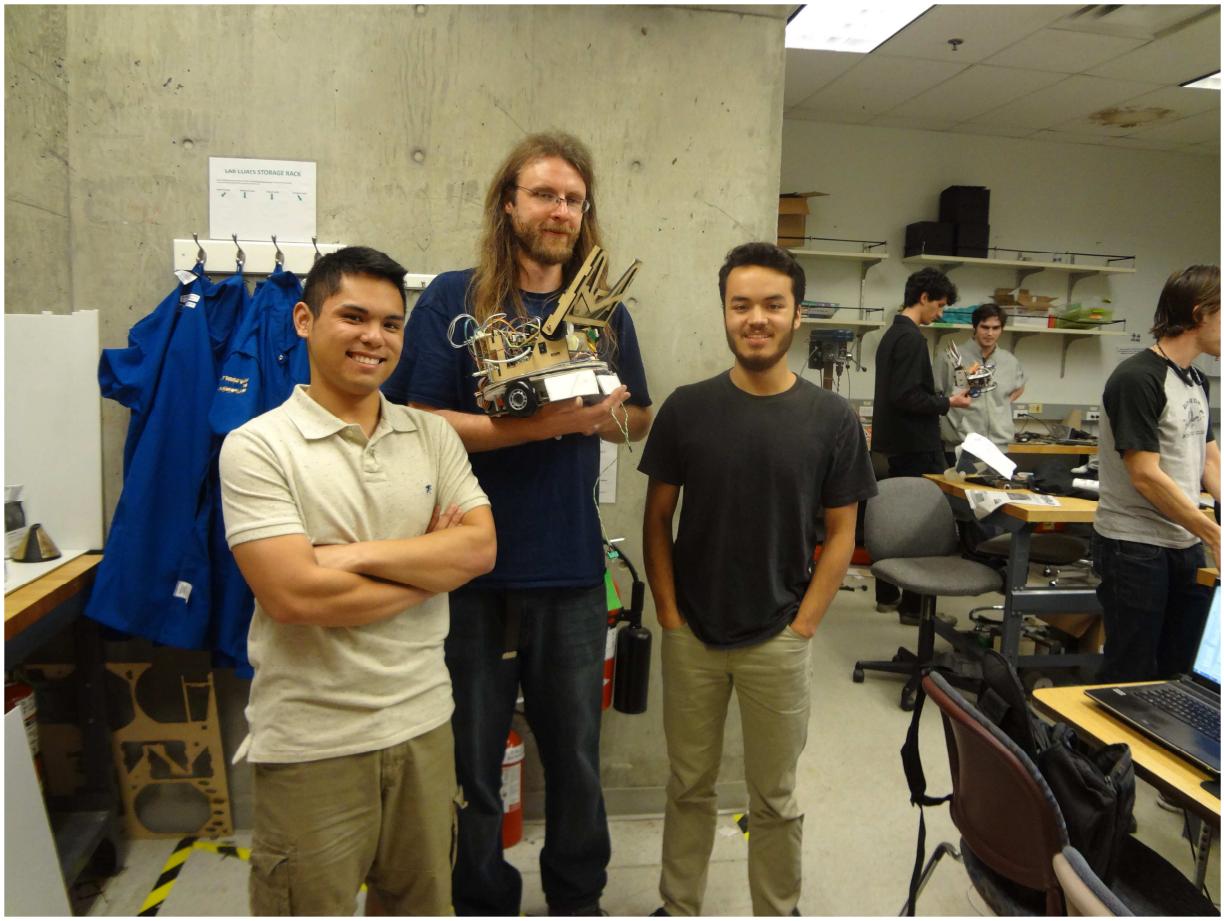
UNIVERSITY OF CALIFORNIA, SANTA CRUZ
CE 118 FALL 2014

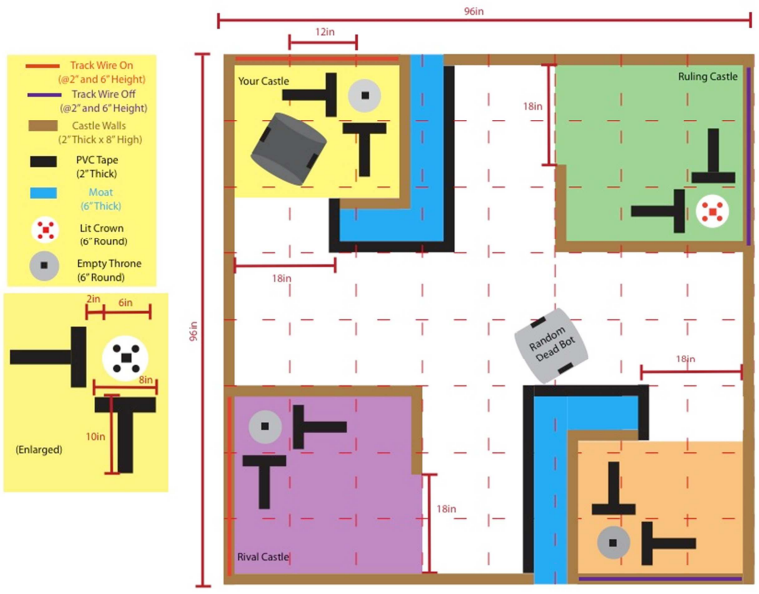
Final Robot Design Report
TEAM MINSPEC



Martin Freerks (Middle)
Reese Robertson (Right)
Matthew Silva (Left)

12/19/2014

Introduction:



Our team was given a task to build a robot that would be able to navigate a field from a specified location and retrieve an item from somewhere on the field and bring it back to some point near it's starting point.

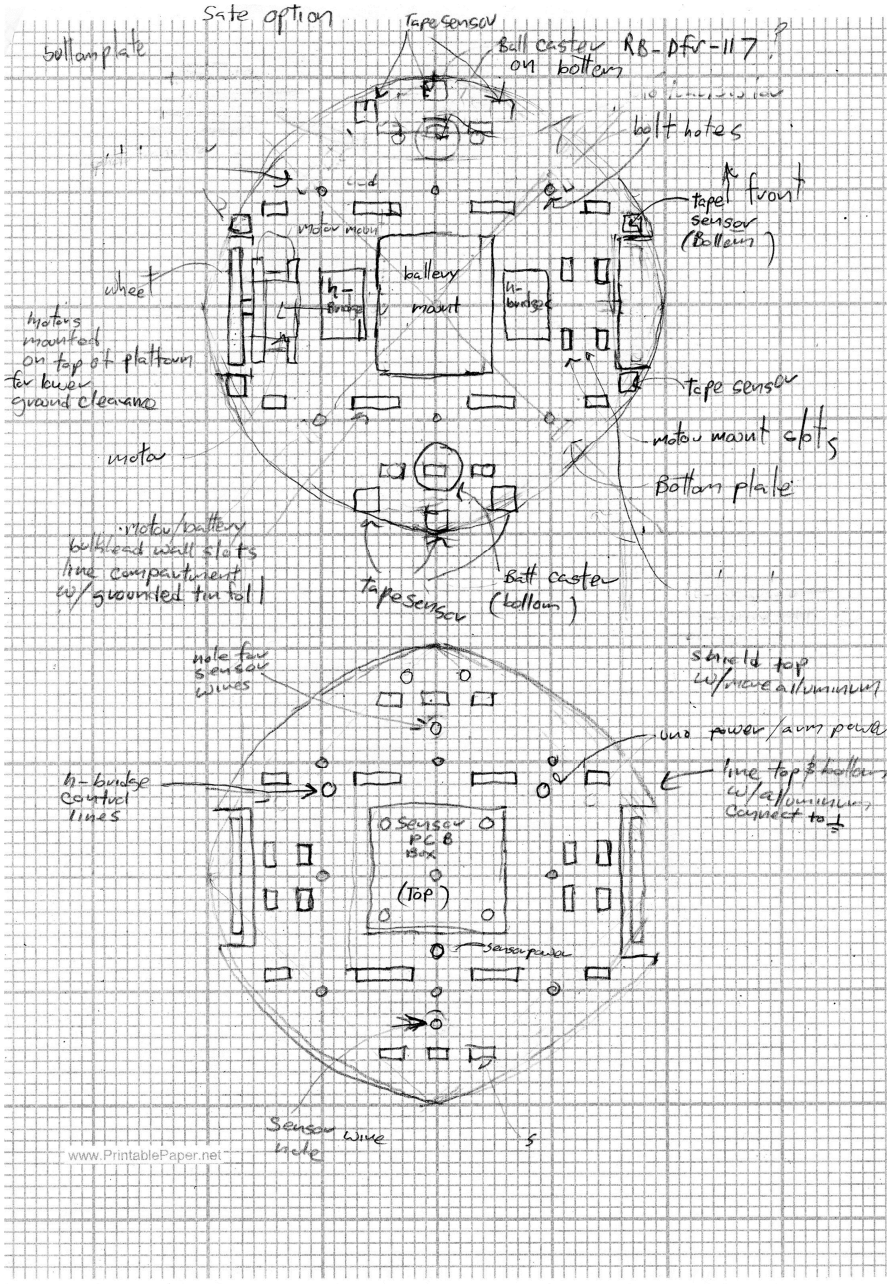
The figure on the right is how the playing field will look. Our bot will be placed in 1 of the 4 corners (castles) of the field. The grey circle with a black dot in the center represents a beacon, which our bot should be able to detect. Our bot needs to be able to exit it's home castle, where there will be a 'ON' track wire, then it will travel to all the other castles and look inside to see if the beacon is lit, which means there is a crown on top of it. Once the bot sees the beacon it needs to approach the crown and pick up the throne, leave the enemy castle and get back to it's home castle. Once the bot is in its home castle it will need to place it's crown on it's home castle's throne. When the bot is not in a castle there will be obstacles such as a dead bot and motes, which are depressions on the field.

This is our team website: **https://sites.google.com/a/ucsc.edu/minspec/**

Hardware Design

Methods:

After reading the Game of Drones Overview and understanding the rules, we sat down as a team and planned out what sensors we needed to use and how we were going to use them. We looked at some of the designs each of us made for the midterm and took the good ideas of each design and put it into the one design that we were going to build for the project. We centered our ideas around one of Martin's design, shown in figure 1.



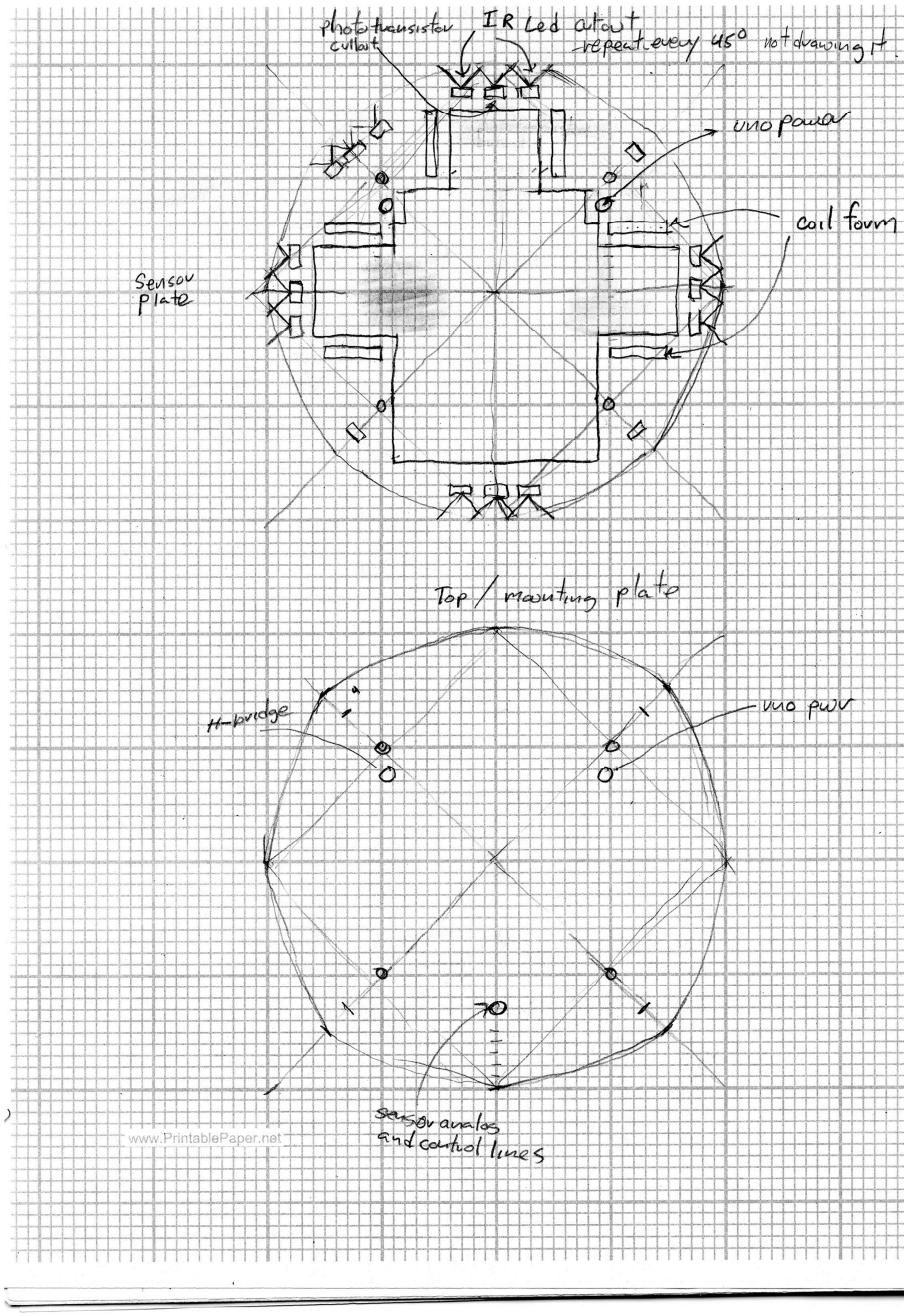


Figure 1: Martins Design Sketch

What we kept and modified from Martin's design:

- We kept a circular base, we did not want any part of our bot getting stuck on any corners of the field.
- Instead of having 8 banks of phototransistor cutouts, we decided to try and implement 4 banks. These phototransistor banks would be responsible for detecting the IR beacon as well as being used to sense distance.
- We wanted our bot to be modular and easy to disassemble, so we liked the idea of having 4 quarter inch bolts locking our bot together.

- We decided to have our motors mounted on top of the bottom base platform of our bot instead of under the bot, mainly because we wanted our bot to be relatively close to the ground so that we could get good tape sensor readings.
- For the tape sensors, we restricted ourselves to using 8, we would place 3 tape sensors on the front in a triangle formation, one at the front and rear of each wheel, and one more tape sensor in the rear of the bot which gives us the grand total of 8 tape sensor.
- As for bump sensors, we decided to use 4 of them, one for every corner of the bot with whiskers attached to extend the range of the bump sensors.
- Also, at this point in the project we were experimenting with the idea of using encoders to track the distance the bot was traveling, to make traversing the playing field easier for our bot
- We decided we would use Muxes to cycle through the groups of our sensors and read their values

Now that we had a foundation to build from, we started making the SolidWork parts and assemblies. We made our base platform a 10 x 10in circle with wheel cut outs. For our second iteration of the base we added cutouts for inserting our tape sensors.

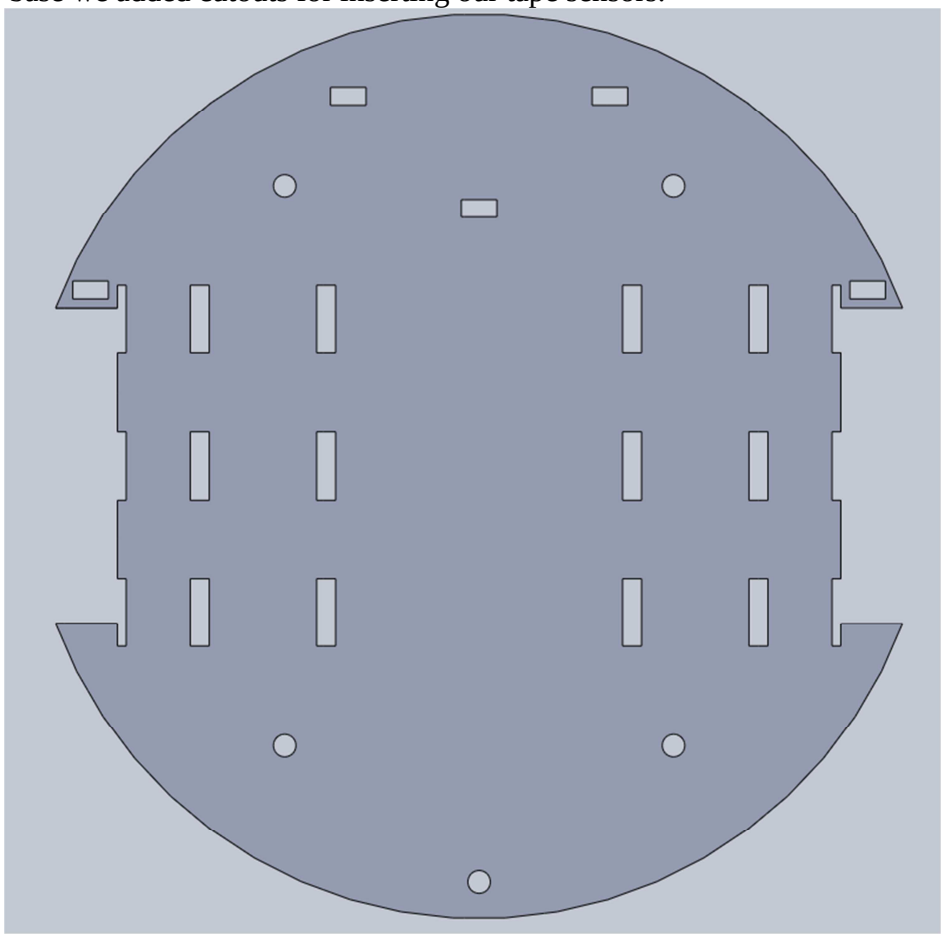


Figure 2: SolidWorks Base

Then once we got our motors from the MPJA bulk order, we made the motor mounts. We were able to find the datasheet online by typing: “Buhler DC Gear Motor 1.61.046.XXX,” which

gave us all the necessary dimensions to build a 3-D model on SolidWorks. As we were building our motor mounts we noticed that the motor shaft was really short and we had to find a thinner material to screw the front face of the motor to, Because of this constraint, we needed to use acrylic with a thickness of .125 inches. We thought we would be able to implement some type of encoder, so just incase we get the capability of encoding our motors, we put cutouts into the mounts for installing encoders. We later took the encoder cutouts because we were not able to implement encoders. For our wheels, we made couplers out of quarter inch thick acrylic to interface with the D-shaft of the driving motors. Reese had a lathe at his home and attempted make our team some aluminum shaft extenders, but we had problems of our wheels being too wobbly.

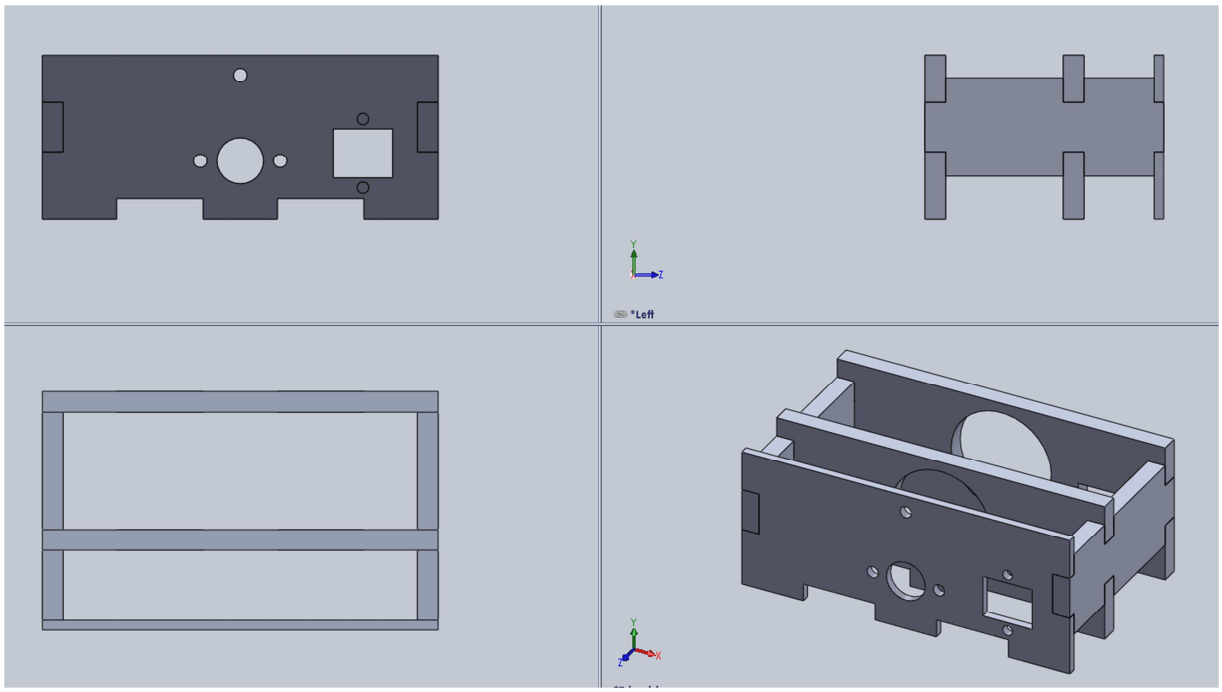


Figure 3: Motor Mount Assembly

After making the motor mounts, we had an idea of sandwiching a layer of MDF between two foamcore layers that would hold our beacon detector and distance sensors. On the bottom layer foamcore, we put cut outs for the sensors and LEDs in the North, East, South, and West orientation of the bot. To make accessing parts in our bot easier, we made a center cut out which extends to the other pieces in this assembly.

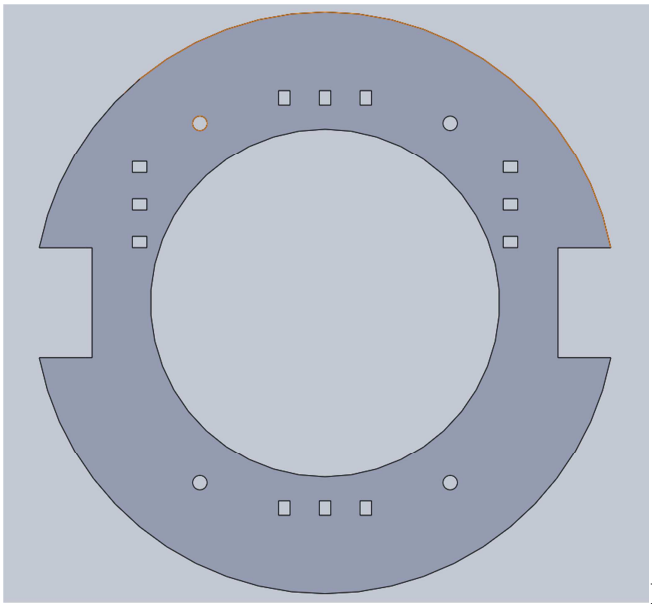


Figure 4: Bottom Foamcore Ring

On the middle MDF layer we put saw tooth like cutouts again for the beacon/distance sensors. This is the layer where the sensors and LEDs would be emitting the IR light and collecting the IR light. We called this piece the Sensor Ring Divider. We made the cutouts straight instead of at an angle because we wanted more accuracy from our sensors. We ended up not even using the sensor ring, mainly because we did not complete making the distance sensors. We were able to complete the beacon detectors, but we ended up using electrical tape to fasten one of the 4 beacon sensors to the front of our bot. One of the reasons why we did this is because the wire of the beacon detector was too short and we did not want to extend it, also it worked fine just being taped to the front.

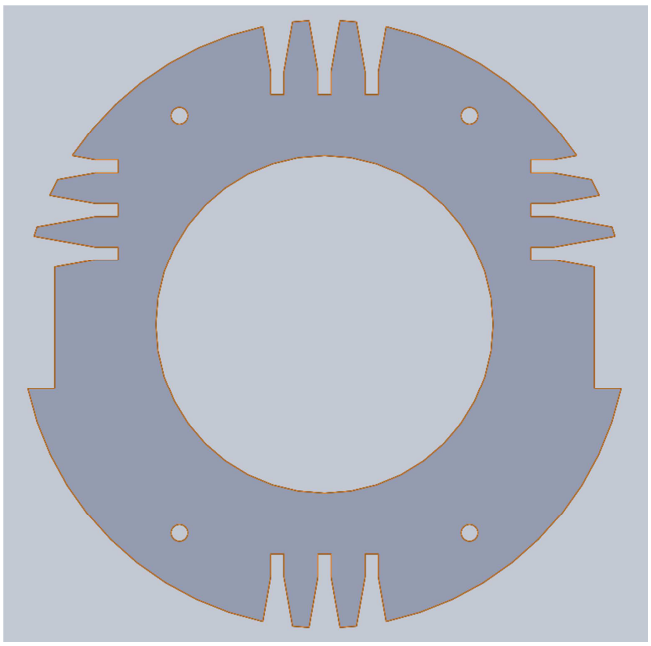


Figure 5: Center Ring Divider

For the top layer of the Sensor Ring, we just had a flat piece of foamcore, just to help focus the sensors, so that the IR light could only emit in the outward direction, not up or down.

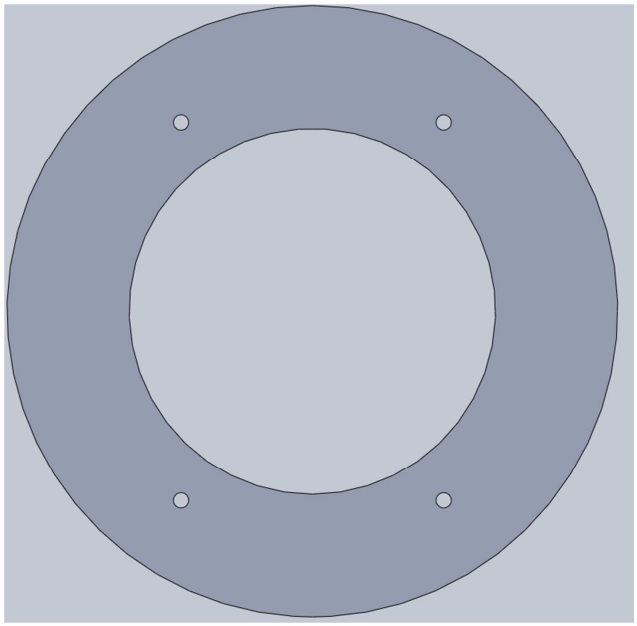


Figure 6: Top Ring

Now that we had a basic shape of how our bot was going to look like, we started the construction or the Lifting Arm Grabber. The Lifting Arm Grabber was made very wide with a “U-shape” cutout so that if we did not align very accurately with the throne, we would still be able to successfully lift the crown. We later found out that the Lifting Arm Grabber would be able to shift the bot’s center of gravity very easily, so we tried to remove as much weight as possible. Even with all the MDF we were able to shave off the Lifting Arm Grabber was still pretty hefty.

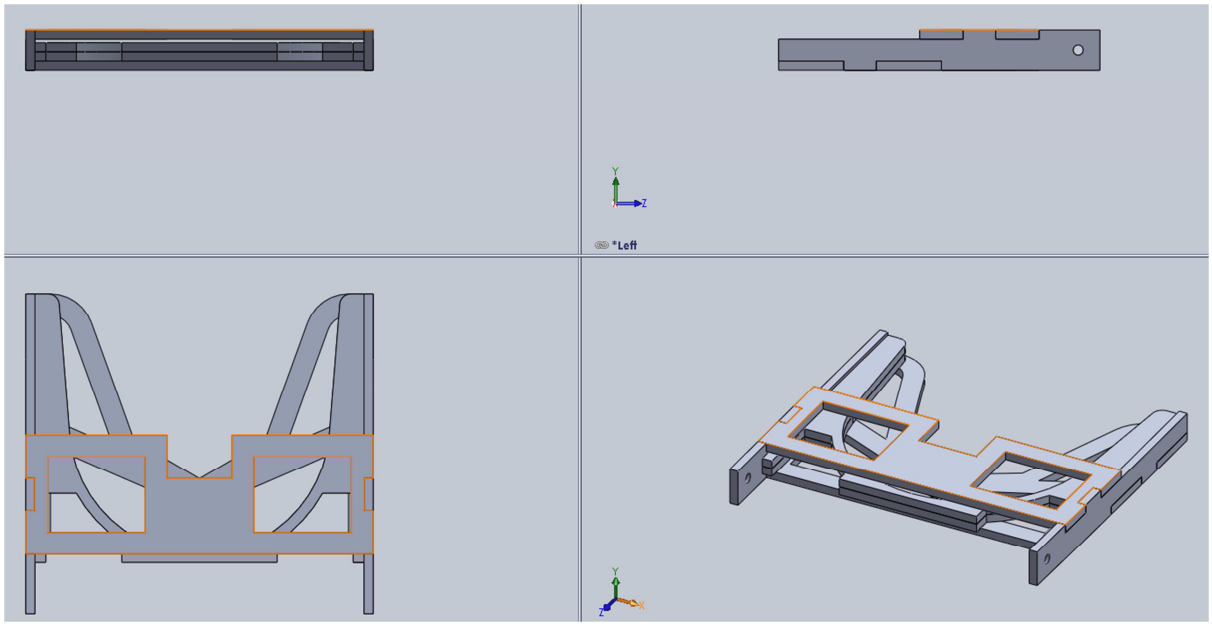


Figure 7: Lifter Arm Grabber Assembly

We then started the design for mounting the grabber onto our bot. We measure the height of the crown and used that height to insert a cutout to put the axle/pivot point for the Lifting Arm Grabber. Also, we later found out that our Lifter Arm Grabber was very capable of snatching the crown at variations of different heights. Due to Martin's resourcefulness, frequently bringing in items from the random E-waste sites, we were able to acquire a low RPM high torque DC motor, originally from an electric wine bottle opener. We used this motor to lift and lower the Grabber Arm. We were not able to find the datasheet for this rogue motor, so we whipped out our handy-dandy calipers and built a rough 3-D model on SolidWorks.

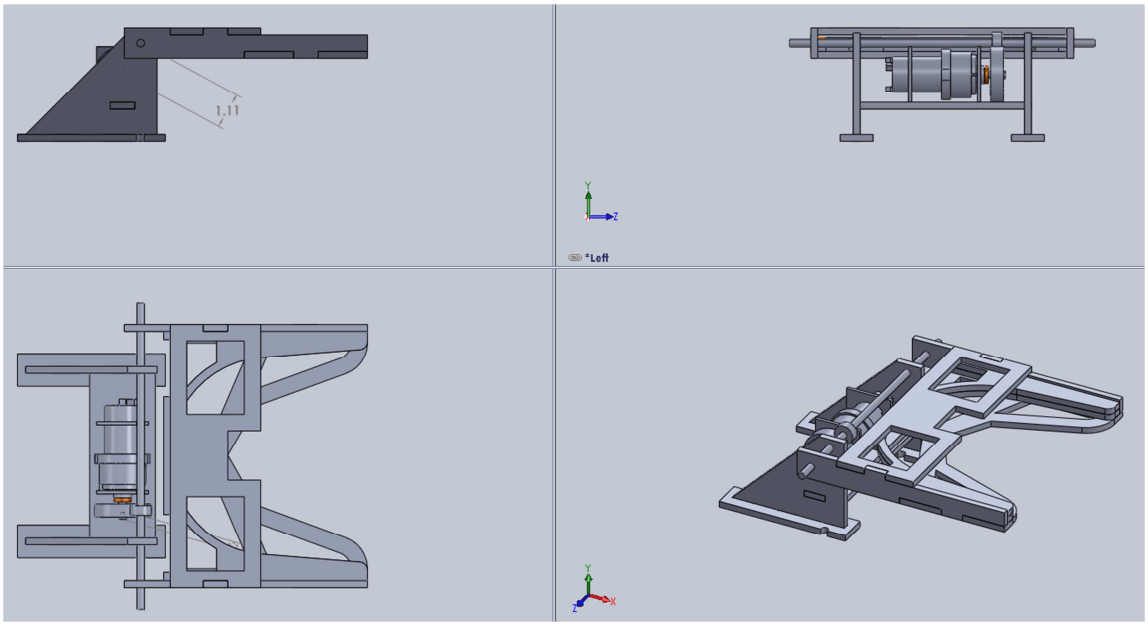


Figure 9: Lifter With Mount Assembly

We were able to find and use gears from old printers, which we attached to the shaft of the motor and the axle rod, a solid piece of metal also found from an old printer. We fastened the gears using super glue and applying heat from the heat gun to cure the glue faster. This method took less than 5 minutes and made the plastic gear and metal rod inseparable. On our actual bot Lifting Arm Grabber we added switches to know when the arm has lowered the correct height and we used a timer to lift the crown, so that we would not drive around the field with our lifter extended outward.

We then had to expand the rear of the Lifter Arm mounting platform so that we had a place to attach our UNO and our large perfboard circuits.

We changed our whisker bump sensors to be three flat bumpers. We did it this way so that we could easily make our bot align perpendicular to the wall and re align itself to make more accurate 90 degree and 180 degree turns.

Results:

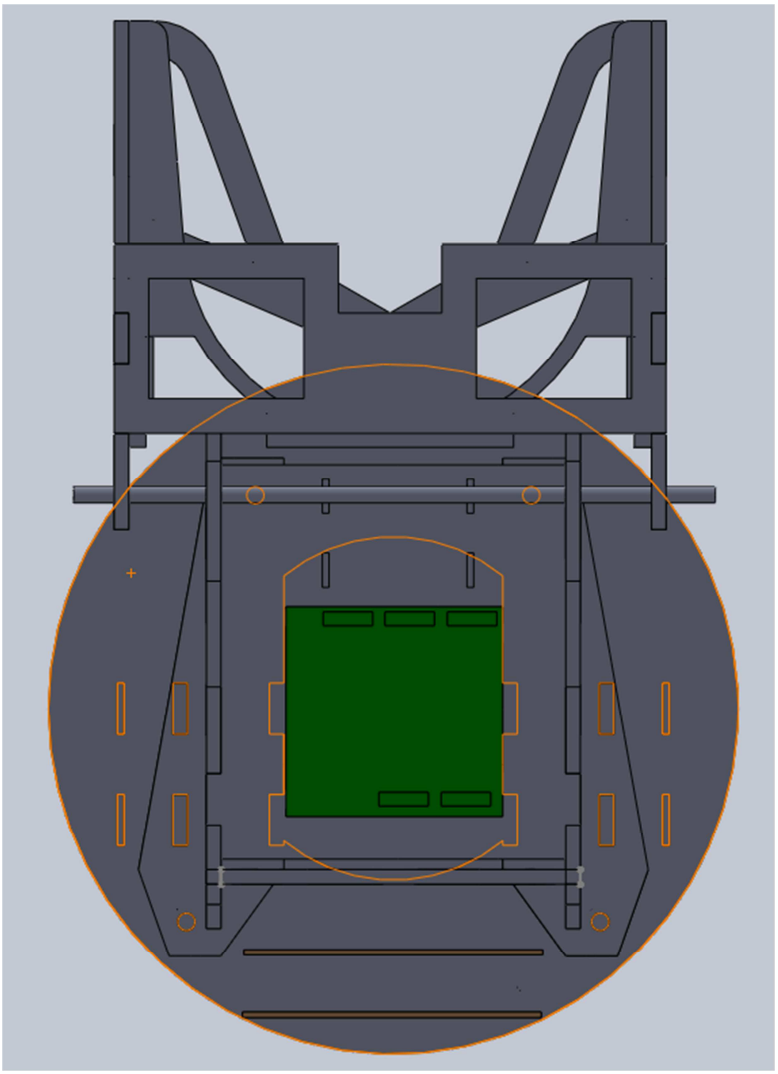


Figure 10: Top View of Final Bot Design

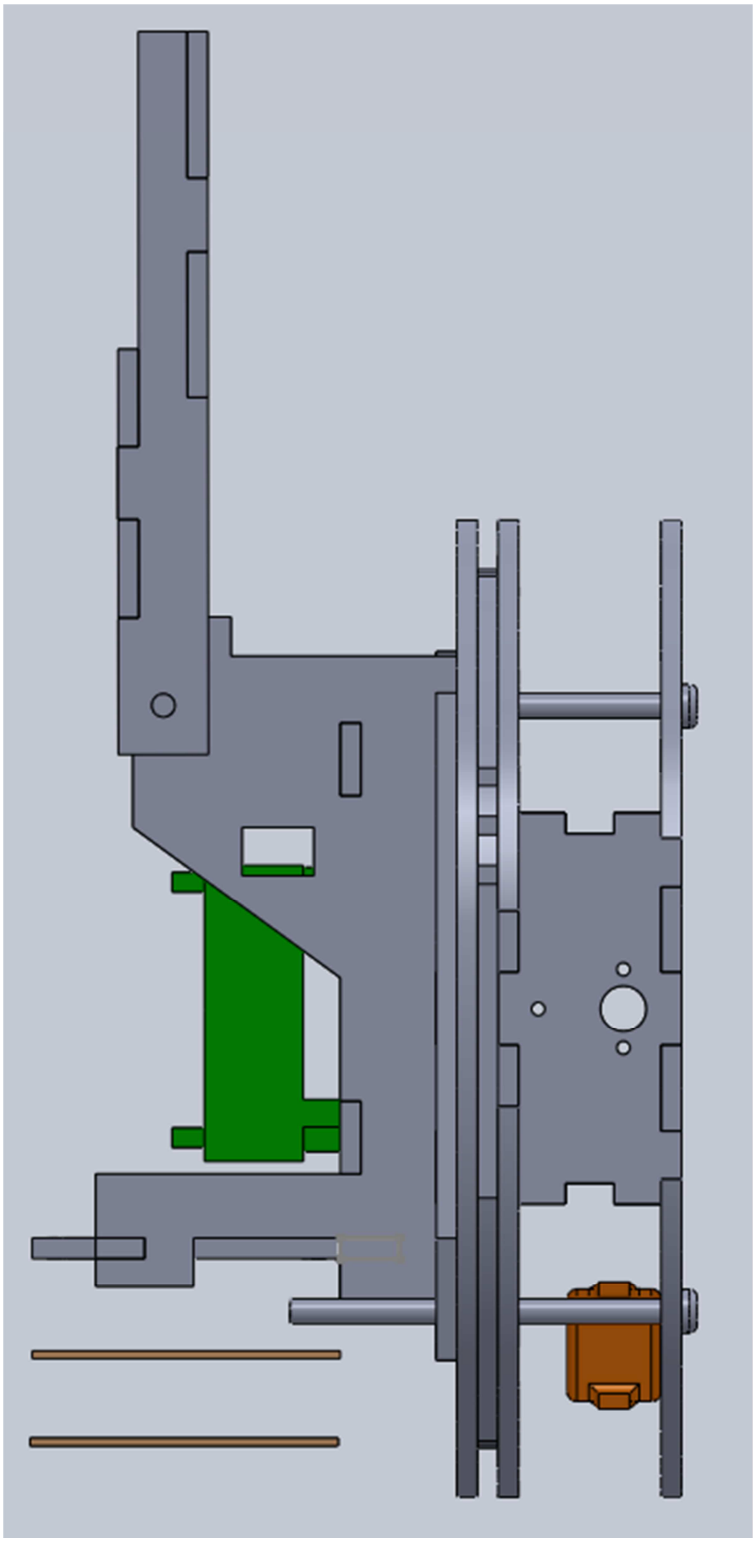


Figure 11: Side View of Final Bot Design

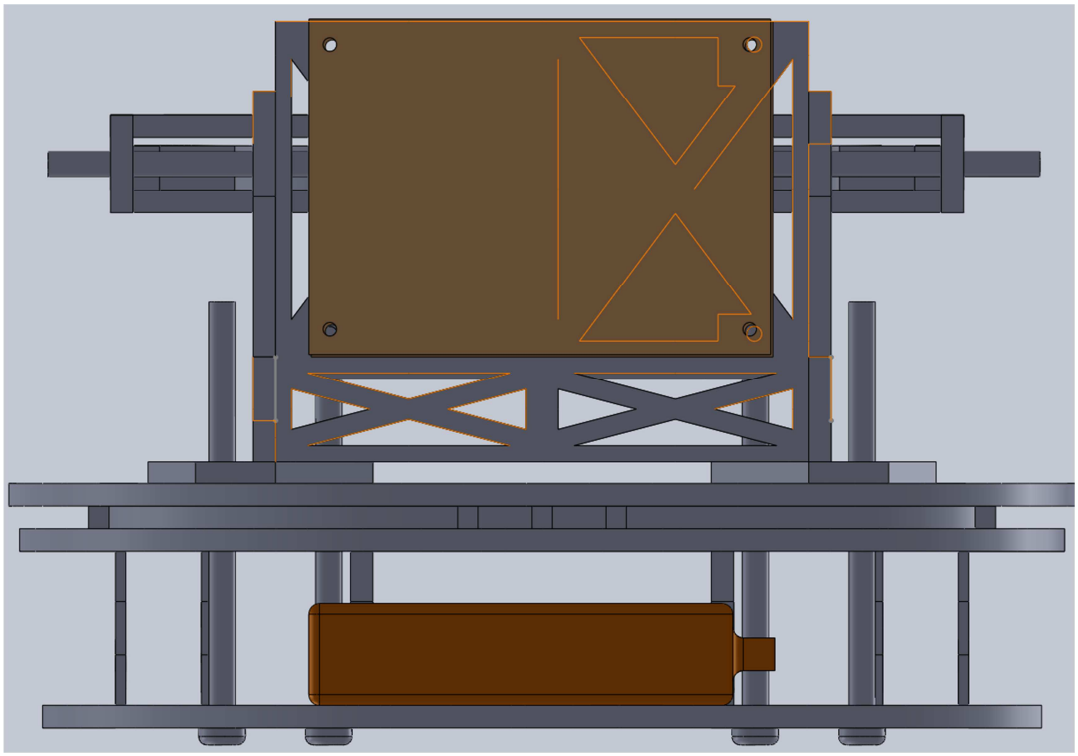


Figure 12: Rear View Of Final Bot Design

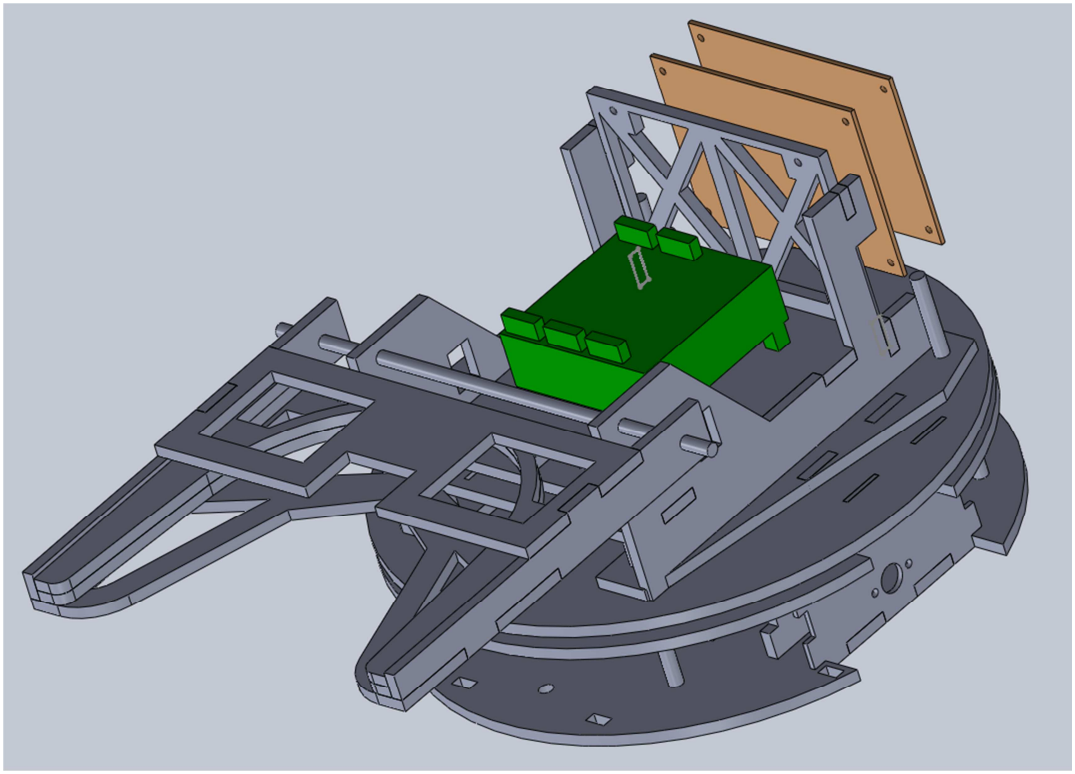


Figure 13: Trimetric View of Final Bot Design

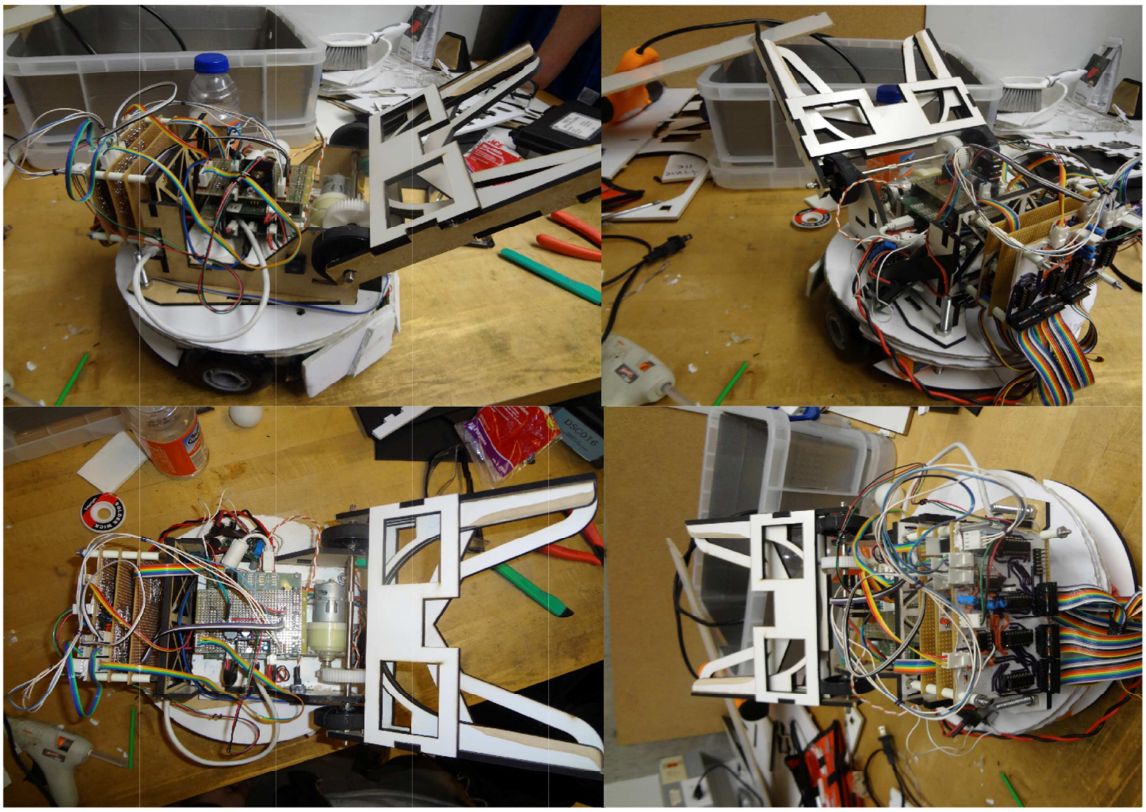


Figure 14: Photographs of Final Bot

Software

Methods:

The software aspect of this project consists of two parts. The first part is building the support libraries and infrastructure needed to control the robot. The second part is designing the actual state machine that it will run. For support libraries, two major libraries are needed for our bot: a motor driving library and an event checker. The motor driving library itself does not need to have complex functionality because all functions are just there to interface with the motors on the bot. All functions operate on a low level of control and so this module consists mainly of operations such as driving straight or tank turning. Its purpose is to abstract away the control of the PWM module and individual ports for interfacing with the H-bridges.

The event checker needs to be able to poll the sensor board described earlier in this report. To do this, it must be able to handle multiplexed sensors as well as mostly analogue levels when interpreting the data. This means operating like a state machine since it must switch the selector pins of the mux, wait for the analogue signal to settle, and then read it and repeat the process. All of this is wrapped up within the event checker functionality so that the state machine only needs to interpret events rather than worry about polling sensors. The events posted by this checker are also simplified so that the state machine level of software does not have to work with analogue readings. The tape sensors use synchronous filtering and the beacon detector is an analogue level, so the event checker handles all processing for these sensors. For the tape sensors, it handles the calculations needed for synchronous filtering as well controlling the tape sensor LEDs, and for the beacon detector, it handles hysteresis. The track wire is a digital signal, but the event checker performs a best of three check to reduce the number of events fired when near the track wire.

Most of the work on the state machine is the actual design process, and so this is laid out in detail in the results section. The state machine is responsible for handling events and controlling the motor driver library.

The final part of software is a large configuration header file which contains all the #define constants for different bot behaviors. These include tuned timer values for turns as well as hysteresis thresholds for sensors. All of this is placed in one location for quick adjustments to the bot during testing.

Results:

The motor driver library implements all the lower level functionality of controlling the bot, as mentioned before. To do this, it runs the PWM module for each of the drive motors. It handles the initialization of this module and is the only library that uses it. For basic driving, it does not handle any timers and this is left up to the state machine level to determine. It is just an interface for working with the motor pins and setting different speeds. The module also adjusts

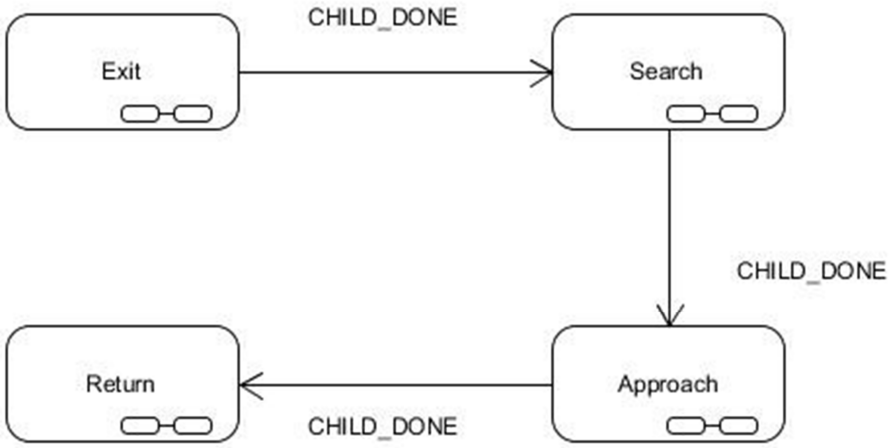
for varying battery voltages. To do this, it reads the battery AD pin and converts it into a multiplying factor that it applies to all 0-1000 speed settings passed to it. The factor is calculated as

$$factor = \frac{V_{max}}{V_{bat}},\ V_{bat} = AD\_Read \cdot \frac{33}{1023}$$

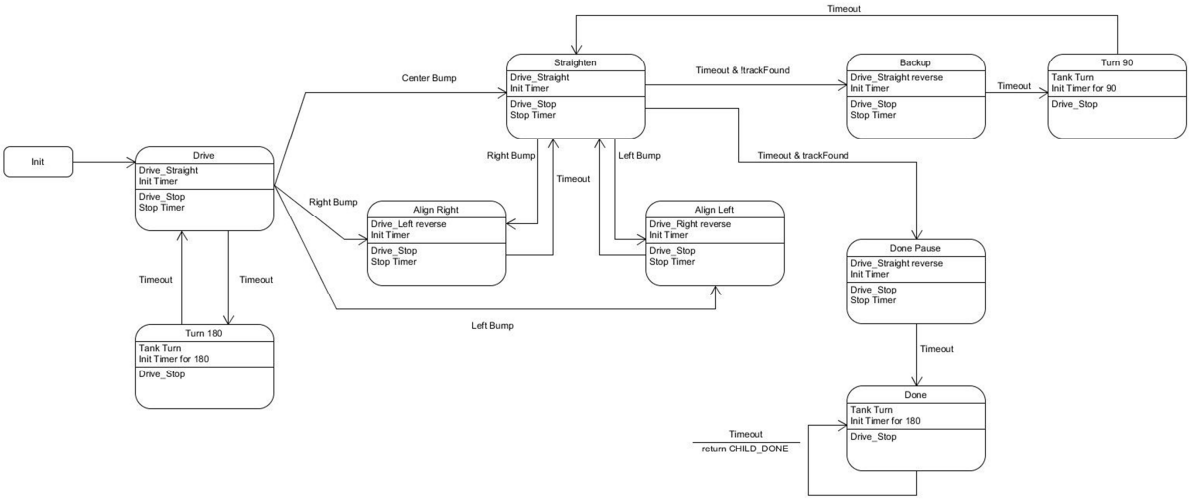
This method does not allow for a full range of duty cycles since it increases the effective voltage on the motors in response to a voltage sag, but our motors are high RPM and generally operating between 20% and 40% which gives us plenty of overhead for this. The advantage of this is that we are never reducing our effective voltage, which would cause problems for us as we drop to low duty cycles.

The event checker is implemented as a service for our robot since it must operate as a simple state machine. The actual states are simple, the event checker is called every 6ms and each time it is called, it polls each mux. This means it polls one bump sensor, one beacon detector, and one tape sensor each time. As it goes through, it also checks the track wire sensor periodically in order to build up a best of three check for events. It maintains an array of all the bumper states and compares using the selector pin settings on the mux as an index. Whenever a bump sensor change is detected, it will immediately post an event. For the beacon detectors, it also maintains an array holding old readings for each of the sensors. These are compared and overwritten each time, and hysteresis is implemented here. The event checker posts an event for a beacon detector if it detects a valid signal transition. For the track wire sensor, as the event checker moves through its 3-bit selector value, the service records 3 track wire readings. The new track wire state is then the best two of these three values, and if this new state differs from the old state, an event is posted. Since the selector must increment fully to finish a best of three, the track wire sensor is polled every 48ms effectively. The tape sensors are handled synchronously so the service records the analogue reading into an array using the selector as an index just like the beacon detectors, but this is repeated twice before an event can be posted. The tape sensor LEDs begin turned off and each sensor is read. Once the selector increments and rolls over to zero, the LEDs are switched on and a second array is populated. After this second iteration, the differences between the on and off states are measured, and one single event is posted for the tape sensors. The event parameter contains two bytes to allow the state machine to determine what kind of event occurred. One byte is a bitfield where a 1 indicates that the tape sensor changed states since the last measurement. The other byte is a bitfield that indicates the current binary state of each tape sensor, regardless of whether it changed this reading or not. By using these two, the state machine can determine if the event means that the sensor has left tape or entered tape.

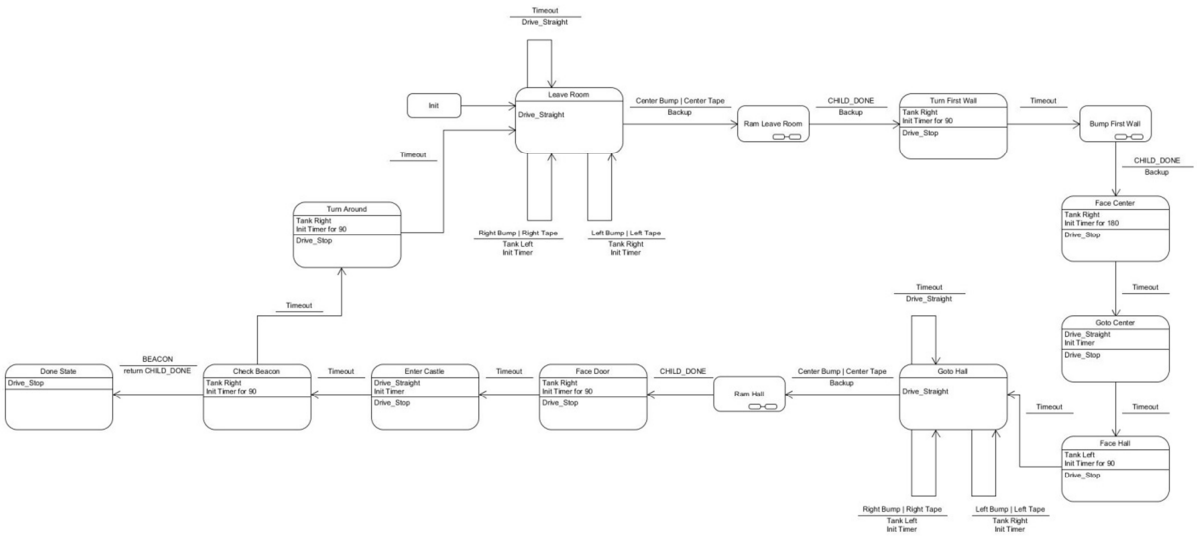
For the state machine implementation, we chose this top level method.



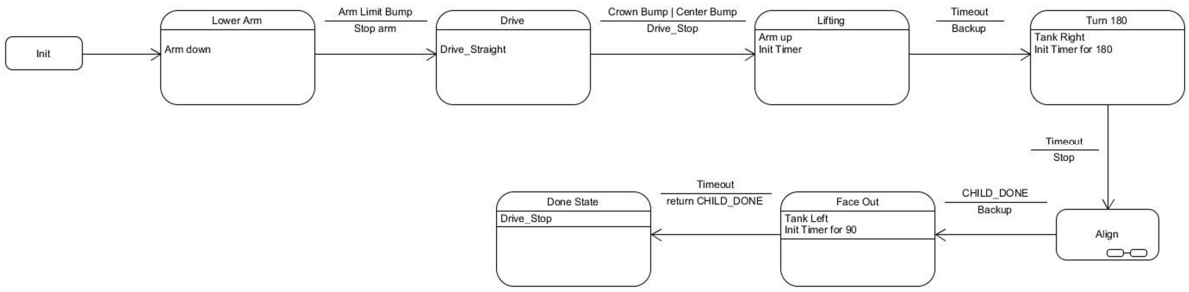
We broke it down into four major actions that the robot must accomplish. It needs to successfully exit its own castle, locate the ruling castle, pick up the crown, and finally find its way back home. To exit the castle, we implemented the following state machine.



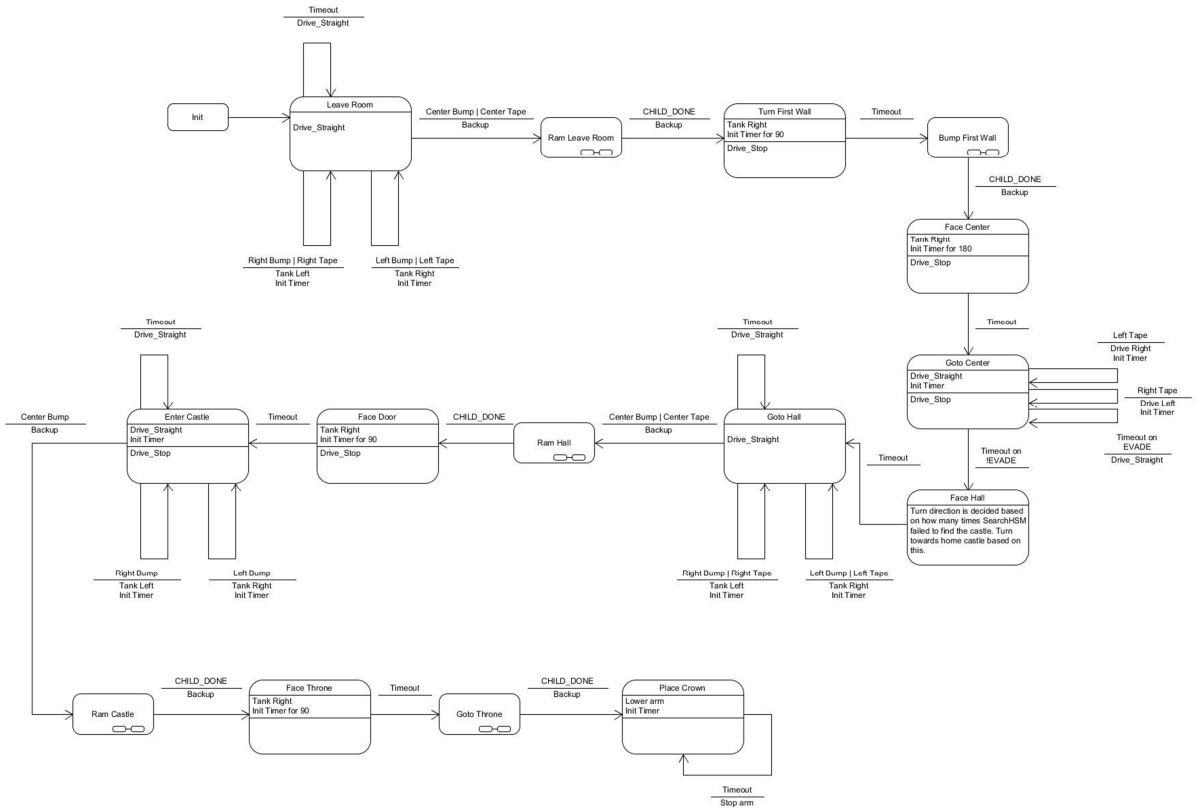
On a high level, the robot will find a wall and then align itself perpendicular to it. Next, it will reverse and turn 90 degrees clockwise and repeat the process. During the alignment time, it also checks for a track wire signal. If it detects the track wire, it knows that it is at the back wall of the castle and can stop searching. It then turns around 180 degrees to face out of its door and prepare for the next state, Search.



This state loops to navigate around the field and check each castle in a counterclockwise order. The sub-HSMs shown here and in following diagrams are all the same Ram Sub HSM which aligns the robot perpendicular to the wall it runs in to. One iteration of this state machine moves the robot into the castle immediately counter-clockwise to where it started and it will end facing out of that castle's door, just as it began in its home castle. It does this by finding the "hallways" on the field. Once it knows it is in a hallway, it knows how long it takes to drive to the center. From there it turns and drives down to the end of the next hallway and aligns with the back wall. Using this it enters the castle and does a 180 degree sweep looking for the beacon signal. If it detects it, it stops turning immediately and moves to the next state, Approach.

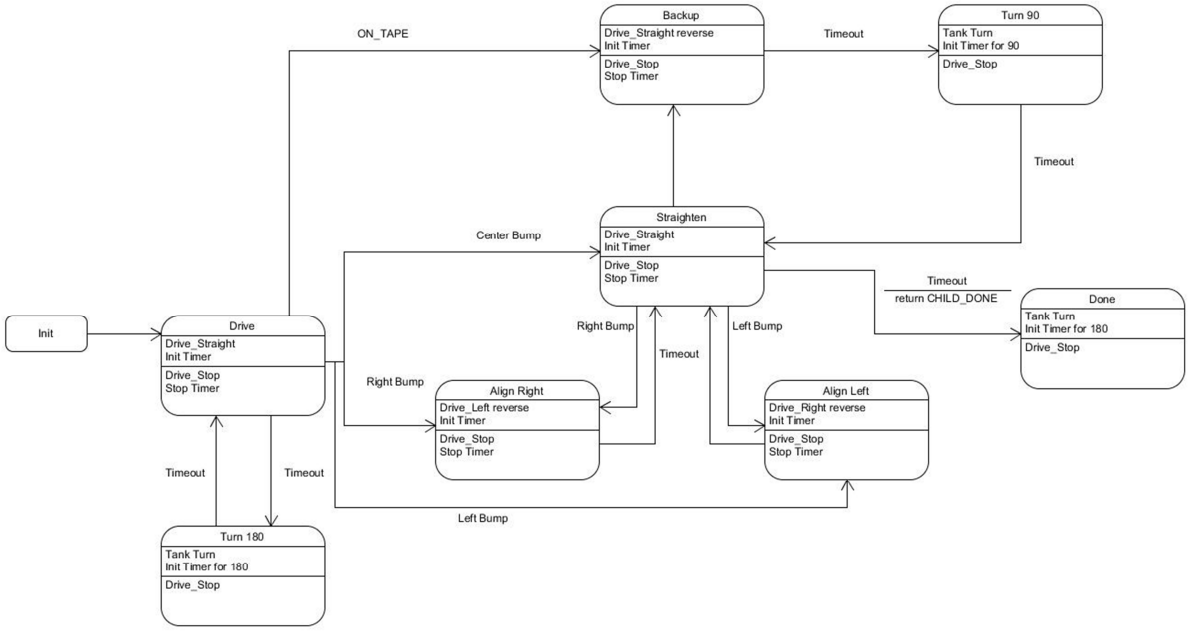


This state machine is simple. It presupposes that the robot is facing directly at the ruling throne. This state just handles lowering the arm, driving forward into the throne, lifting the crown, and then aligning to face outside the castle door as before. This allows it to transition immediately into the Return state, which closely mirrors the Search state.



This state follows the exact same template as the Search state with one small change. During the Search state, each time the state machine loops, a global variable is incremented. Return uses this global variable to decide where it is relative to its home castle. When it gets back to the center of the field, it will choose which hallway to face and then continue as usual for the Search state. Once it is facing into its home castle, instead of entering and looking for a beacon, it aligns itself against the back wall. Next it faces the throne and approaches it to return the crown.

The final state to talk about is the Ram sub-state used in all of these.



This state is based on the Exit state and so closely resembles it. The major difference is that it responds to tape. Whenever one of the tape sensors is triggered while driving forwards, the robot will stop, back up, and turn away from it. Originally, we had tried having it align along tape that it finds but it turns out to be too unreliable. It’s better to simply bounce off of the tape and align against a wall nearby instead.

Circuits

Introduction:

The electrical system for our robot consisted of the battery, Uno stack, three DC motors and two H-bridge boards to drive them, and the sensor stack. The electrical system block diagram below provides details about the interconnections between the various sections of the system.

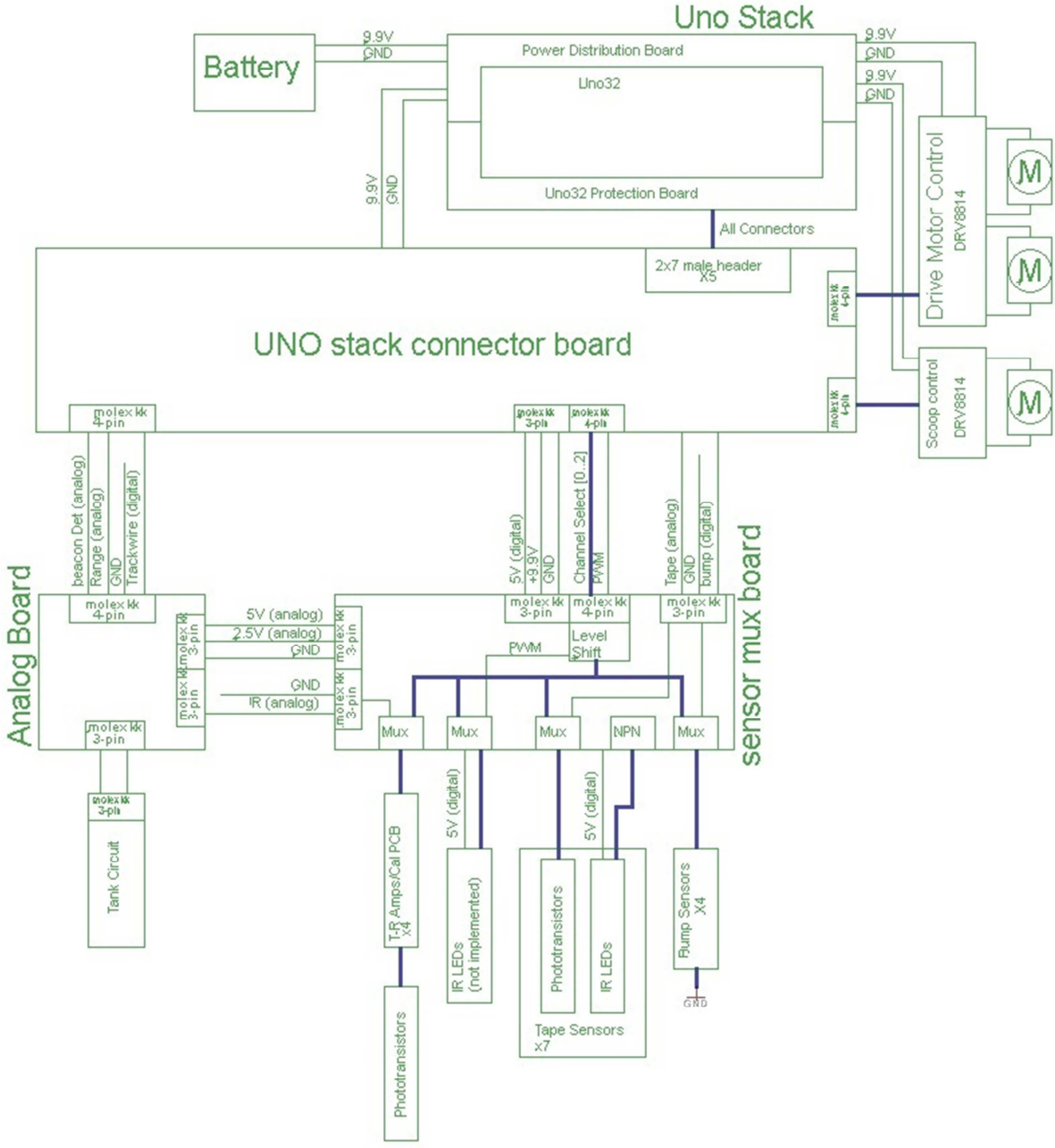


Figure 1: Bot Electrical System Block Diagram

Methodology:

Power

Power for the 'bot was provided via the 9.9V LiFo battery pack provided. This battery was plugged into the power distribution board of the Uno32 Stack. The power distribution board provided power for all of the stock boards as well as the Stack Connector board.
The majority of the circuitry ran off of a 5V supply voltage. Since there were both digital and analog circuits involved, the 5v supply was implemented as independent analog and digital rails to provide noise isolation.

Uno32 Stack Connector Board

This board was created to plug into the Uno32 stack via the provided headers on stack. It provided an interface between Uno32 stack and the rest of the system. It also provided the main 5V digital supply rail for the remainder of the system. Due to the multiplexed nature of the bot's sensor array, the number of I/O pins used was minimal despite the large number of sensors the system was capable of supporting

Interconnects

In order to provide for ease of disassembly during design and provide a degree of modularity, all board to board wiring was connectorized using either 0.100" pitch Molex KK series connectors or 14-pin IDC ribbon cable interconnects.

Actuators

Three DC motors were used in the robot. Three identical DC gear motors were purchased from MPJA. Two were used to provide propulsion and steering and the third one was kept as backup in case one of the other two were to fail.
The third motor assembly used in the bot was used to power the crown lifter arm. This motor was a low speed high torque model. A planetary gearbox was liberated from an electric corkscrew found in the e-waste. The original motor was faulty, but turned out to be a standard size. A DC motor from a printer was used as a replacement.
Since this motor was not on the approved list stall torque needed to be measured to ensure that the H-bridge boards provided could handle them. This necessitated the removal of the DC motor from the gearbox as the output shaft of the gearbox could not be stopped by hand.

Sensors

In addition to the sensor circuits developed in previous labs, a variety of other systems were explored in order to aide navigation.

PS/2 Mouse

Wheel Tachometers

In order to track the bot's trajectory and thus aid in implementing the shortest path home after a successful crown retrieval, both the use of reflective tachometers and a PS/2 ball mouse were

explored. Due to difficulties presented by the PS/2 communication protocol development on the mouse driver was dropped after several days of development.

Trackwire

The trackwire sensor was the last subsystem to be completely implemented. It was used to detect when the bot was within two inches of an active trackwire and worked as expected after gain was properly adjusted.

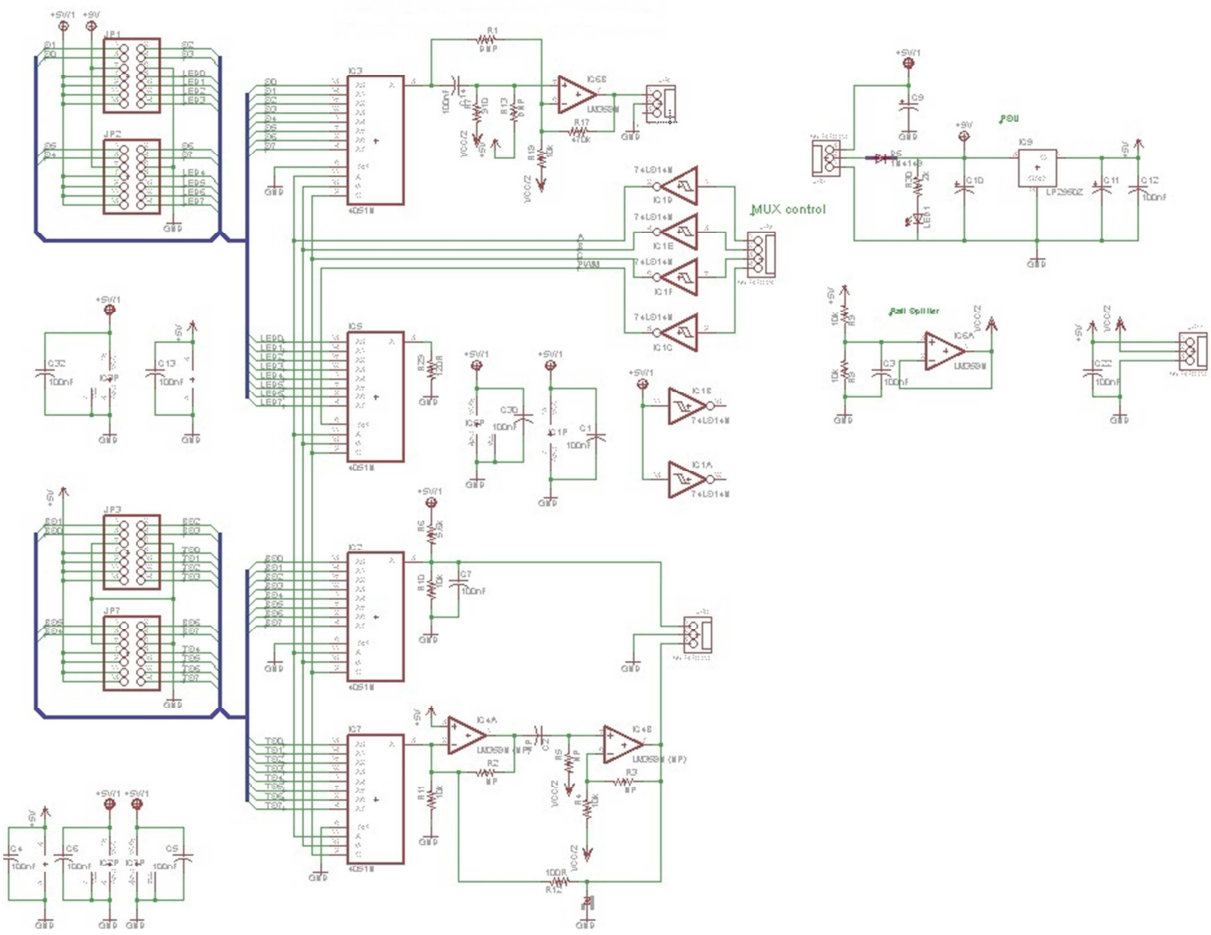


Figure 2: Sensor Mux Board Schematic

Multiplexed Sensors

The remainder of the sensors were read and driven using four 8-channel analog multiplexers. This was done to conserve I/O lines as well as to prevent the necessity of soldering multiple identical circuits. It may have also limited the errors due to component tolerances as all sensor outputs were routed through the same processing circuitry and read by the same ADC pin. All mux selector pins were tied together. In this way all sensor channels were changed in parallel.

Beacon Detectors

The beacon detector design from Lab 1 was modified by placing an analog mux at the input of the active filter stage. The new circuit thus had the capability of reading up to eight different phototransistors while only implementing one filtering circuit. This would allow the state machine to determine where the beacon signal was the strongest. This would greatly speed up navigation as the bot would only need to reach the center of the field and then go in the direction of the strongest beacon signal.

The trans-resistive stages were kept on the input stage of the mux as a means of sensor calibration. A 10k trim-pot and 4.7k fixed resistor replaced the single 10k fixed resistor originally used.

Output of the circuit was an analog voltage read by the ADC. This enabled thresholds to be set in firmware for additional design flexibility.

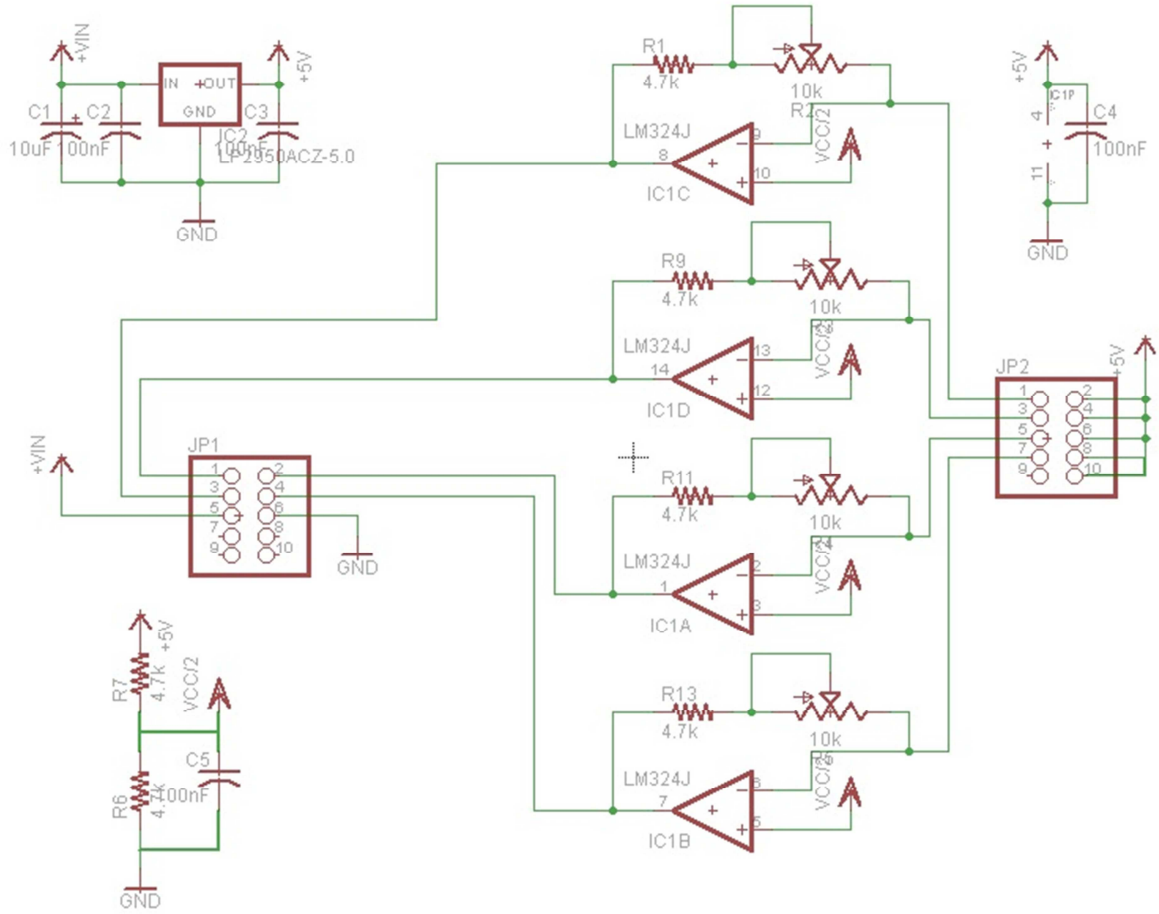


Figure 3: Beacon Detector Trans-Resistive Stages

Distance Sensors

A simple reflective distance sensor topology was also implemented by adding two IR emitters on either side of each beacon detector phototransistor. Since switching the LEDs at the same frequency as the beacon would have interfered with our beacon detectors as well as jammed

the other bots, a different switching frequency was needed to drive them. 13 kHz was chosen both because it was roughly half way between the operating frequencies of the beacon and the trackwire and because 13 is a prime number. This should have minimized any harmonics coming from other parts of the circuit and simplified filtering.

A second filter circuit was connected to the output of the beacon detector mux. Analog output of the distance sensor system was inversely proportional to four times the distance from the object squared when the phototransistor was perpendicular to the reflecting object.

Proof of concept for the design was obtained by adding an IR LED to an existing beacon detector board and driving it at 2kHz. During testing, this circuit detected a mirror from up to 12 inches away when the phototransistor was properly shielded. It could detect the edge of an MDF ruler from about four inches.

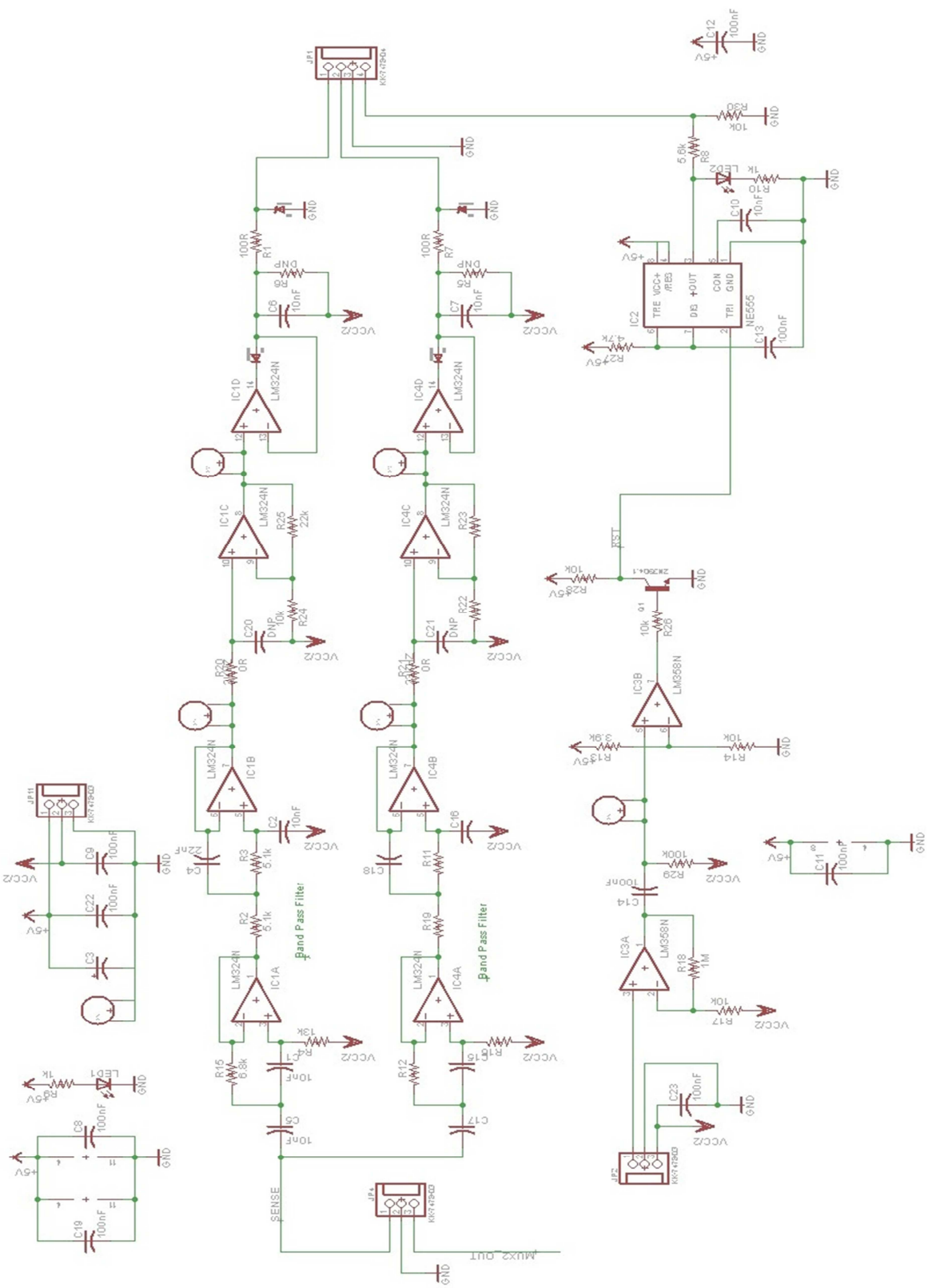


Figure 4: Beacon Detector and Distance Sensor Filters and trackwire

Bump Sensors/Limit Switches

Three bump sensors were implemented on the front of the bot to detect collisions and to aid in wall alignment. Another limit switch was placed within the arm to detect when the crown had been grabbed. The final limit switch was placed on the base of the lifter arm to detect when the arm was completely lowered. This was necessary as the motor used to lift and lower the arm probably had enough torque to damage the bot if it was left on too long.

All switches were connected in normally open mode one pole of each switch and the other was connected to a switch mux input. The output of the mux was connected to the output of a 5V to 3.3V voltage divider. In this way if the selected switch was open the Uno saw a high on its input pin. If the switch was closed, the voltage divider's output was grounded through the mux and switch and the Uno sees a zero.

Tape Sensors

Tape sensors were read in analog mode using active filtering as such all tape sensor LEDs were connected to individual resistor and then to a common NPN transistor under software control. Phototransistor emitters were connected to individual mux channels. The tape sensor mux's output was connected across a 10k resistor and then read by the Uno's ADC.

A resistor and 3.3V zener diode protected the Uno from any unexpected voltage spikes.

General PCB/Perfboard Design Methodology

The designs for majority of the perfboards used on the bot were first laid out using Eagle and then manually routed in such a way as to accommodate the limitations of using perfboard. Once a satisfactory PCB layout was achieve, the top and bottom copper layers were printed out using 1:1 scale and used as routing templates for the actual perfboards for the top layer, holes were punched in the paper over the drill holes for the IC packages. IC sockets were then placed through the holes and tacked down using solder. This ensured proper alignment and also secured the paper to the PCB. The rest of the top side of the PCB was then populated using the paper template as a guide for both component placement and wire routing.

The solder side of the perfboard was handled differently, as securing the paper to the PCB would have been both difficult and probably counterproductive. Instead the paper was used as a template to form the bare wire on before it was soldered to the board.

This approach provided both a record of the circuit design should troubleshooting be necessary, and sped up the assembly process.

Results/Conclusion:

The stack board may not have been a good use of design time as there was minimal usage of Uno I/O pins. It would have probably been a better choice to use the IDC's provided and then crimp the KK's to the end, as there was no cross pollination between Uno stack ports used.

The flexibility and ease of disassembly provided by using removable connections was definitely helpful during both assembly/disassembly and hardware debugging

The serial mouse tachometer was abandoned early on in the project due to software timing issues.

Wheel tachometers were abandoned due to implementation and mechanical issues.
The multiplexed sensors were sort of indeterminate. On the one hand the system worked as expected. On the other hand given the percentage of capability utilized, a less complicated system would have worked as well and taken less time to implement.
IR distance sensors were eventually abandoned after it became apparent that foamcore does not insulate IR light very effectively. In the foamcore prototype the IR signal passed through the walls dwarfed the signal amplitude reflected from a piece of MDF. Fixing this issue would have necessitated re-cutting of parts of the chassis out of MDF. This was deemed to be too time consuming and as a result these sensors were dropped from the final robot design. Given the performance of the other tape sensors gave during the competition, this was probably just as well.
The final design utilized seven tape sensors, five limit switch/bump sensors and only one of the four beacon detectors implemented.

Conclusion:

This project has been one of the most challenging projects each of us have ever had to do. We each underestimated how difficult working on a team was and how little time we had to built a souped-up Rumba. We all had our own ways we wanted to implement things in the robot, and for the beginning phase of building the robot we were all doing our own thing. We realized we did not communicate with each other very well, which could make any project unsuccessful. We sucked it up and spent days trying to make up for our awful communication skills. In the end we gained more knowledge than we have had in any other of our classes in the University California of Santa Cruz. In the end, we built a awesome robot that has the ability to do what we built it to do a good percentage of time and we had so much fun struggling and suffering with all the other geniuses in CMPE 118 Mechatronics.