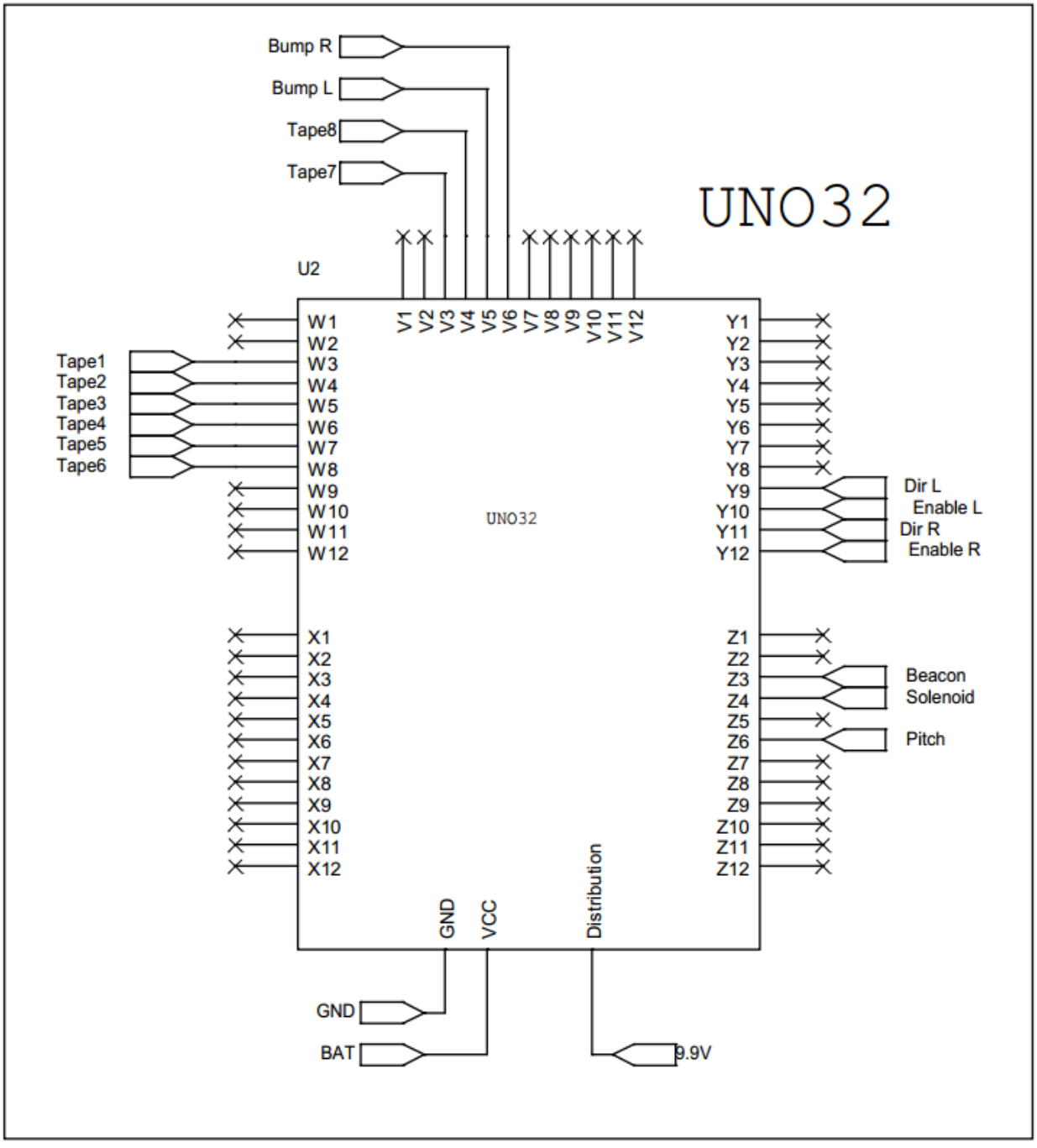


Figure 8: The UNO Stack connections.

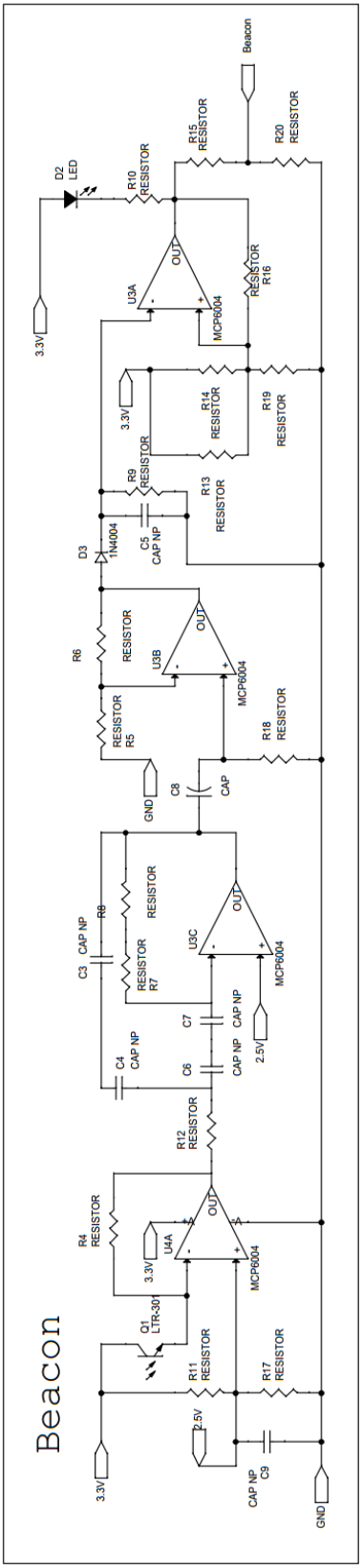


Figure 9: The beacon detector circuit schematic.

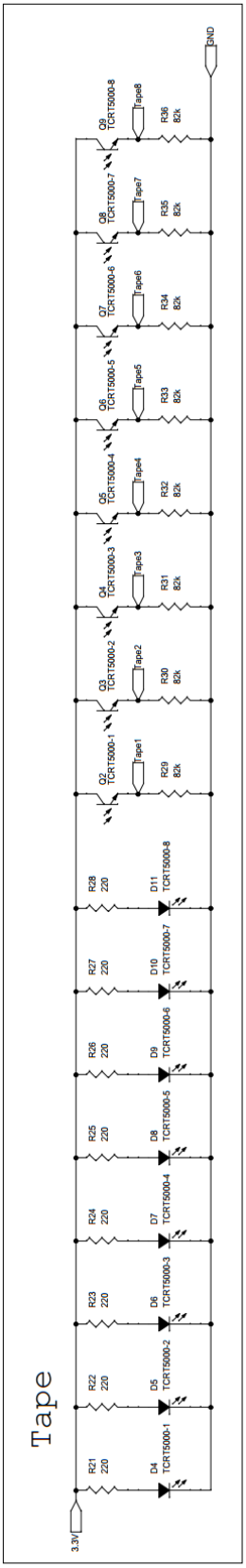


Figure 10: The tape sensor circuit schematic.

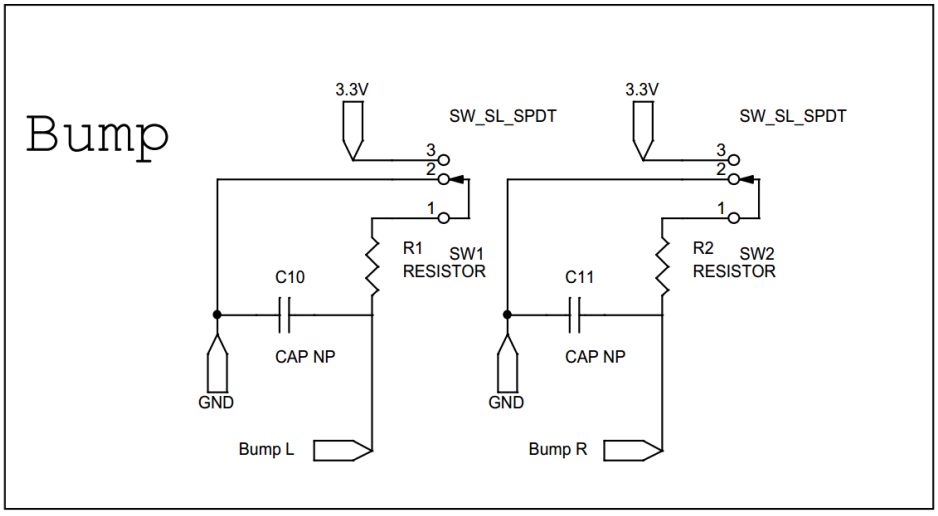


Figure 11: The bump sensor circuit schematic.

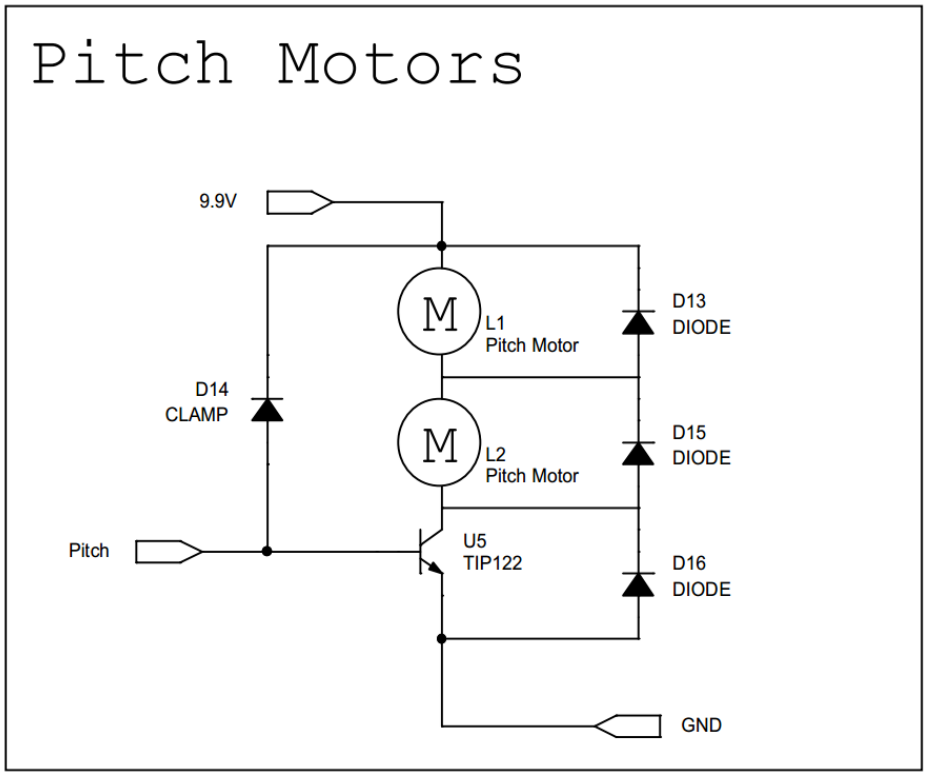


Figure 12: The pitching motors circuit schematic.

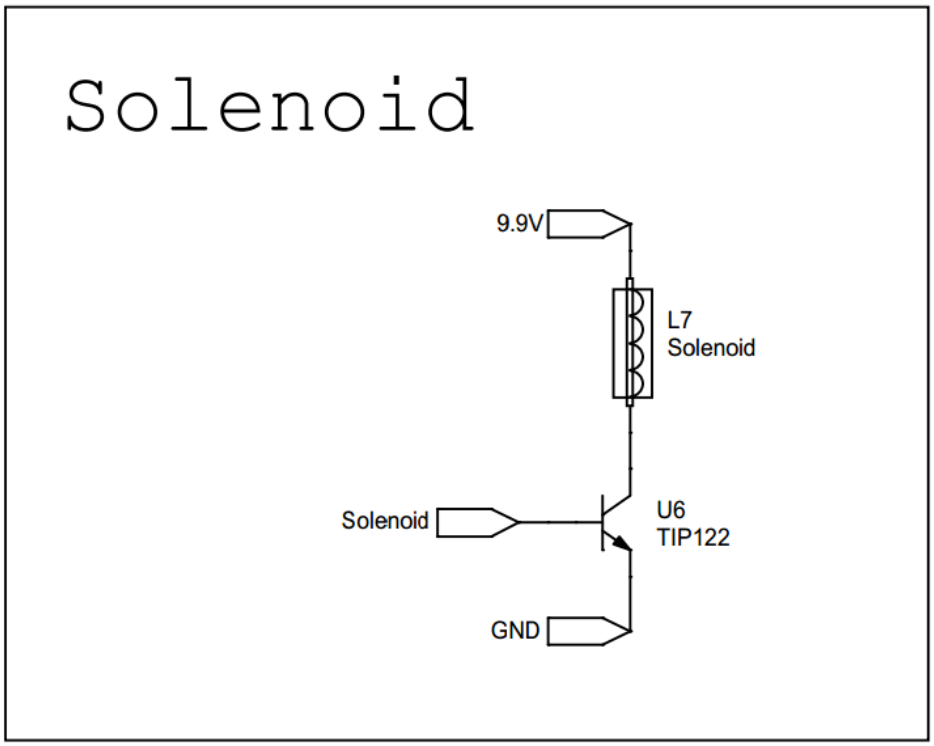


Figure 13: The solenoid circuit schematic.

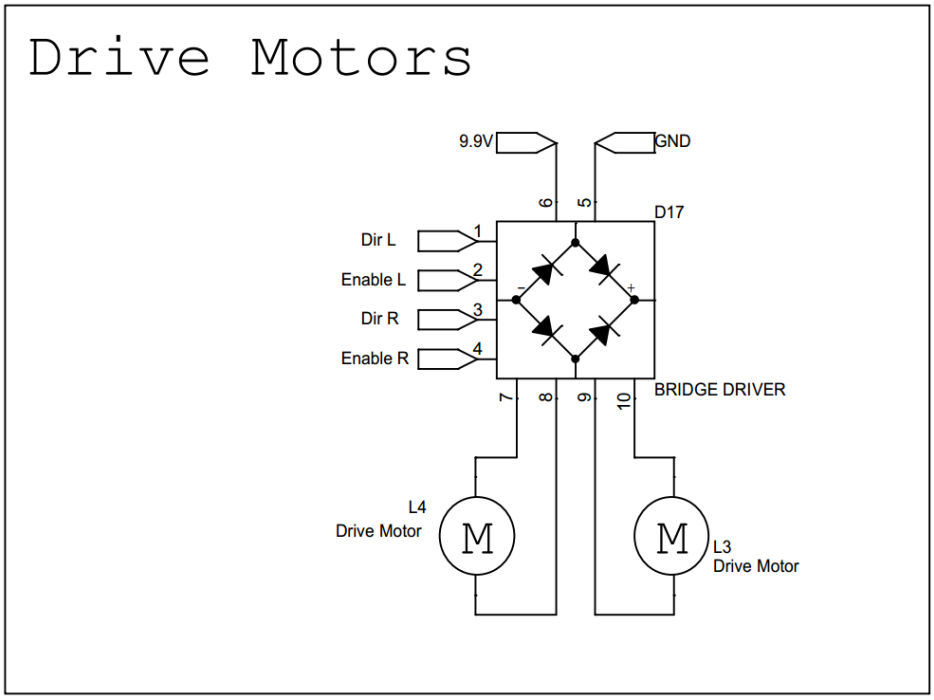


Figure 14: The driving motors circuit schematic.

C State Machine

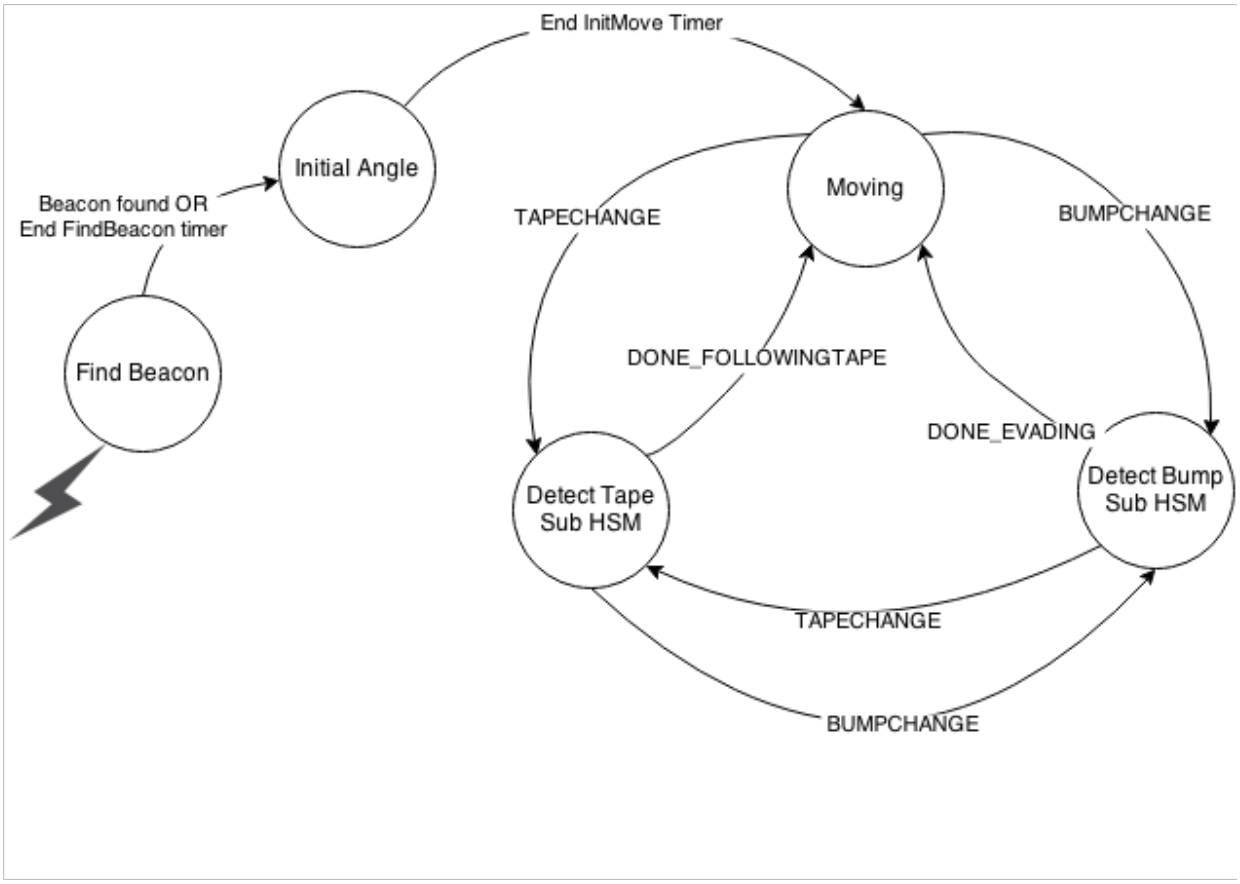


Figure 15: The top-level hierarchal state diagram for our robot.

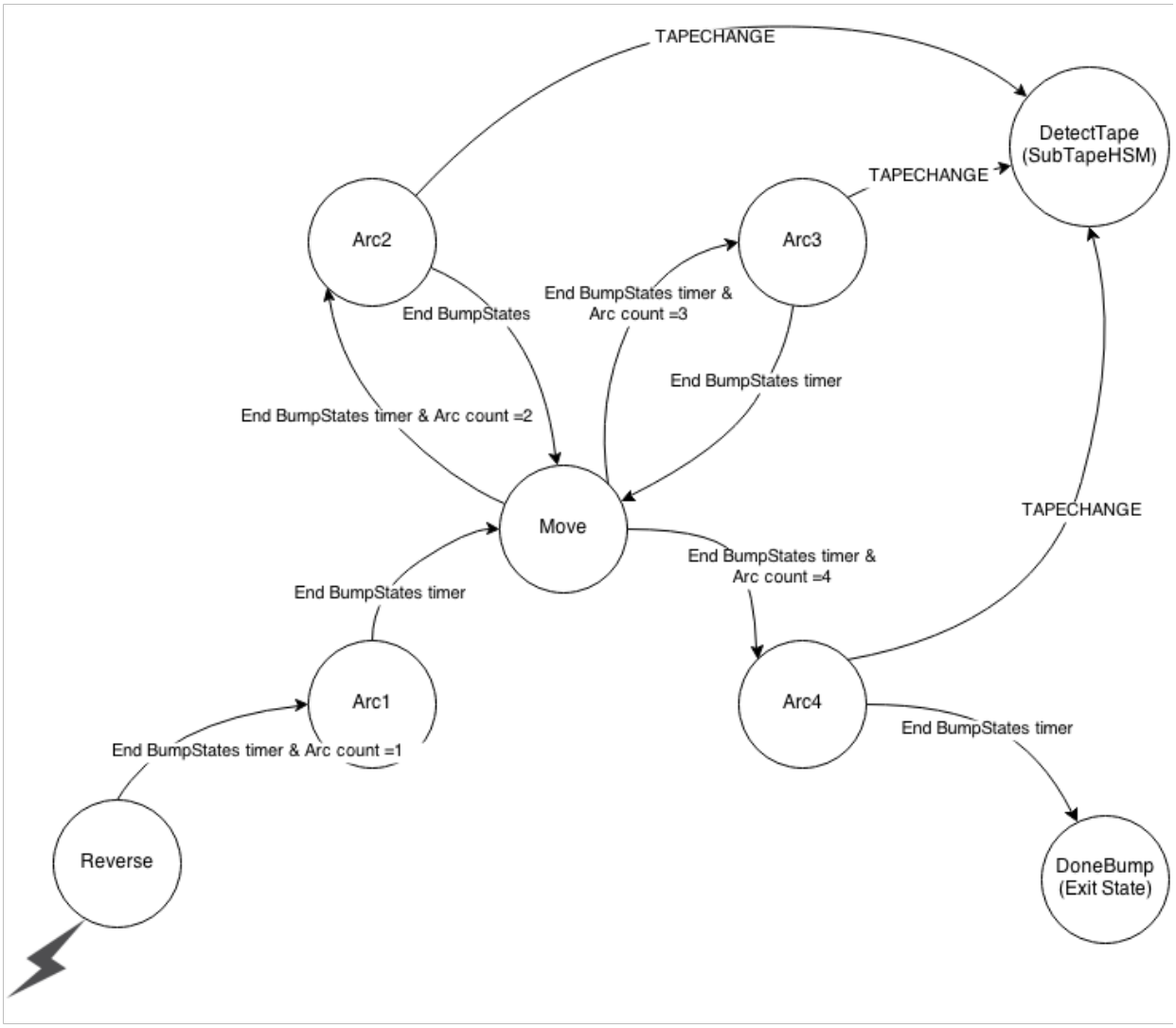


Figure 16: The state diagram for the bumped sub-HSM.

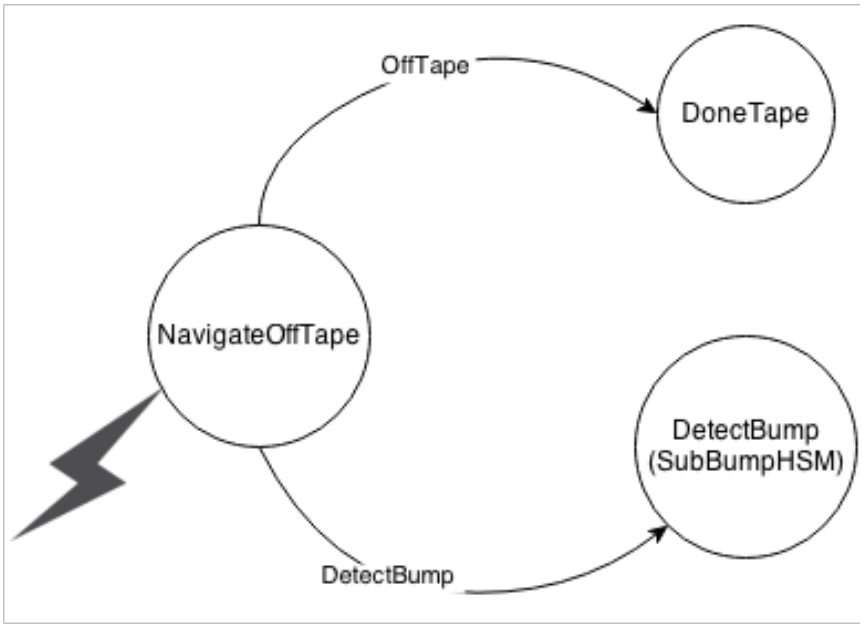


Figure 17: The state diagram for the tape sub-HSM.

D Bill of Materials

components	usage	notes	quantity	unit price	total price	total (w/o tax & shipping)
DC Motors	locomotion	jameco	3	$14.95	$44.85	$119.07
Toy Motor Drive	ball launcher	mpja	2	$3.5	$7	
MDF board	structure		2	$5	$10	
hard wheels	locomotion		4	$2	$8	
switches	bump sensors	jameco	10	$0.89	$8.9	
perf board, large	circuitry	mpja	1	$1.75	$1.75	
perf board, medium	beacon filter	mpja	2	$1.1	$2.2	
perf board, small	tape sensors, collision sensors	mpja	2	$0.75	$1.5	
solenoid	release mechanism	amazon	1	$9.35	$9.35	
foamcore sheet	prototype		1	$3	$3	
screws/washers/nuts	motor mounting, boards to MDF		hella		$0	
fuse	fuse		1	$0.75	$0.75	
duct tape			1	$4.99	$4.99	
pvc elbow joints	ball launcher		2	$1.99	$3.98	
paint			1		$0	
super glue			1		$0	
brush			1	$12.8	$12.8	

Figure 18: The final bill of materials.