Tony Huang Group 203

Phat Tang

Octavio Gonzalez

Mechanical Systems Laboratory Final Report

I. System Description

1. Design Concept

Mechanical Design

a. Two mechanical design goals for our robot was to optimize steering accuracy and efficient propulsion for maximum speed.

b. The final steering design utilized a rack and pinion. This design was unique because the rack was fixed onto a wood block that was fixed to a horizontal guide that only allowed side to side motion. This design prevented the rack and pinion from skipping and unmating. With some lubricant on the flat metal bar, the steering glided almost effortlessly. The rest of the steering was up to the servo to do its job. This design was very successful because it allowed for a gradual and smooth turning that went hand in hand with the code uploaded to the Arduino sketch to change steering angles. The biggest success of this design was showcased when it corrected itself accurately and almost effortlessly towards the 150-point course marker by the magnetometer, given the code and offset were properly set.

   The final propulsion design utilized a rack and pinion setup as well. This design set apart from many other robots right from the start because it utilized a one way bearing, one rack, and one pinion. The rack and pinion were 3-D printed which

was very light. This aided in reducing wasted power from piston to get the propulsion rack and pinion moving and instead redirected the power to the rear wheel drive. The power stroke utilized the spring return stroke of the air piston and this was very successful because spring rates are usually gradual and even which translates to a smooth rolling robot. This design deemed successful in the competition when coupled with a successful firing rate, it took off from start and accelerated faster than all the other robots. In order to avoid crashing into other robots or joining robot pile ups, this design was crucial in order to get into the trench as soon as possible.

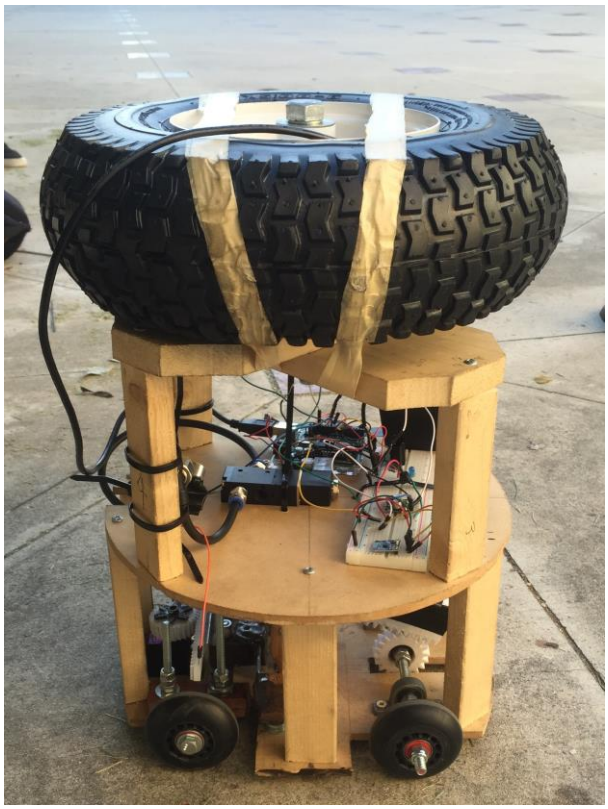
Software Design

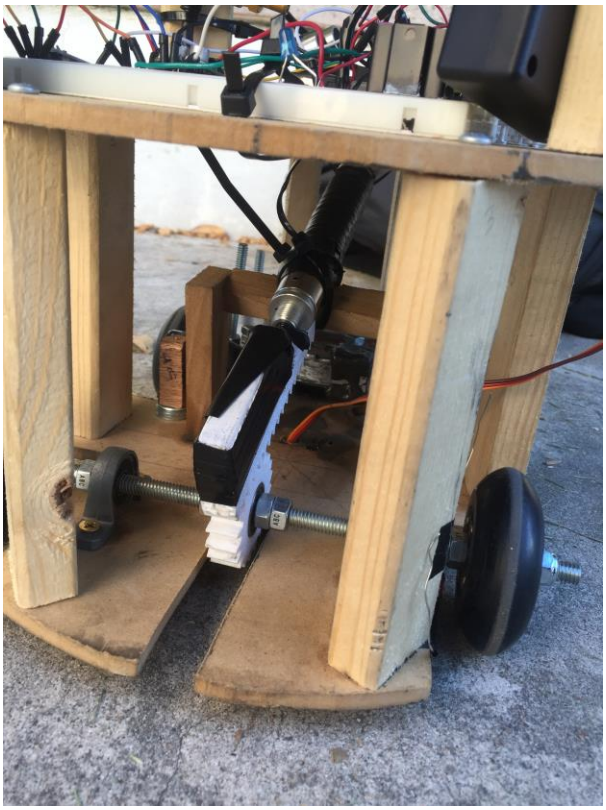
a. Two software design goals for our robot was to optimize steering from start into trench and to fire piston at faster speed once it hits the straight towards the 150-point marker.
b. The final Arduino sketch utilized a hard code in order to get the robot smoothly turned into the trench without any hard turns which resulted in the robot tipping over. This also prevented the robot from hitting the trench wall even if the robot had to start at the position right next to the wall. This code used “pistonfire()” lines to fire the piston a number of times from the starting position at 90 degrees(straight) and had small adjustments in between each pistonfire() to gradually turn the robot into trench. For example, 4 lines of pistonfire() fired the piston 4 times while the robot was facing the straight position, followed by a servowrite(120) and a couple more pistonfire(), then a servowrite(140) and a couple more pistonfire(), lastly written back to 90 and allow for the magnetometer

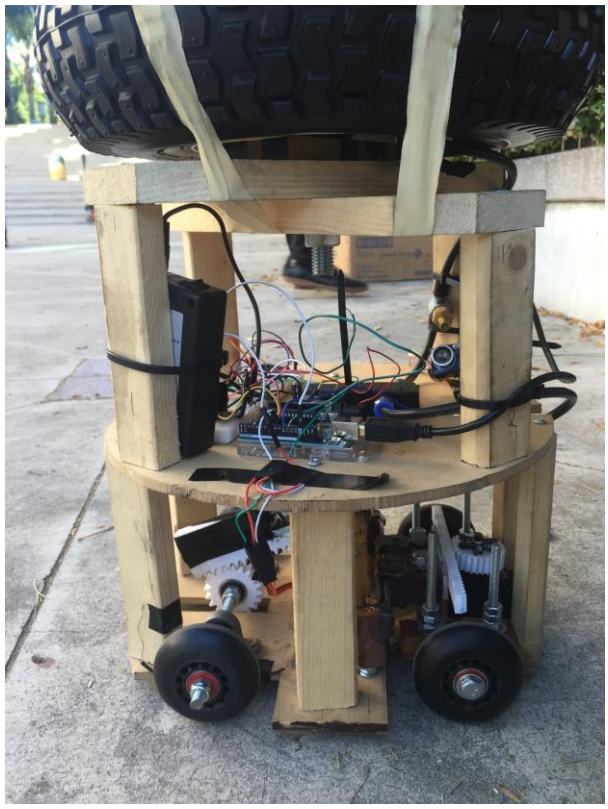
to correct the rest of the way. This code was successful because we had to find a start point and the code would work at any other spots in starting with the exception of adding or subtracting a pistonfire() depending on the distance it needed to get to trench before beginning gradual turn. As shown several times in the competition, the code led the robot to the trench, turned, then gave the reigns to the magnetometer and fired to the 150-point marker.

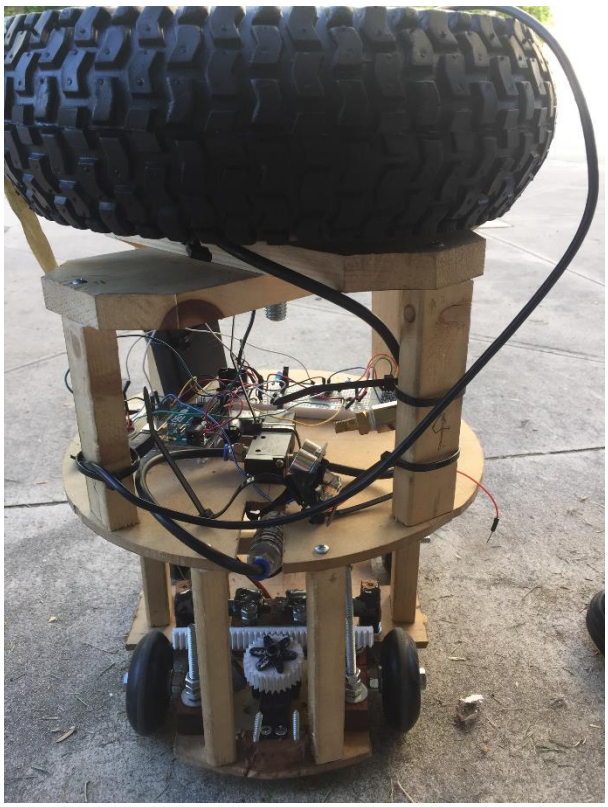
The second software design goal was to avoid collision and pile-up with other robots in the beginning by firing the piston at a faster speed after the turn from start to get out of the pile-up zone was necessary. The Arduino sketch utilized the hard code in the beginning to get to the trench smoothly and quickly jumped into the loop where it initialized a faster firing speed, “pistonfire2(). This code was successful because the robot was already almost facing straight towards the 150-point marker so there was little fear in the robot tipping over. Tipping over was worrisome at the start because too much speed into the turn cause a tip over. This secondary firing code quickly propelled the robot out of the zone right before the trench walls where crashing occurs most.

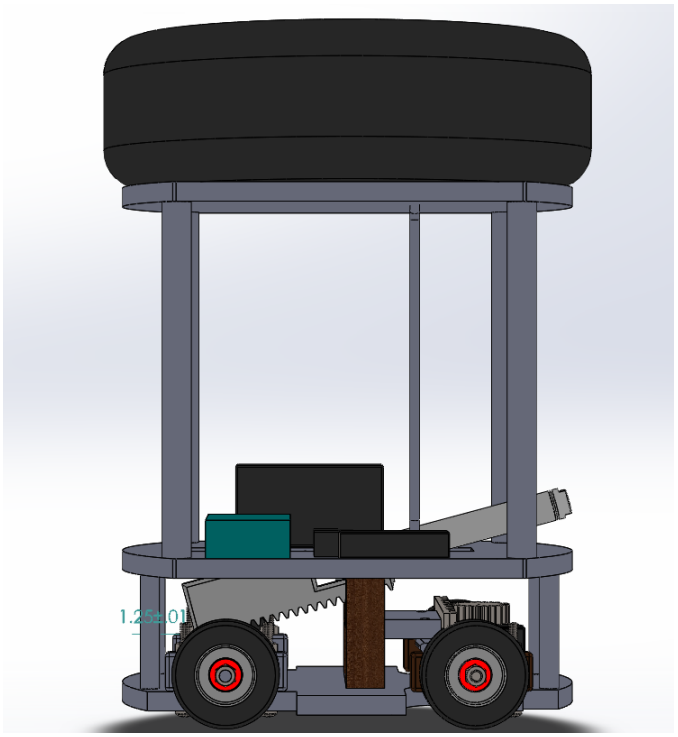
2. Photograph and CAD Model

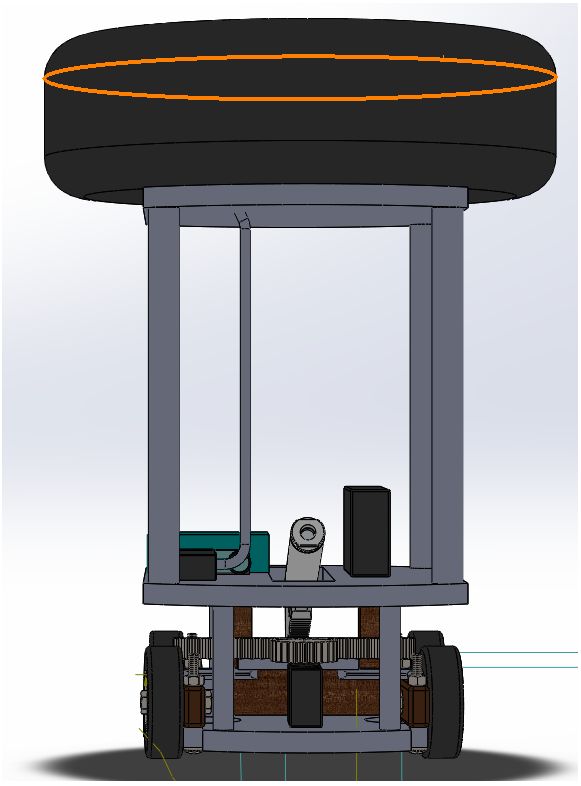


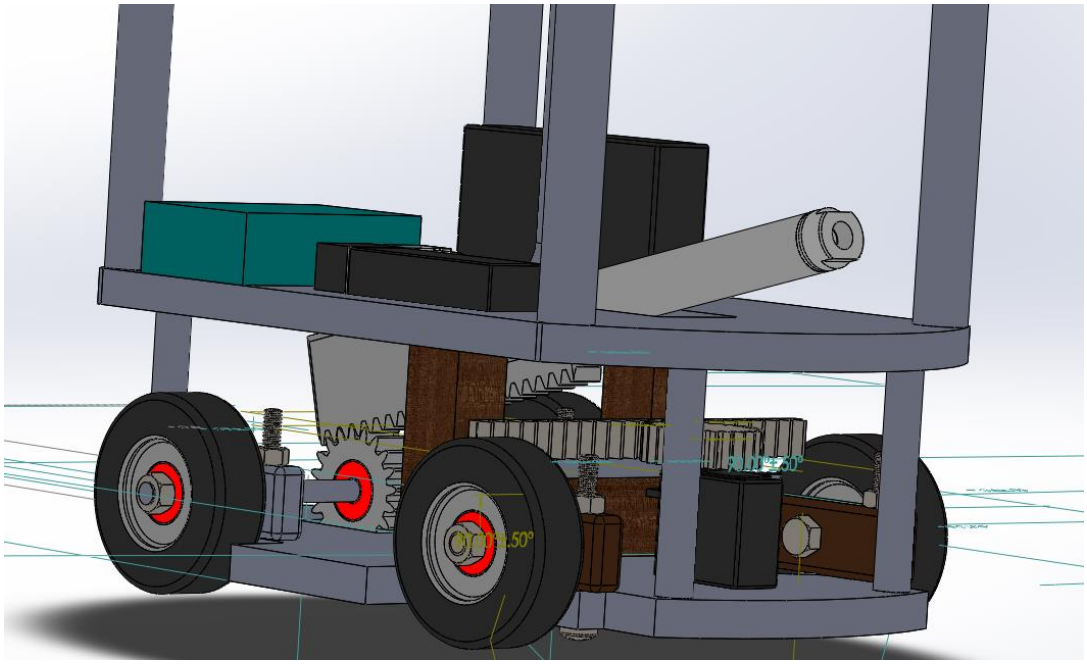


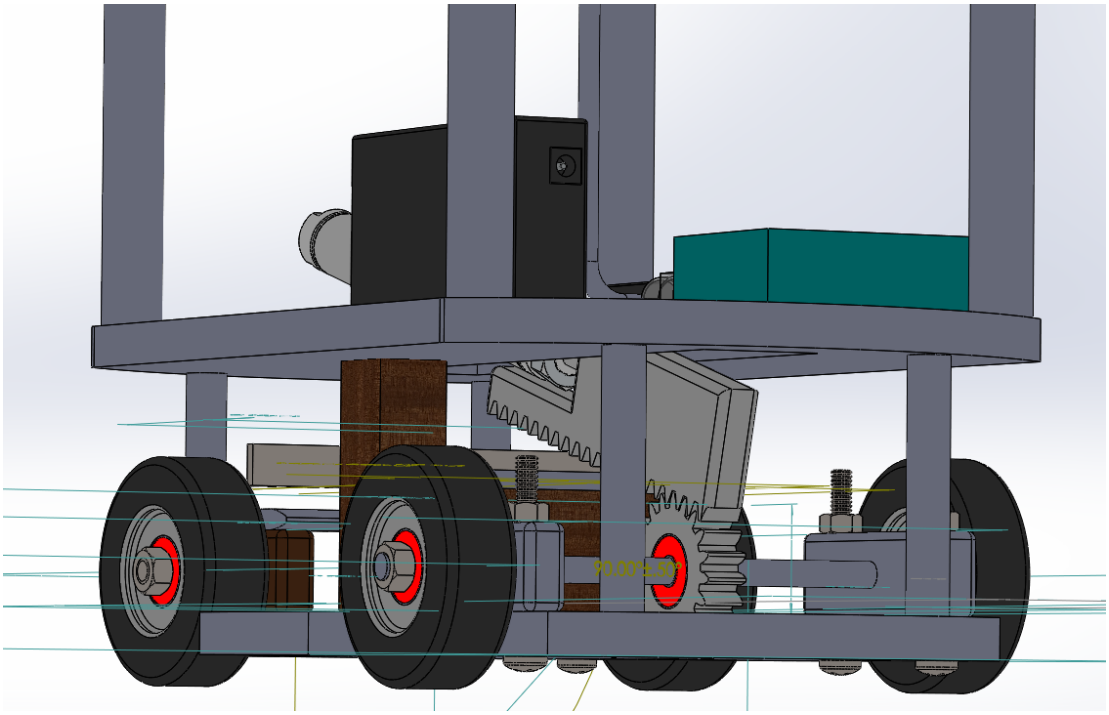












3. Electrical and Pneumatic Circuit Diagrams

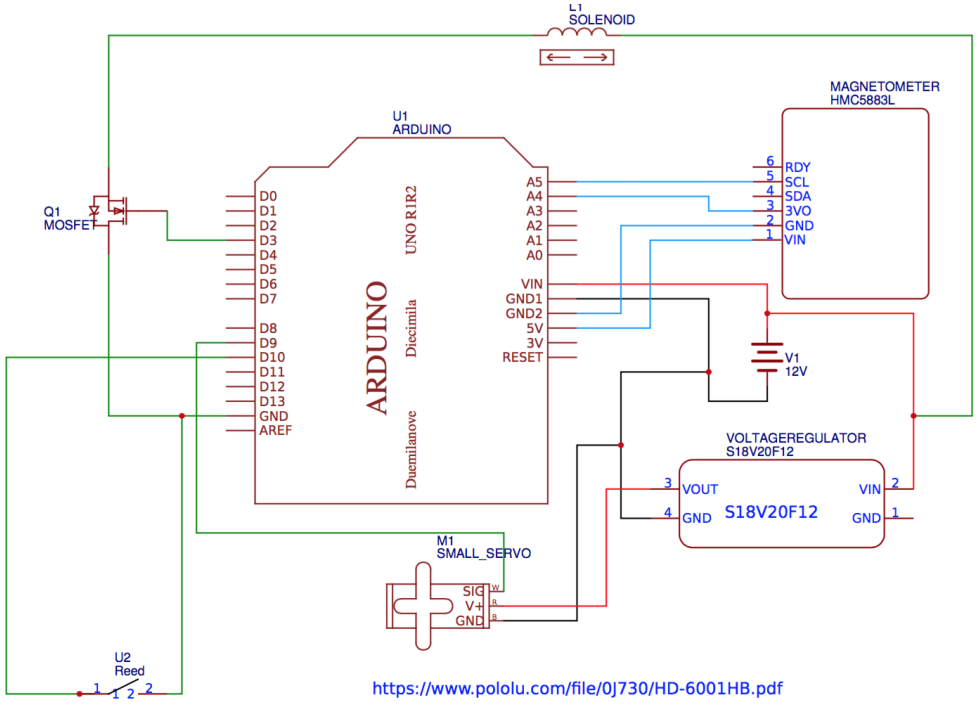


Figure 1: Electrical circuit diagram

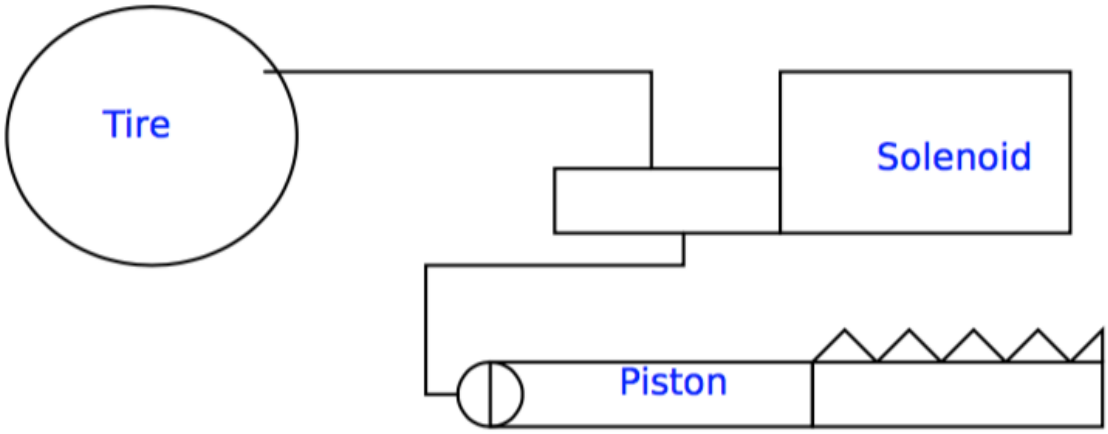


Figure 2: Pneumatic circuit diagram

4. Software Code

```
/*
* magnetometerExample is a program that makes a servo (connected to pin 9) follow the angle your robot is facing in regards to North.
* To avoid damaging the servo, it will not be actuated if the angle is greater than 170 or less than 10 degrees.
* The x and y components of the magnetic field, as well as the calculated angle, are presented in the Serial Monitor.
* Author: Diogo Schwerz
*/

#include <Wire.h> //I2C Arduino Library
#include <Servo.h>
unsigned long timer;
#define address 0x0D // I2C 7bit address of the magnetometer
// Copy the numbers for the following variables from the calibration program YOU ran:
// The calibration values may change slightly depending where you are and through time.
// For better results, make sure to re-run the calibration from time to time or when running your robot in a different location.
int xMin = -4552;
```

```
int xMax = 967;
int yMin = -60;
int yMax = 5737;
int mosfet=3;

// Angle variables
float thetaOffset = 37.53;//change this variable at your need
float theta;
// Magnetometer variables
int xRaw,yRaw,zRaw; // raw numbers
float x, y, z; // calibrated numbers
// Servo variables
Servo myservo;  // create servo object to control a servo
int maxServo = 160;   //Set max servo angle
int minServo = 20;    //Set min servo angle
// Reed Switch Variables
int reedswitch=10;   // assign pin 10 for reedswitch
float distance = 0;   // distance travelled
float distance1;
float radius = 1.5;   // radius of wheel [inches]
const float pi = 3.1459265359;    // define pi
void setup(){
  //Initialize Serial and I2C communications
  Serial.begin(9600);
  myservo.attach(9);  // attaches the servo on pin 9 to the servo object
  // Set pin for MOSFET and Output to Air Solenoid
  // Set pin for REEDSWITCH and Write Data to Serial Monitor
  pinMode(mosfet,OUTPUT); // Set the mosfet pin (3) as an output
  pinMode(reedswitch,INPUT_PULLUP); // Sets the reedswitch pin (10) as an input
  pinMode(13,OUTPUT);
            digitalWrite(13,HIGH);

  Wire.begin();

  Wire.beginTransmission(address); //open communication with HMC5883
```

```
  Wire.write(0x09); //select mode register
  Wire.write(0x0D); //continuous measurement mode
  Wire.endTransmission();

  Wire.beginTransmission(address); //open communication with HMC5883
  Wire.write(0x0B); //select Set/Reset period register
  Wire.write(0x01); //Define Set/Reset period
  Wire.endTransmission();
  delay(16000);          //Set 16 second delay to prevent false start and disqualification
  myservo.write(90);  //Set servo position to 90 degrees(Straight) in the beginning before propulsion
  pistonfire();          //1 pistonfire() equals 1 fring stroke of air piston
  pistonfire();
  pistonfire();
  pistonfire();
  pistonfire();
  pistonfire();
  pistonfire();
  myservo.write(120);   //Begin gradual servo turn to 120 degrees
  pistonfire();
  myservo.write(140);   //Continue gradual servo turn to 140 degrees
  pistonfire();
  myservo.write(90);  //Return servo to straight 90 degrees and let magnetometer in loops correct the remainder of the way
  delay(200);   //Set a 200ms delay before main loops initializes
}
void loop(){
  timer=millis();   //Setup if statement to stop air solenoid if it pasts 45 seconds to avoid disqualification
   if(timer > 45000)    //timer in ms
  {
            exit(0);
  }
  else{
  updateMagnetometerData();

  pistonfire2(); // Calls function set down below this code to fire the piston at a faster rate
```

```
 int proximity = digitalRead(reedswitch); //Initializes a variable to store the state of the reedswitch
 if (proximity == LOW)
 {
        int pi = 3.1459265359;
        distance1 = distance + ((radius*2*pi/4)/12);        // distance in feet
        distance = distance1;
 }
 Serial.print("distance: "); Serial.print(distance,5); Serial.println("\t");
 // Calculated the calibrated variables from the magnetometer's raw readings
 x = 2 * (xRaw - xMin) * 1.0 / (xMax - xMin) - 1;
 y = 2 * (yRaw - yMin) * 1.0 / (yMax - yMin) - 1;
 theta = 180. / 3.14159 * atan2(y, x) + thetaOffset; // convert to degrees, apply offset
 theta = modulo(theta, 360.); // ensure that theta is between 0 and 360 degrees

 // Print out values of each axis and theta from magnetometer
 //Serial.print("x: ");
 //Serial.print(x);
 //Serial.print("\ty: ");
 //Serial.print(y);
 Serial.print("\ttheta:");
 Serial.println(theta);
 // If theta is inside the servo's limits, change the servo's direction
 if(theta > minServo && theta < maxServo)
 {
        myservo.write(theta);
 }

 delay(10);
 }

}
float modulo(float dividend, float divisor) {
 float quotient = floor(dividend/divisor); // find quotient rounded down
 return dividend - divisor*quotient; // return the remainder of the divison
}
```

```
void updateMagnetometerData()
{
  //Tell the magnetometer where to begin reading data
  Wire.beginTransmission(address);
  Wire.write(0x00); //select register 0, X MSB register
  Wire.endTransmission();

  //Read data from each axis, 2 registers per axis
  Wire.requestFrom(address, 6);
  if(Wire.available()>=6){
                xRaw = Wire.read(); //x lsb
                xRaw |= Wire.read()<<8; //x msb
                yRaw = Wire.read(); //y lsb
                yRaw |= Wire.read()<<8;; //y msb
                zRaw = Wire.read(); //z lsb
                zRaw |= Wire.read()<<8; //z msb
  }
}
// Funtions used to fire piston(turn on air solenoid)
void pistonfire()   //function used in the "hard code" portion before the loop
{
  digitalWrite(mosfet,HIGH);    //Opens air solenoid to fire piston
  delay(225);              // set a 300ms delay
  digitalWrite(mosfet,LOW);     //Closes air solenoid to allow piston to retract
  delay(225);              // set a 300ms delay
}
void pistonfire2()   //function used in the main loop to fire piston at a faster rate in the trench straight
{
  digitalWrite(mosfet,HIGH);    //Opens air solenoid to fire piston
  delay(200);              // set a 200ms delay
  digitalWrite(mosfet,LOW);     //Closes air solenoid to allow piston to retract
  delay(200);              // set a 200ms delay
}
```

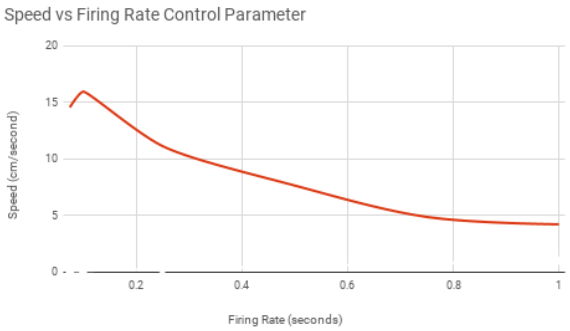
Testing and Development

1. *Experimental Testing*

    a. The two performance criteria that tested and improved on was the speed of the piston firing and the angle of the servo as it takes the turn into the trench.

    b. The experimental parameter was the piston firing, and we varied its speed between 100ms to 1000ms, the control parameters were the distance and the pressure inside the tire. The distance was kept constant, which was 89.5cm and the pressure of the tire was 40 PSI. We plotted the speed of the piston firing VS the time it took to travel the constant distance of 89.5cm and the speed of the piston firing VS the distance it traveled with a constant PSI of 40 in the tire (below are the graphs). The second experimental parameter was changing the amount of degrees the servo was turning before it got to the trench. We kept the distance constant and measured the time. We also kept the firing rate constant and measured the stability of the robot on a scale of 1-10, with 1 being unstable and 10 being really stable.

    c.



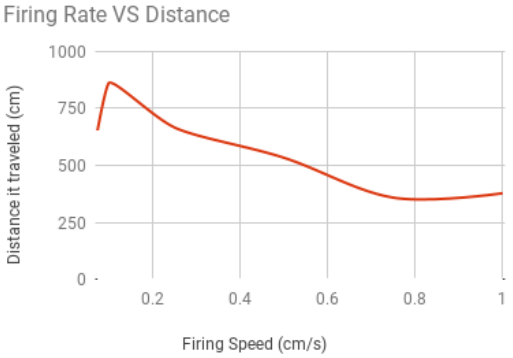
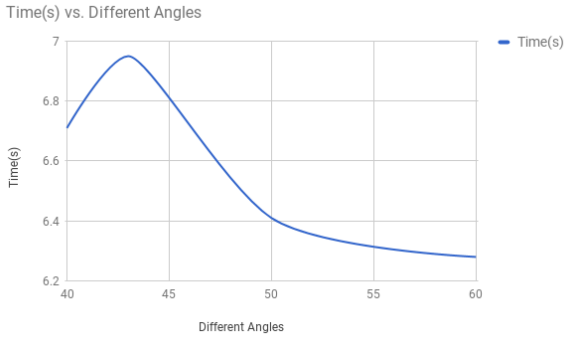


Figure 3: Speed VS Firing Rate

Figure 4: Distance VS Firing Rate



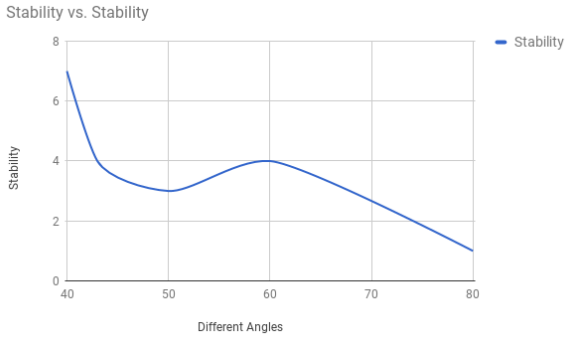


Figure 5: Time VS Different Angles

Figure 6: Stability VS Different Angles

d. Based on Figure 3, we found that the optimal speed was when the piston fired at 100ms per stroke. For the fastest time to enter the trench we noticed the best angle to steer was around 40 degrees. Based on Figure 5 we can see that an angle of less than 40 would give us a slower time and an angle larger than 40 would give us a slower time and we also lose stability. We noticed that the higher than angle was, the more shakey the robot was and when we got close to 80, it would sometimes fall on its side.

2. Comparison with Mathematical Model

For the mathematical model we wrote the distance traveled as a function of piston strokes. Our gear ratio was set so that one piston fire would equal to 1 revolution of the wheels. The known parameters in our model are the piston stroke length and the axle gear circumference which are both 4 inches. This means that for every piston stroke the gear completed one revolution. We also know the diameter of the wheels which is 1.5 inches. Being fixed on the same axle as the axle gear the wheels also complete one full revolution per piston stroke.

Therefore in theory for every piston stroke the distance traveled by the robot should be equivalent to the circumference of the wheel which is about 4.7 inches. From these parameters we derived the equation

$distance\ traveled\ =$ wheel circumference * # of piston strokes

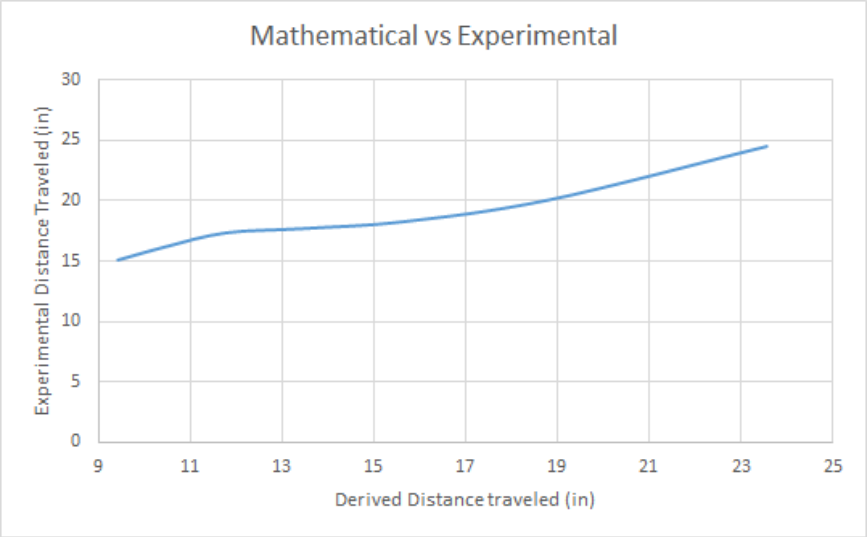
We then integrated the piston fire delay to calculate the distance traveled. We set the delay to be equal for the piston to open and close therefore the piston fired once every 2*piston fire delay.

$$distance\ traveled\ in\ one\ second$$

$$= \Pi * diameter\ of\ wheel\ * number\ of\ piston\ strokes\ in\ on\ second$$

where $number\ of\ piston\ strokes\ per\ sec\ = (\ 1\ sec\ /\ 2 * piston\ fire\ delay\ in\ ms) * 1000$ The unknown parameters are the friction between the wheels and the ground as well as the friction between the rack gear on the piston and the axle gear. Also our model is based on the assumption that the time that the piston takes to open is a lot shorter than the delay time.

Distance traveled (in)	Experimental distance traveled (in)	Difference
9.424778	15.1	5.675222
10.47198	16.2	5.728024
11.78097	17.35	5.569028
13.46397	17.7	4.236031
15.70796	18.3	2.592037
18.84956	20.1	1.250444
23.56194	24.5	0.938055



From the experimental data we concluded that our model is more accurate at faster piston fire rates than for slower fire rates do to the fact that the momentum of the robot kept the robot rolling in between the piston fire delay therefore traveling a farther distance than expected from mathematical model. At faster firing rates there is less time in between piston fires for the robot to roll freely under its momentum and therefore the mathematical model better defines the distance traveled by the robot at faster piston fire rate.

Summary of Contributions

1. Tony

a. Uploaded arduino sketch to controller and edited variables for piston firing and steering in order to achieve accurate robot steering and piston firing while testing before competition.

b. Cut the wood in order to assemble the base, mid-base, and top base. Framed robot together.

c. Calculated reed switch variables and equations to output the correct distance and units.

d. Wrapped air piston in electrical tape for friction against mount to prevent sliding.

e. Came up with the idea for a one way bearing coupled with rack and pinion propulsion.

2. Phat

a. Soldered necessary wires and wired all connections for air solenoid, magnetometer, reed switch, MOSFET, battery and voltage regulator.

b. Changed magnetometer parameters in the code for control of the robot

c. Tested the servo to make sure it steers properly, using the sweep code in the Arduino library

d. Successfully solved the issue regarding the malfunctioning MOSFET which led to successful air solenoid function.

3. Octavio

a. Mounted Arduino Uno controller, battery, circuit board, pillow bearings, piston, etc.

b. Rewire connections after mechanical modifications were made.

c. Figured out to integrate code to stop air solenoid firing at 60 seconds.

d. Completed entire robot design on solidworks.

e. Came up with idea for rack and pinion steering along with metal rod guide that prevented the servo from skipping rack and pinion teeth.

2. One lesson learned about working on an engineering design team is things will always go wrong. As soon as you think you got the code, mechanical, and electrical connections down, paired with successful testing, things start malfunctioning. For our robot, bolts and mounts became loose so gears would not mesh properly. Tolerances would increase the more tests we ran on the robot. We had a successful robot Saturday, issues arose Sunday as soon as more tests were ran. Ended Sunday with a properly running robot. Ran a few more tests Monday night before Competition Tuesday at 8AM and the servo mount broke. Had to go back to mechanical phase and fix and recut wood. Fixed issue, ran robot successfully twice and left before more

testing led to more chaos. Although we ran 50+ successful tests Saturday, we learned much about our robots functioning, where it excels and where it needs improvement. There were probably over 25 iterations of the original code to get the robot to work flawless. Even when the code was set, tolerances in mechanical end called for a need to adjust code to adapt to tolerances. Overall, designs for anything, in our case, robot, can always be improved and made better. But this comes with the price of parts breaking and rebuilding, fixing, improving in this endless cycle of engineering.

The second lesson learned about working on an engineering design team is all parties, mechanical, connections, and controls have to be on the same page and be able to meetup at the same times. Groups of one were seen while testing at engineering gateway that had to carry the entire team because other team members were either busy or prioritized other tasks before 106. Felt this was vital in our group's success because we always met together and set time aside to get everything done. There is always something a team member can do whether it's part of their roles or not. Teams working together also learn respective roles and can help one another. For instance, Tony chose the controls, but Phat was exceptional in diagnosing and solving issues to get the sensors to function properly. Tony aided Octavio in mechanical building of the robot because Tony had past experience with tools. Octavio learned the connections and was a backup in case Phat was unavailable.