Roborodentia Competition Report

Stephen Schmidt

June 15, 2018

Contents

1 Introduction

For my Senior Project, I designed and built a robot for the 2018 Cal Poly Roborodentia competition. The goal was to gain exposure to more CPE and EE skills and integrations with software. For this project and competition, I worked by myself with the help and advice of Dr. Seng. I spent just over 12 weeks planning, designing, sourcing parts for, building, programming, and testing my robot. Due to the nature of the competition, the robot needed to be fully autonomous and be able to navigate the course as well as accomplish the main objective which involved both collecting foam balls and firing them across the course into opponents targets.

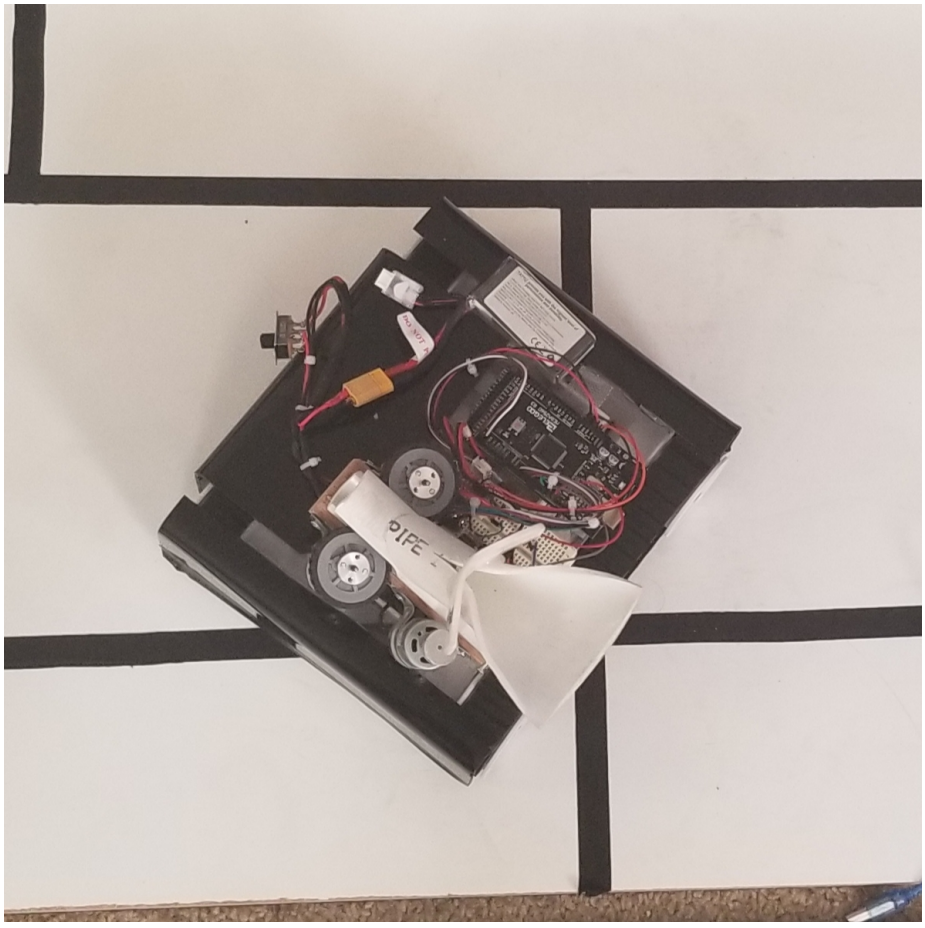

Figure 1: TreadBot

2 Problem Statement

The complete guidelines and rules to the 2018 Roborodentia competition can be found online on the Cal Poly Roborodentia Current Contest rules. Assuming that the reader is familiar with the rules and objectives of the competition, I will begin by outlining the problems I attempted to solve with my design.

2.1 Projectile Launching

Given that the main objective was to launch balls into an opponents target, I set out to design a robot with a consistent and accurate and simple means of launching projectiles. The solution had to be one that was adjustable, and remarkably consistent and resilient to changes or variances in projectiles.

2.2 Projectile Acquisition, Storage, and Feeding

The secondary (and just as important problem) was the problem of acquiring, storing, and feeding the projectiles to the launcher. The contest rules stated that there would be gravity fed supply tubes at certain positions on the course, and outlined the specifics of the dimensions and other relevant restrictions.

2.3 Navigation and Movement

The third main problem to solve in the competition was the movement and navigation of the robot. The course itself had black line demarcations on it to facilitate the use of line tracking modules. So, to be successful, my design had to solve all three of these problems while conforming to the rules and physical constrains of the competition and the course itself.

2.4 Miscellaneous Problems

Aside from the main objectives and problems my robot attempted to solve, various other smaller problems became apparent throughout the course of development. Some of these problems included handling jams, inconsistencies in motor functionality in my chassis, handling the variances in the course itself, and avoiding and handling other external factors like projectiles on the course itself that could interfere with my robots movement.

3 Software

3.1 Introduction

The software component of this project was unfortunately the one where I was able to spend the least amount of time, so I had to write a lot of code very quickly. Most of this was due to my issues and pace in overcoming the design and creation of the physical and electro-mechanical systems on the robot. Had

I been working on a team, I feel that I could have written a much more robust and better implemented system. However, I feel that my model still was on the right track and I would have kept this model if I had more time to improve the code itself.

3.2 Platform

My robot was using an Arduino Mega board for its main processing and control unit. This introduced a number of constraints and limitations of possible architectures and designs for my robot's main logic and control system. First, the Arduino platform runs C/C++ code using one setup function and as soon as execution is finished it repeatedly runs one main "loop" function. This model limits design patterns that can be used.

3.3 Basic Architecture

Due to how the Arduino interfaces with sensors and other I/O devices, the main loop calls composite movement function that contain control flow structures that are based on global flags and semaphores that are set by the I/O devices on the robot. Additionally, this means consistent and frequent polling of these sensors is necessary in order to gather and store the information and set the flags appropriately. I abstracted my movement functions into two layers to enable modularity of code and reusable code. Some example code with comments from my design are included in later sections. I broke my code into 5 basic function levels/types described in the following sections.

3.3.1 Basic Movement Functions

These functions were designed to guide the robot in "stages" according to sensor input. An example function would be something like "followLineForwardUntilIntersection(int spd)". This function aims to be descriptive in its name so that when it is used later in a higher level composite function it is clear what the function accomplishes. Inside this function, there are calls to poll sensor data, as well as library function calls to the my motor driver. These functions are the core building blocks of the composite functions and form the basic "atomic" movements that the robot makes. (See section 3.4.1)

3.3.2 Composite Movement Functions

These functions form more lengthy and complicated actions of the robot. These functions would "compose" a series of calls to the lower level basic movement functions in order to perform more complicated actions. (See section 3.4.2)

3.3.3 I/O Functions

These functions were for composite actions like firing the launcher or engaging the vibrational anti-jamming motor. I also consider calls to the Arduinos digital

or analog I/O pins I/O functions as well. (See section 3.4.3)

3.3.4 Computational Functions

These functions were responsible for analyzing the sensor data returned by calls to I/O functions. These functions were used to set flags and semaphores based on analysis of sensor input so that the basic movement functions would have the data needed in order to operate correctly. Lots of tweaking of values and algorithmic experimentation was done to try to achieve reliable and accurate results based on the input of the sensors. (See section 3.4.4)

3.3.5 Main Loop

This function is a built in function that the Arduino itself runs repeatedly while it is powered on. This function was the sort of "main" of my program where I call my composite movement functions in the correct order to get the robot to navigate the course and perform actions like reloading, firing, etc. (See section 3.4.5)

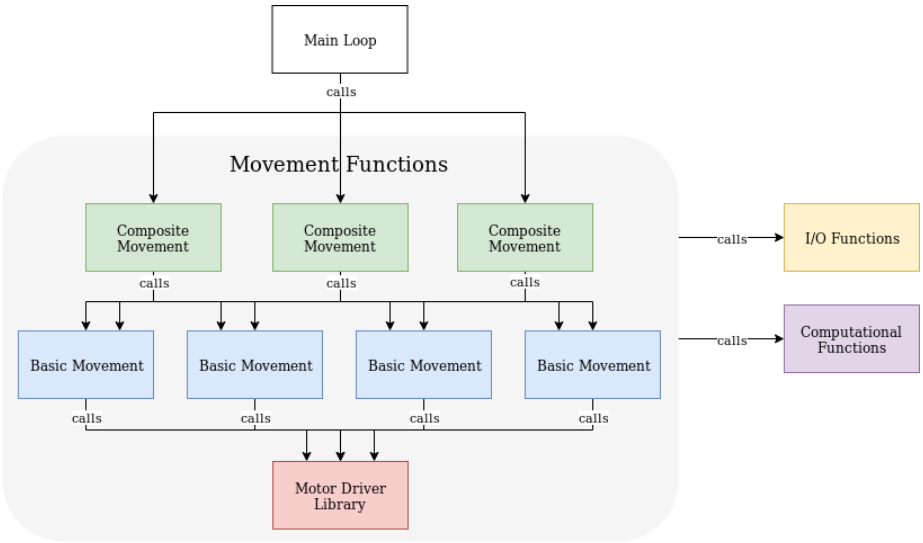

Figure 2: Architectural Block Diagram

3.4 Code Functionality and Examples

Functionality and example code from my robot are included below and the functionality is discussed in the code comments.

3.4.1 Example 1: Basic Movement Function

```
//takes a value speed between 0-255;
void followLineForwardUntilIntersection(int spd) {
  direct = FWD; //sets the flag of the current direction of the robot
  while(!atIntersection) { //loop while we haven't encountered an intersection
    pollSensors(); //gather data from IR sensors for line tracking
    checkTurns(); // analyze IR data for possible turns and intersections
    //LM_ADJUST account for the left motor in my robot being slightly slower
    // correction is determined by the analysis of the IR sensors to guide the robot
    motor_right.drive(spd -correction);
    motor_left.drive(spd+LM_ADJUST + correction);
    delay(50); //necessary to keep polling of sensors to a reasonable rate
  }
  brake(motor_left, motor_right); //stop both motors
}
```

3.4.2 Example 2: Composite Movement Function

```
//this function is responsible for forward navigation across the course and firing
//the launcher into each of the enemies targets for "full house" bonus and an
//additional final shot into the enemies high value target, stopping at an intersection
void traverseForward() {
  delay(2000); //initial delay for competition start
  digitalWrite(VIBE, HIGH); //initiate the vibration on the hopper system to mitigate jams
  direct = FWD; //set the robot direction
  uchar fireCount = 0; //keeping track of the fire count
  followLineForwardFor(1000, 60); //basic movement function call: (millis, speed)
  fire(); //fire first shot
  //the FIRE_INTERVAL was trial and error based to fire in each of the opponents targets
  followLineForwardFor(FIRE_INTERVAL, 60);
  fire();
  followLineForwardFor(FIRE_INTERVAL, 60);
  fire();
  followLineForwardFor(FIRE_INTERVAL, 60);
  fire(); //second to last shot
  delay(300);
  fire(); //5th (last) shot, fire twice, highest value target
  digitalWrite(VIBE, LOW); //disable the vibration to save battery
  followLineForwardUntilIntersection(40); //basic movement function call
}
```

3.4.3 Example 3: I/O Function

```
//function to fire the launcher
void fire() {
  digitalWrite(SOLENOID, HIGH); //engage the solenoid
```

```
  delay(500); //wait
  digitalWrite(SOLENOID, LOW); //disable solenoid, launching the projectile
}
```

3.4.4 Example 4: Computational Function

```
//this function is responsible for analyzing the array of data from the IR
//sensor to determine if any turns are available. The direction the robot is
void checkTurns() {
  if(direct = FWD) { //check the robots current direction of travel
    //use the values from the arrays of IR diodes to detect a right hand turn
    canTurnLeft = ((data[0] + data[2] + data[4] + data[6]) < 15);
    canTurnRight = ((data[8] + data[10] + data[12] + data[14]) < 15);
  }else if(direct = BCKWD) {
    canTurnRight = ((data[0] + data[2] + data[4] + data[6]) < 15);
    canTurnLeft = ((data[8] + data[10] + data[12] + data[14]) < 15);
  }
  //if both a right and left turn are possible, an intersection flag is set
  atIntersection = canTurnLeft && canTurnRight;
  if(atIntersection) Serial.println("atIntersection"); //debugging log output
}
```

3.4.5 Example 5: Main Loop

```
//main loop of the robot, this calls all of the composite movement functions
//in the proper order and repeats indefinitely
void loop() {
  pollSensors(); //init sensor data
  checkTurns(); //check turns

  traverseForward(); //first (preloaded) traversal and firing
  loadFarSideRight(); //reload far right side

  traverseBackward();
  loadNearLeftSide();

  traverseForward();
  loadFarSideLeft();

  traverseBackward();
  loadNearSideRight();

  brake(motor_left, motor_right);
}
```

3.5 Function Prototypes And Overview

As mentioned earlier, I set out to write long and descriptive function names so that I could compose them into groups of more complicated functions. As such, I feel that given the earlier examples and similarity between many of these functions, no further elaboration is necessary on what they accomplish or how they work. Therefore I have simply included the full list of prototypes from my program. Note that the library functions for my motor controller are not included in this list, as I did not write them despite using them.

```
//pin mappings etc
void setup();
//I/O functions
void fire();
void pollSensors();
//Computational Functions
void wait();
void checkTurns();
//Basic Movement Functions
void followLineForwardFor(int duration, int spd);
void followLineForwardUntilIntersection(int spd);
void followLineForwardUntilLHT(int spd);
void followLineForwardUntilRHT(int spd);
void followLineBackwardFor(int duration, int spd);
void followLineBackwardUntilIntersection(int spd);
void followLineBackwardUntilRHT(int spd);
void followLineBackwardUntilLHT(int spd);
//Composite Movement Functions
void traverseForward();
void loadFarSideRight();
void loadFarSideLeft();
void traverseBackward();
void loadNearLeftSide();
void loadNearSideRight();
//Main Loop
void loop();
```

4 Hardware

4.1 Introduction

My robots electronic hardware included an Arduino Mega, a Sparkfun TB6612FNG H-Bridge dual motor driver board, a Sunfounder 8 diode IR line tracking module, and some smaller transistor circuits I designed to drive higher current drawing components such as the solenoid and the vibrational motor from the Arduino as well as two motors for the launcher and two motors to drive the robots chassis.

4.2 Block Diagram and Architecture

Below is a diagram of how my components were connected. I/O is labeled for the two main logic boards, the Arduino and the TB6612FNG H-Bridge. Power connections are not shown here for simplicity's sake. Detailed schematics can be found in section 4.4

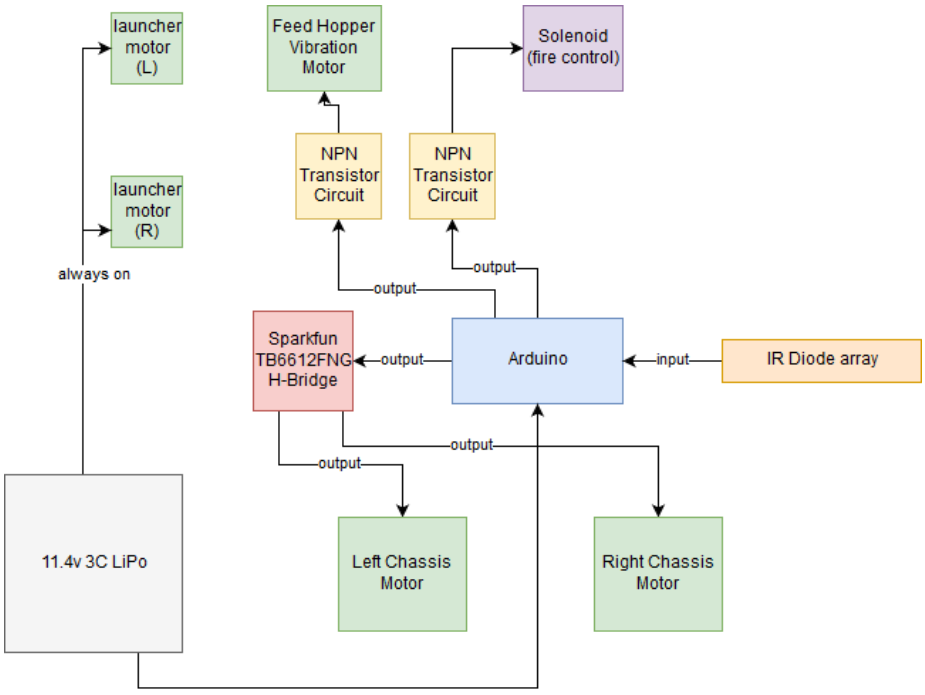

Figure 3: Hardware Block Diagram

4.3 Overview and Component Interaction

The two main logic boards powering my robot were the Arduino Mega itself and a Sparkfun TB6616FNG H-Bridge motor driver to power and control the robots main two motors. Additional electrical hardware included two motors that comprised the launcher, a solenoid that was the feed/fire control, and vibration motor to prevent jams, and of course the IR array for the line tracking component. Additionally, I built two small NPN Darlignton Transistor circuits used to switch higher current than the Arduino could handle to both the vibrational motor and the fire control solenoid. The entire robot was powered by one 11.4V 3C LiPo battery pack.

4.3.1 Arduino

I used an Arduino Mega as the main logic board for my robot. It was the main controller for every hardware component with the exception of the two launcher motors. These motors together in parallel only drew 137mA so I decided for simplicity's sake to hardwire them to the main power so that they were always on. This eliminated the need for extra logic and drivers and simplified my circuits and my code logic. The Arduino controlled the solenoid and the vibration motor via its digital out pins through the Darlignton transistor circuits I made. It interacted with the TB6612FNG H-bridge via its PWM pins, and received input from the IR Diode Array on its digital in. The Arduino was powered via its 12V in pin directly from the LiPo power source. However, at the time of the competition, the 5V regulator fried (my fault), and I needed to fix it by powering the Arduino with its own separate USB battery bank that I hot-glued to the robot.

4.3.2 TB6612FNG H-Bridge Motor Driver

This motor driver received its power from both the battery directly as well as an on signal from the 5V output of the Arduino. It drove the chassis motors and was a core component of my robot. It performed perfectly and the C library that was available through the sparkfun website to control it worked perfectly as well. I had no issues with this component whatsoever.

4.3.3 Darlignton Transistor High Current Switches

I used Darlington TIP 120 NPN Transistors to switch to high current to my high-current drawing components like the fire control solenoid (which pulled over 2 Amps) and the feed vibration motor which was surprisingly current thirsty. These were basic circuits that comprised of a an input from the Arduino via its digital out pins and used that signal to switch a higher current directly from the battery to whichever component it was driving. Additionally, I placed a diode in parallel with each component the driver was powering because the solenoid and the motor both utilize coils who's inductance could "clog" up the transistor when the current was switched off. The diode prevented this from happening.

4.3.4 IR Diode Array/Line Tracking Module

I used a Sunfounder IR line tracking module for my robots navigational input. It uses 8 IR diode and IR receiver diodes on one board and has logic controllers built in. I had to desolder its output connector and flush solder the wires myself to adapt it to fit on my chassis. Once the physical modifications were done I was able to test its input to the Arduino and calibrate it based on the mock course I had built. I dialed in the output of the IR module using the on-board potentiometer and connected it to a digital input on my Arduino.

4.3.5 Motors

The two motors that drove the chassis were basic 9-12V motors that came with the chassis. They had built in hall-effect sensor motor encoders but I believe that one of them was defective as it gave extremely inconsistent and inaccurate readings forcing me to abandon the idea of using the motor encoders as part of my robots navigation. Additionally, despite the motors being driven by the same power source directly in initial testing, one motor (the left) consistently was slower than the right. This meant I had to compensate for it in my code so as not to have to fully rely on my line tracking software to correct the motor. The launcher motors were fantastic scrap motors I purchased at an electronics scavenging store. They were quiet, uniform, and extremely current efficient. They were so efficient and quiet that I connected them directly to the power source so that they were always on so as to simplify the rest of my circuits. The last motor was the vibrational motor used to prevent and clear jams in my robots hopper. It had a surprisingly high current draw so I used one of the previously mentioned Darlignton transistor circuits to drive it. I milled a piece of aluminum rod and drilled a slightly off-center hole and then mounted it to the motor spindle to create the vibration effect.

4.3.6 Solenoid

The solenoid was sourced from the same store as the launcher and vibrational motors. It was a basic 12-24 volt solenoid. I used a mill to drill and tap the piston and then ran a set screw through the back so I could control the exact piston travel. Additionally, I included a spring in the barrel behind the piston so that at no current the piston would be extended, and under power the piston would overcome the spring and retract inside the solenoid. When the power was switched off again, the piston would leap forward due to the spring but stop exactly as far as the adjustable set screw would allow. This allowed me to tune the exact stroke of the solenoid to give me control over the feeding mechanism. The solenoid drew a tremendous amount of current at 11.4 volts and I needed to control it via the Arduino so I used a Darlignton transistor circuit to drive it. Like the vibrational motor, the solenoid consists of a powerful electromagnetic coil, so a parallel diode was necessary to not over-dope the transistor with the inductance when power was switched off.

4.3.7 Battery

Due to the high current demands of some of my components, a serious 12v source was required. Both of the chassis motors, the vibrational motor, and the solenoid drew a huge amount of current so I needed something with the capacity and charge to handle it. I used a 11.4V 3C 2200 mAH LiPo battery from a drone that I had built as it had suitable power ratings. Eventually, I bought a second battery and third backup as I could not afford to wait for a battery to charge while building and testing. This battery was able to power the whole robot easily for several runs.

4.4 Schematics

Below are the 3 main schematics used in my robot. Some of the wiring was rather trivial and so I have not shown it in a schematic. Rather I will briefly describe it here.

- The launcher motors are hard wired to the main power source
- The main power source is on a toggle switch
- The Arduino receives power from the main power source on its 12V power input pin

4.4.1 TIP 120 Darlington Transistor Current Switch

Below is a diagram for the circuits I created to switch power to the devices that required higher current like my solenoid and vibration motor.

4.4.2 TB6612FNG H-Bridge Motor Driver

Below is the hookup guide from the Sparkfun website I used to connect my Arduino to my motor driver and chassis motors.

4.4.3 Sunfounder IR Tracking Module

As mentioned previously, my IR tracking module had a 4 wire connector soldered on to the board which I had to remove and solder the wires directly to the board. The wires then connected to the arduino's 5V power, ground, and SCL and SDA pins.

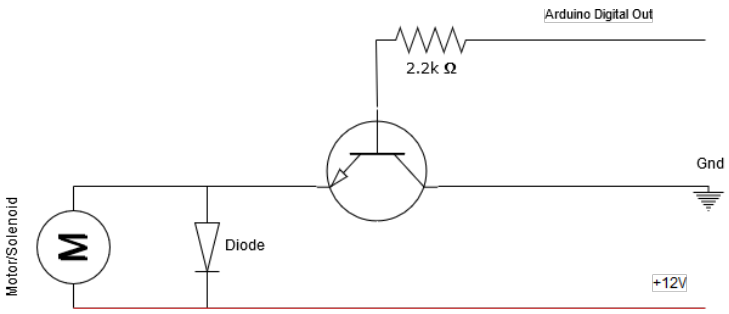

Figure 4: Transistor High Current Switch

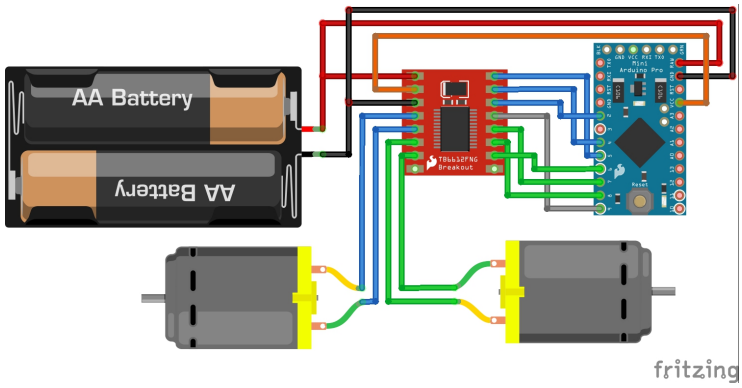

Figure 5: TB6612FNG H-Bridge Motor Driver (Image courtesy of Sparkfun)

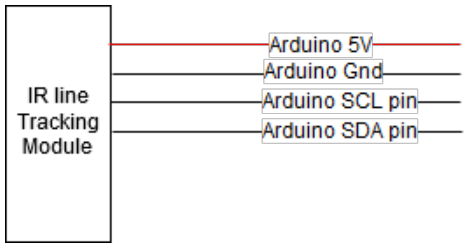

Figure 6: Sunfounder IR line tracker module

5 Mechanical

The overall design of my robot was very simple. The tracked chassis provided a square platform for my main launcher and feed tube to attach securely too. Both launcher motors, vibrational motor, and solenoid were all attached to the launcher itself (see the figure below), which in turn was connected via a hinge to the top plate comprising the main square surface of the chassis. I ran an adjustable bolt through the plate to allow for adjustments in launcher angle. The rest of the design was simply finding room on top of the chassis plate to attach my arduino, circuits, motor driver, and battery. The IR module was attached to the center of the underside of the main chassis in between the treads. Lastly, I made "skirts" of acrylic plastic on the front and rear and right side that kept the projectiles from jamming up the treads or altering the robots direction.

5.1 Materials

5.1.1 Chassis

Given that I was working by myself, I was allowed to use a pre made chassis for my robot. I used a Dagu Rover 5 Treaded chassis with two 12v motors and encoders. It was made of plastic and rubber. I made a top plate to bolt to the chassis out of acrylic. I originally had purchased a top plate but the quality was not very good and the plate soon cracked and became useless.

5.1.2 Launcher

- Tube - 1" ID PVC tubing
- Launcher base - oak hobby wood
- Motor fasteners - Metal hose clamps
- Launcher wheels - Medium sized KNEX brand toy wheels
- Hopper/Feed Tube - PVC tubing

5.1.3 Miscellaneous

I also sourced a few parts from online stores like Pololu.com for some motor shaft adapters. Most materials I either got from my work or bought from the hardware stores in town (Miners Ace and Home Depot)

5.2 Diagrams

Below is a diagram of my robots launcher mechanism as well as hopper and feeding system accompanied by some brief descriptions of the figures.

5.2.1 Launcher

The launcher was made from PVC tubing as well as the hopper. I used a heat gun to mold and form the PVC as necessary and a dremel tool to make cutouts and holes in it. The solenoid inside the launcher tube was held in place by a set screw. I drilled and thread tapped the back of the solenoid piston and then ran a bolt through the rear of the solenoid, through a spring and threaded it into the solenoid shaft. this allowed me to adjust the stroke of the solenoid and also act as a spring and piston retainer for the whole mechanism. The piston also acted as a feed control in addition to fire control as projectiles could not feed into the launch wheels unless propelled by the piston. (Fig. 7)

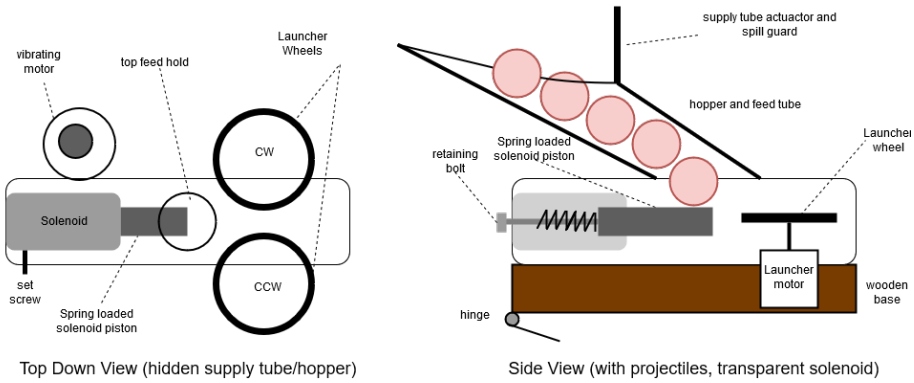

Figure 7: Launcher and Feed Tube Diagram

5.2.2 Photos

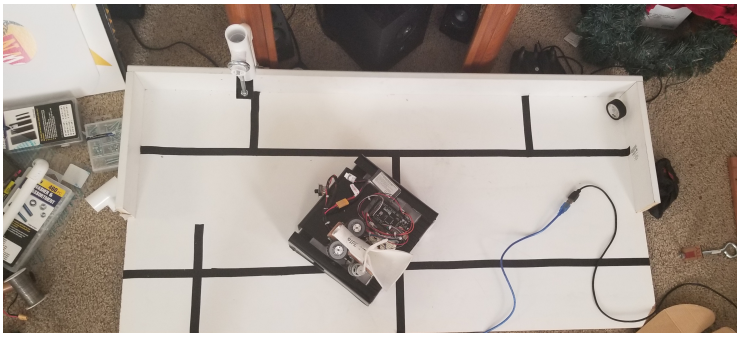

Figure 8: The Final Version of TreadBot

6 Budget and Bill of Materials

6.1 Budget

Since I was working alone, I decided to not really constrain myself to a budget and I just bought what I could afford. My biggest expenses were the chassis itself and the miscellaneous hardware that I bought (motors, solenoids, etc). I spent well over $200, but the final product was probably only about $150 worth of materials.

6.2 Bill of Materials

I bought everything I used in this robot from Amazon, Pololu, Sparkfun, hardware stores, and HSC Electronics. I have receipts for my major expenses and

Figure 9: Before addition of the vibrational motor and skirts

online expenses and some of my receipts from hardware purchases. I will include those as a separate document.

6.2.1 Amazon

On Amazon, I spent a total of around $131 dollars. I bought the following items:

- 2 X Arduino Mega boards
- 2 LiPo Battery Packs
- USB battery bank
- IR tracking module
- Ultrasonic Sensor pack
- Pack of NPN Transistors
- Breadboard and hookup cables
- XT60 Battery connectors

6.2.2 Pololu

Through Pololu, I purchased my chassis, a top plate (which ended up falling apart) and some motor shaft adapters. I spent a total of about $93 on the following items:

- Dagu Rover 5 Chassis
- Dagu Rover 5 Chassis Plate
- Aluminum motor mounting plates (shaft adapters)

6.2.3 Sparkfun

On the Sparkfun website, I purchased my TB6612FNG H-Brigde dual motor driver. I only spent around $10 at Sparkfun.

6.2.4 Hardware Stores

I lost track of how much I spent at Hardware stores. At this point in the project I had already gone over my initial $200 reimbursable budget and there was no looking back. I'm not sure of the exact amount I spent because I lost track of all of the receipts but I spent at least $80 on the following items:

- PVC tubing
- Wood for a mock course
- Miscellaneous assorted hardware, screws, clamps, bolts, etc
- Adhesives
- Raw materials - wood, plastic, acrylic, etc
- Tools

7 Lessons Learned

The problems that I attempted to solve by building this robot brought a lot of unique challenges to the table that required creative and solutions. More often than not, the simplest solution was the best solution.

7.1 What I Learned

7.1.1 Software

I was surprised in how easy getting started with an Arduino was. This project was my first experience using an Arduino and so it did take a little getting used to. Adjusting to the architecture and learning the particular best ways and worst ways to do something was much of a trial and error process. Figuring out how subroutines work and how blocking and non-blocking processes on the Arduino affect the execution of certain functions was tricky as well. Additionally, figuring out how to create complex flows based on synchronously updating flags was a unique challenge in itself. However, once I was able to decompose my design correctly and create modular and efficient functions, the pieces started to fall into place. I was also pleasantly surprised how easy it was to download header files and use them in my project directly from the sparkfun website for the specific motor driver I used.

7.1.2 Hardware

Firstly, I learned how many resources and parts are available for building robots. There are nearly limitless pre-made boards, modules, sensor packs, and custom IC's made and sold for nearly any kind of robot one needs to build. Learning how to integrate my own circuits with the Arduino was a cool experience, and when I finally was able to operate my solenoid using the TIP transistors being controlled by my Arduino it was a pretty great feeling. Additionally, I had no major mishaps or problems with any of my electro-mechanical parts. All of my motors and solenoids and transistors worked perfectly. The only issue I had was with one of the motors and its encoder that came with the chassis itself. I also learned that it is very easy to brick/fry an Ardunio, I went through two of them in this project. I also learned its better to over-plan than under. On item I bought that I didn't think I would end up using ended up being my saving grace on the day of the competition. I purchased a USB battery bank to power the Arduino with and ended up needing it when the onboard 5V regulator fried and the USB in was the only way to power the board.

7.1.3 Mechanical

The mechanical component of this problem was the hardest. The physical constraints of the course with my design and chassis presented a lot of problems in itself. I had a particular issue designing a simple and reliable hopper and feeding mechanism, and this ended up being the primary reason my robot did not do very well. The mechanical aspect of this project was for me by far the hardest.

7.2 What I Would Do Differently

I think that I actually did a disservice to myself by using a pre-made chassis as a custom made one would have allowed me more room in my designs, and allowed me more flexibility when I ran into issues with things like the chassis motor encoder not being accurate or one of the motors being slower than the other. Additionally, I would spend more time researching line tracking algorithms and understanding the best way to use the line tracking module. I definitely would change my hopper and feeding mechanism, perhaps opting for a more complicated feed mechanism in exchange for a much more reliable and simple hopper and re-supply routine. As far as other mechanical aspects, like I mentioned earlier I would have liked to have made my own chassis and used wheels instead of treads as the treads ended up becoming the problem I was attempting to solve by using them in the first place, where as wheels would have avoided them. Poor planning on my part. Additionally, a little more experience might have given me an indication that maybe wall following as opposed to line tracking was the best way to approach this competition. The only thing I would not change would be my launcher. It was remarkably consistent and was probably the best part of my robot.

8 Conclusion

This project was as challenging as it was rewarding. I gained a huge insight into an area directly related to my major in which I had little to no experience; which was my exact goal when I set out choosing robotics as my senior project topic. I think that if I attempted another robotics competition in the future, I would do significantly better after this initial experience. This project was fun at times and immensely frustrating in others. (Particularly when you fry your board at 5AM before the competition with more code left to write). The process of designing and building the robot was mostly enjoyable, but predictably the most fun I had was writing the software to run it. I wish I had more time and more resources, but I tried to make the most of what I had available in terms of time and money given I was working alone. The overall process of building a mechanical system, integrating hardware to drive it, and then finally writing the software to bring it to life is and extremely rewarding and educational experience.

9 Resources

Below are a few resource that I found helpful in completing this project

- http://wiki.sunfounder.cc/index.php?title=Line_Follower_Module
- http://bildr.org/2011/03/high-power-control-with-arduino-and-tip120/
- https://learn.sparkfun.com/tutorials/tb6612fng-hookup-guide