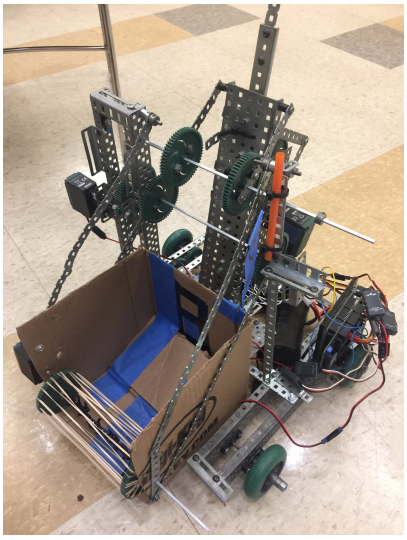
RBE 1001 Introduction to Robotics Engineering
A Term 2017
Final Project Report Team 8



Member	Signature	Contribution %
Henry Poskanzer	Henry Poskanzer	20%
Ying Zhang	Ying Zhang	40%
Yujia Qiu	Yujia Qiu	40%

Grading:

Presentation	/20
Design Analysis	/30
Programming	/30
Accomplishment	/20
Total	/100

Table of Contents

List of Figures

Section 1- Introduction

As a team, we are to design, build, and program a robot for the savage soccer robotics competition. The main focus of the competition is to earn points when our robot picks up small objects, known as EGGS, and delivers them to baskets. There are three different baskets on the field that we can use to score points, the COOP, the PEN, and the NEST (see figure 1). Each basket is at a different height above the ground, and it is worth more points to deliver EGGS into higher baskets rather than lower ones.

At the beginning of the game, our robot must run autonomously for 20 seconds, using sensors to track the white line on the field to score points by either driving onto the RAMP, pushing the PEN out of the vertical projection its starting area, or putting EGGS into baskets. During this period, it is worth more points to deliver EGGS. After that, the teleoperated period, which is 2 minutes in total, starts. Two members in our team will remotely control our robot from our driver station on the side of the field. At the end of the match, which is the last 20 seconds of the teleoperated period, if the robot is fully supported by a PERCH, it will earn 30 points. The team which earned the most points will win.

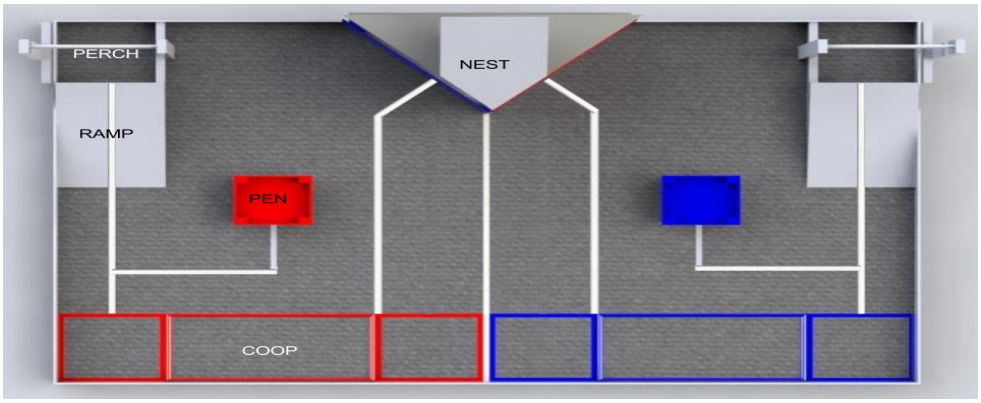


Figure1

Section 2- Preliminary Discussion

For the autonomous period, we considered the limitation of time and the restriction of control. We thought it would be best to either get on the RAMP or push the PEN out of the vertical projection of its starting area because there is white tape on the black ground from the robot's starting position to each of these features and we can use photoresistors to easily track the lines. The robot could also use rangefinders or bump switches to locate EGGS, but their starting positions are random, and we could not guarantee that we would find any EGGS. If we program our robot to deliver EGGS during the autonomous period, we would get potentially better but unpredictable results. Therefore, we choose to use our autonomous period to push the PEN outside its starting position or drive onto the RAMP.

For the teleoperated period, our initial thought is to have a lifting arm with a clamp on it, so we can collect EGGS and deliver them into baskets. After the preliminary design review, however, we realized that indeed this strategy is easy to achieve but at the same time, it is inefficient since we can only grab one EGG at a time. It also requires us to line up our robot perfectly with each EGG, which is time-consuming. Hence, we made a big change on our design of collecting EGGS. We decided to use a box with a swiper for storage and collecting. And instead of a lifting arm, we decided to use a four bar linkage structure attach to the box for lifting to make the process more stable. This design will allow us to carry more EGGS at one time and earn more points.

For the end of the game, our thought is that we should earn as many points as we can. So we decided to design a hanging mechanism so that our robot can be fully supported by the PERCH.

Section 3- Problem Statement

Based off of our main objectives and restrictions set by the rules, we have created a priority list.

Priority list:

1. Fully operated four-bar linkage structure.
2. Feasible roller size that can successfully get the EGGS in and out of the box.
3. Making sure that the motor has enough torque to lift the box.
4. Stability of the box during the process of lifting.
5. Use the line tracker to ensure that the robot reaches its destinations during the autonomous period.
6. Use sensors to figure out when the robot has reached its destinations and stop during the autonomous period.
7. Making sure that the hanging arm can reach the PERCH.
8. Making sure that the hanging arm is strong enough to lift the robot itself.

Our major consideration would be the stability of the four-bar linkage, because there is only one motor attached to the crank, the two 4-bar linkages on each side of the box may not turn at the same rate, which may caused the EGG box to become unstable during the lifting process. Another concern is how to program the roller to roll clockwise when collecting balls and roll counterclockwise when the box is lifted to a certain position. Another potential problem is that the hanging arm can only move straight up and down, even though it needs to reach out to the PERCH. The arm is going to be quite heavy, and the motor may not have enough torque to

stabilize the arm at a desired position. Hopefully we can solve this problem and successfully finish our task.

Section 4- Preliminary Designs

For the purpose of storing and collecting EGGS, we will use a cardboard box with one side open and a roller attached inside. This idea was inspired by many FTC robot teams found in online videos. Most of them use a roller to collects balls. Many use a mechanism like popping the box up and down, or an incline angle to get the balls out of the box. However, we decided to use roller for getting the EGGS both in and out in order to make the design less complicated. The feasibility still needs to be tested, though. The roller will be driven by a VEX 393 motor, and when it's rolling clockwise, it will swipe the EGGS into the box. The box is also attached to a four bar linkage structure where the crank is also driven by a VEX 393 motor. The four bar linkage structure will lift the box to the desired position above a basket, and the roller will be rolling counterclockwise to get the EGGS out of the box.

For the end of the match, we have two designs for a device that hangs our robot on the PERCH. Our original design involves a hook attached to the edge of a gear. When the gear turns, it pulls the hook down, pulling the rest of the robot up. The hook is attached to the gear by a single pivot so it stays vertical as the gear rotates. In order to extend the range of motion, we can attach a bar to the gear at two points and attach the hook to the end of the bar by a pivot. Figure two shows a diagram of this device.

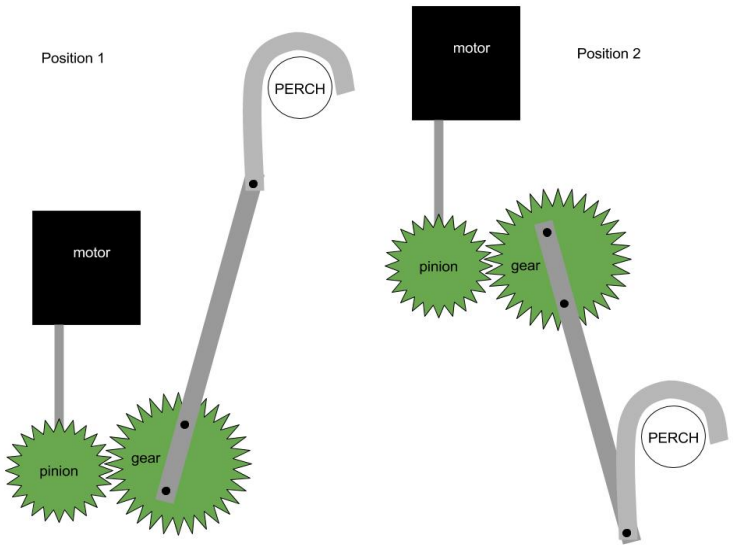


Figure 2

Our second design is more traditional. It involves a rack and pinion. When the pinion turns, its pulls the rack downward, which is attached to a hook. When the rack and hook are pulled downward, the base of the robot is pulled upward. Figure three shows a diagram of the rack and pinion design.

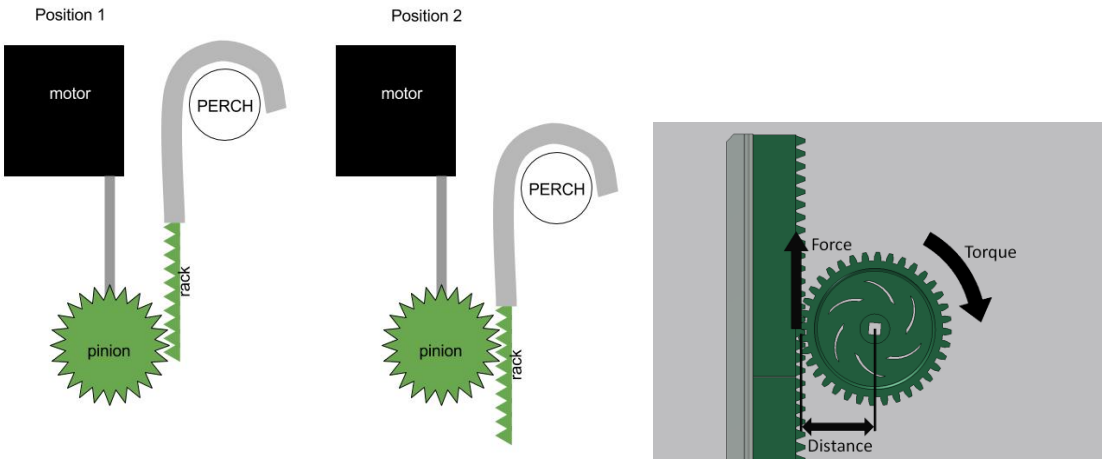


Figure 3

Section 5- Selection of final design

For our hanging device, we chose the rack and pinion design. As a team, we are more experienced with rack and pinion systems.

The major adjustment that we made from our preliminary design of the hanging device is in the gear train. We calculated the torque required using the rack and pinion system with one gear, which is more torque than the Vex 393 motor can continuously provide (the maximum operating torque is ½ of the stall torque). Hence, we decided to use a two stage gear combination to increase the torque. Figure 4 is our final design for the hanging system.

The redesigned gear train uses one 24-tooth, one 36-tooth, and two 60-tooth gears. Using the equation e = Teeth in/ Teeth out = Torque in/ Torque out, (⅕)*(⅗)=(3/25). The output torque is 7*12.5 (the arm length) = 87.5 inlb. Using the gear train ratio, the input torque is 10.5 inlb.

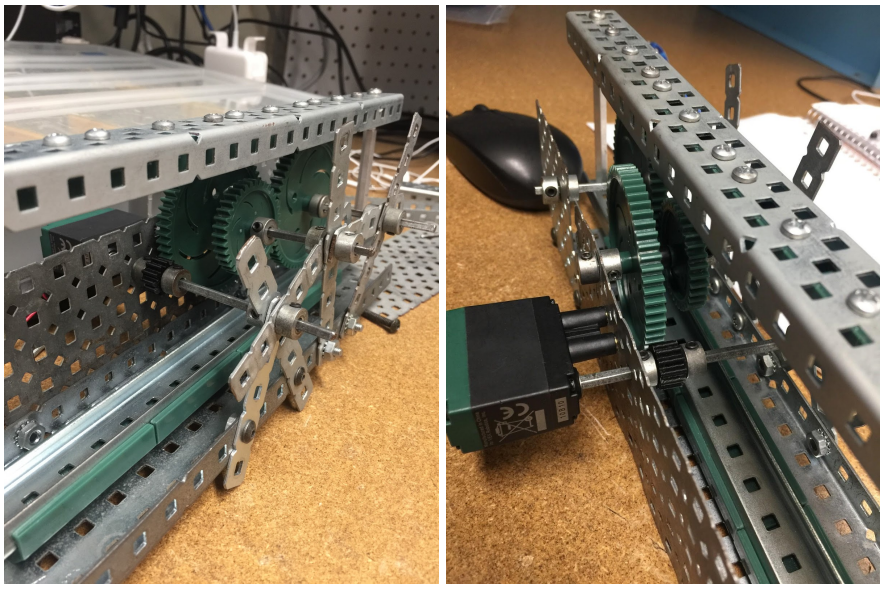


Figure 4

As for the EGG-collecting system, we want the whole device to be as light as possible so that the four-bar linkage system could work properly even when the cardboard box is filled with EGGS. Our roller is made by the combination of many zip ties, which are light and long enough to smoothly get the EGGs in and out. Figure 5 is our final selection to collect and drop EGGS.

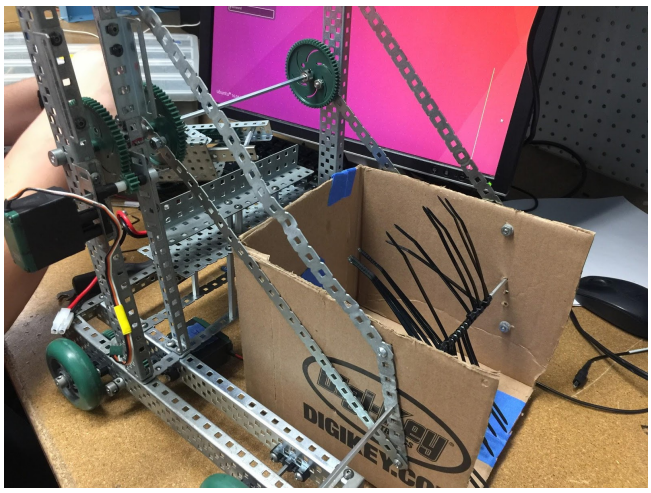


Figure 5

Section 6- Final Design Analysis

Estimated Weight: 7 lbs(likely to change)

Diameter of the Wheels: 2.4 inches

Estimated Coefficient of Friction: 1.0

Estimated Center of Mass: halfway between the front and back wheels

The RAMP is 30” long, 18” wide, and 6” tall at the higher end. -- so the angle is 11.53 degree

As part of our design process, we mathematically analyzed the robot for its driving effort. We also computed the power requirements for operating a four bar linkage to lift the EGG box, and lifting the robot on the perch.

I. The weight of robot is 7 lbs. The friction-limited traction force is $F = \mu N_{driven}$. The estimated value of μ is 1.0. Since the center of mass is halfway between the front and back wheels, the normal force on the driven (back) wheels is half the weight of the robot, or 3.5 lbs. So the friction-limited traction force on level ground is F = 3.5 lbs. Therefore, the total torque on the driven wheels is $\tau = F * D/2 = 7 * 2.75 / 2 = 9.625$ inlb. This torque is divided evenly between two wheels, so torque on each wheel is $\tau /2 = 9.625 /2 = 4.8125$ inlb. Using linear interpolation from motor data, the power required is 3.844975 W.

II. These are the calculations for the power required to lift the four bar linkage controlling the EGG box. For this required power, we need a Vex 393 motor.

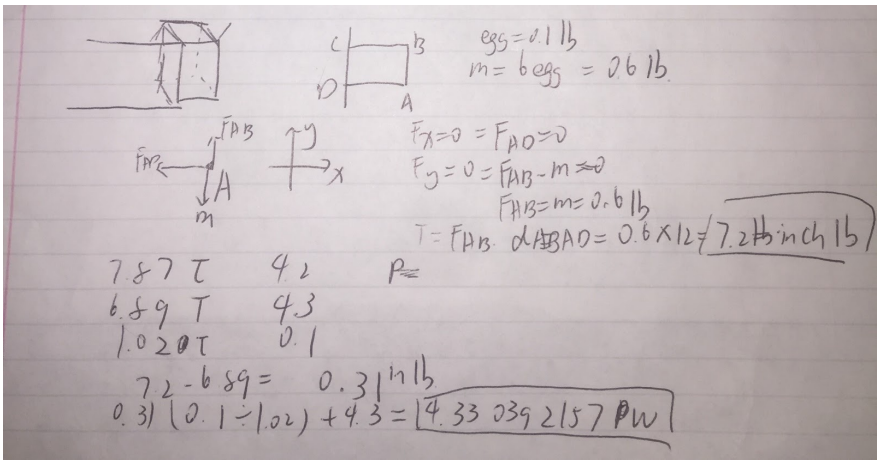


Figure 6

III. In order for our robot to hang on the PERCH at the end of the match, it will need a motor and possibly a transmission. At first, we hoped to use a simple rack and pinion system. However, our calculations(Figure 7) show that a simple rack and pinion will not provide enough lifting force. Instead, we must use a transmission(Figure 8) to increase the output force.

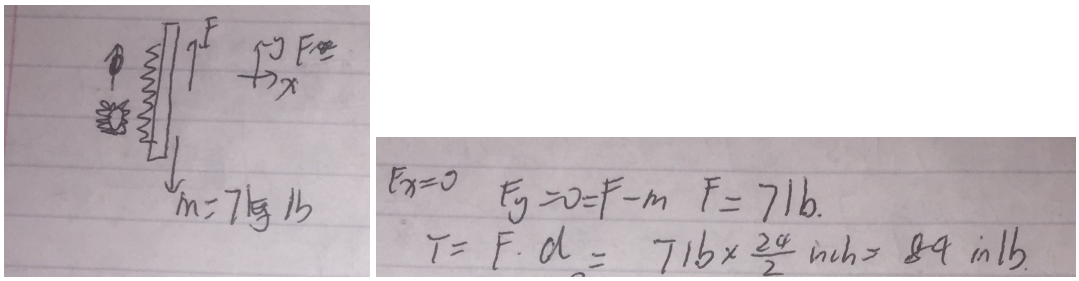


Figure 7

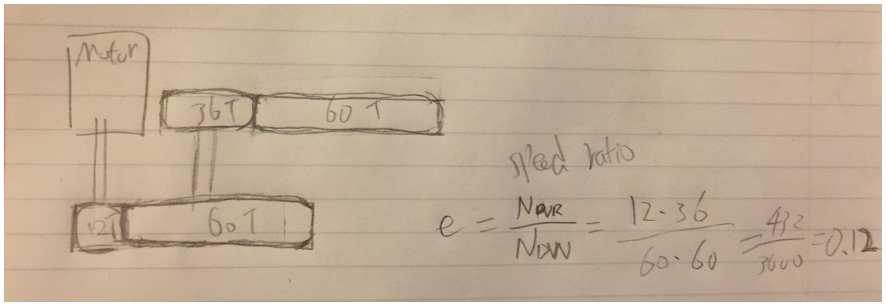


Figure 8

Section 7- Summary/Evaluation

During the critical design review, our robot did not perform as well as we had hoped. The driveline worked as expected, but several other systems failed. During the autonomous period,

the robot veered to the left, even though it was programmed to drive straight. Our analysis is that the program did not account for the difference in the exact motor speeds and tractive forces. In addition, the EGG-collecting roller worked inconsistently. It rotated quickly, as we had designed, but the zip-ties often pushed EGGS away from the box, rather than into it. The roller did successfully catch a few EGGS, but overall, the results were unsatisfactory. At the beginning of the run, the motor that lifts the EGG box was accidentally turned on. We had not programmed a way to turn the motor off. It continued driving the four bar linkage at its maximum position, stripping one of the gears. Even though we replaced the gear, it continued to get stripped. As it turned out, gears were being stripped because the two gears (the 60-tooth gear and the 12-tooth gear attached to the motor) were held too close together, so the large gear was stripped as it was pushed away from the small gear. Later, when we actually wanted to use to four bar linkage, it was not operational because the stripped gear continuously rotated around its axle. Lastly, the robot was able to drive onto the ramp, but the hanging arm could not reach all the way to the PERCH. It simply was not long enough or angled correctly.

Before the OED later that night, we made several improvements to our robot. On the EGG-collecting roller, we replaced the zip-ties with rubber bands stretched between two gears on each end of the roller. The rubber bands had more frictional force than the zip-ties, and they worked excellently. We collected several EGGS each match. In order to improve the four bar linkage, we also replaced the stripped gear, and we and programmed a button on the controller to turn off all motors. We were able to avoid stripping the gears during the OED, but the four bar linkage still was not operational. We were able to collect more EGGS than we had expected, which required more torque to lift than the motor and transmission could provide.

Section 8- Appendix

File name: final_project

```
#include <DFW.h> // DFW include
#include <Servo.h>
#include "MyRobot.h"
class DFWRobot: public AbstractDFWRobot {
  public:
    virtual void robotStartup() = 0;
    virtual void autonomous( long, DFW &) = 0;
    virtual void teleop( long, DFW &) = 0;
    virtual void robotShutdown(void) = 0;
    virtual int getDebugLEDPin(void) = 0;
};
int ledpindebug = 13;
int rightSensePin = 22, leftSensePin = 23;
Servo leftServo;
Servo rightServo;
Servo boxServo;
Servo sweepServo;
Servo hangServo;
```

```
const long target = 350;
const long kp = 10;
int current;
long duration;
int distance;
int rightSensorState = 0; //starting an initial state for the right sensor (off)
int leftSensorState = 0;
boolean autoWork = true;

DFW dfw(ledpindebug); // Instantiates the DFW object and setting the debug pin. The debug pin will be set high if no communication is seen after 2 seconds

void setup() {
  Serial.begin(9600); // Serial output begin
  dfw.begin(); // Serial1 output begin for DFW library. Buad and port #."Serial1 only"

  sweepServo.attach(6, 1000, 2000);
  leftServo.attach(5, 1000, 2000);
  rightServo.attach(7, 1000, 2000);
  boxServo.attach(8, 1000, 2000);
  hangServo.attach(4, 1000, 2000);
```

```
  //for the Ultrasonic range module
}

float error () {
  return (current - target);}
void armUp() {
  boxServo.write(80 );
}
void armDown() {
  boxServo.write(100);
}
void goStraight() {
  setPower(180, -180);
}
void goBack() {
  setPower(-100, 50);
}
void setPower(float left, float right) { //Power -100 to 100
  rightServo.write(180 - map(right, -100, 100, 0, 180));
  leftServo.write(map(left, -100, 100, 0, 180));
}
```

```
void extend() { // let the hanger longer
  hangServo.write(90);
}
void shrink() {
  hangServo.write(-90);
}
void loop() {

  dfw.run(); // Called to update the controllers output
  if (dfw.getCompetitionState() != powerup) {
    //clears the trigPin
    //digitalWrite(trigPin, LOW);
    //delayMicroseconds(2);
    //sets the trigPin on HIGH state for 10 micro seconds
    //digitalWrite(trigPin, HIGH);
    //delayMicroseconds(10);
    //digitalWrite(trigPin,LOW);
    // Reads the echoPin, returns the sound wave travel time in microseconds
    //duration = pulseIn(echoPin, HIGH);
    // Calculating the distance
    //distance= duration*0.034/2;
    //if(distance < 3){
```

```
    //leftServo.write(0);
    //rightServo.write(0);
    //}
    rightServo.write(180 - dfw.joystickrv());   //DFW.joystick will return 0-180 as an int into
rightmotor.write
    leftServo.write(dfw.joysticklv());
    if (dfw.start()) {
      if (autoWork == true) {
        teleop;
        autoWork = false;
      } else {
        setPower(90,90);
        autoWork = true;
      }
    }
    if(dfw.up()){
        sweepServo.write(180);//sweep the eggs into box
    }
    if(dfw.down()){
        sweepServo.write(0);//sweep the eggs out of box
    }
    if(dfw.left()){
```

```
      hangServo.write(180);//let hanger longer
    }
    if(dfw.right()){
      hangServo.write(0);//let hanger shorter
    }
    if (dfw.one()) {
      boxServo.write(180);//let the box higher
    }
    if (dfw.two()) {
      boxServo.write(0);//let hanger longer
    }
    if(dfw.three()){
     sweepServo.write(90);
     leftServo.write(90);
     rightServo.write(90);
     boxServo.write(90);
     hangServo.write(90);
    }     }
}
```

File name: MyRobot.cpp

```
#include "MyRobot.h"
#include "Arduino.h"
```

```
/**
 * These are the execution runtions
 */
void MyRobot::initialize(unsigned armMotorPin, unsigned armPotPin) {
  potPin = armPotPin;
  pinMode(armMotorPin, INPUT);
  const int trigPin = 9;
  const int echoPin = 10;
  motor.attach(armMotorPin, 1000, 2000);
  lleftServo.attach(5, 1000, 2000);
  rrightServo.attach(7, 1000, 2000);
  NewPing sonar(trigPin,echoPin,MIN_DISTANCE);
}
void MyRobot::moveTo(unsigned position) {
  motor.write(position);
}
/**
 * Called when the start button is pressed and the robot control begins
 */
 void MyRobot::robotStartup(){
 }
 void MyRobot::setPower2(float left, float right) { //Power -100 to 100
```

```
  rrightServo.write(180 - map(right, -100, 100, 0, 180));
  lleftServo.write(map(left, -100, 100, 0, 180));
}
  void MyRobot::goStraight2() {
  setPower2(100, -50);
}
/**
 * Called by the controller between communication with the wireless controller
 * during autonomous mode
 * @param time the amount of time remaining
 * @param dfw instance of the DFW controller
 */
 void MyRobot::autonomous( long time= 2000){
    Serial.print("\r\nAuto time remaining: ");
    Serial.print(time);
 }
/**
 * Called by the controller between communication with the wireless controller
 * during teleop mode
 * @param time the amount of time remaining
 * @param dfw instance of the DFW controller
 */
```

```
void MyRobot::teleop( long time = 2000){
    float duration, distance;
 const int trigPin = 9;
 const int echoPin = 10;

 digitalWrite(trigPin, LOW);
 delayMicroseconds(2);
 digitalWrite(trigPin, HIGH);
 delayMicroseconds(10);
 digitalWrite(trigPin, LOW);
 distance = (duration / 2) * 0.0344;
 duration = pulseIn(echoPin, HIGH);
   Serial.print(sonar.ping_cm());
   int num = sonar.ping_cm();
   if (distance <= MIN_DISTANCE) {
     setPower2(0, 0);
   }else{
     goStraight2();
   }
 }
/**
 * Called at the end of control to reset the objects for the next start
```

```
 */
 void MyRobot::robotShutdown(void){
    Serial.println("Here is where I shut down my robot code");

 }
```

File name: MyRobot.h

```
#include "MyRobot.h"
#include "Arduino.h"

/**
 * These are the execution funtions
 */

void MyRobot::initialize(unsigned armMotorPin, unsigned armPotPin) {
  potPin = armPotPin;
  pinMode(armMotorPin, INPUT);
  const int trigPin = 9;
  const int echoPin = 10;
  motor.attach(armMotorPin, 1000, 2000);
  lleftServo.attach(5, 1000, 2000);
```

```
 rrightServo.attach(7, 1000, 2000);
 NewPing sonar(trigPin,echoPin,MIN_DISTANCE);
}
void MyRobot::moveTo(unsigned position) {
 motor.write(position);
}
/**
 * Called when the start button is pressed and the robot control begins
 */
 void MyRobot::robotStartup(){
 }
 void MyRobot::setPower2(float left, float right) { //Power -100 to 100
 rrightServo.write(180 - map(right, -100, 100, 0, 180));
 lleftServo.write(map(left, -100, 100, 0, 180));
}
 void MyRobot::goStraight2() {
 setPower2(100, -50);
}
/**
 * Called by the controller between communication with the wireless controller
 * during autonomous mode
 * @param time the amount of time remaining
```

```
 * @param dfw instance of the DFW controller
 */
 void MyRobot::autonomous( long time= 2000){
   Serial.print("\r\nAuto time remaining: ");
   Serial.print(time);
 }
/**
 * Called by the controller between communication with the wireless controller
 * during teleop mode
 * @param time the amount of time remaining
 * @param dfw instance of the DFW controller
 */
 void MyRobot::teleop( long time = 2000){
  float duration, distance;
  const int trigPin = 9;
  const int echoPin = 10;

  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);
```

```
 distance = (duration / 2) * 0.0344;


 duration = pulseIn(echoPin, HIGH);
   Serial.print(sonar.ping_cm());
   int num = sonar.ping_cm();
   if (distance <= MIN_DISTANCE) {
     setPower2(0, 0);
   }else{
     goStraight2();
   }
 }
/**
 * Called at the end of control to reset the objects for the next start
 */
 void MyRobot::robotShutdown(void){
   Serial.println("Here is where I shut down my robot code");
 }
```