RBE 1001 C '17, Introduction to Robotics
Final Project Report

"ROB III"

Team 11

Member	Signature	Contribution (%)
Michael Abadjiev	___Michael_Abadjiev______________	______34______
Cole Flegel	___Cole_Flegel___________________	______33______
Nicholas Johnson	___Nicholas_Johnson______________	______33______

Grading:	Presentation	_____/20
	Design Analysis	_____/30
	Programming	_____/30
	Accomplishment	_____/20
	Total	_____/100

Contents

Figures

Introduction

The task of this project is to evaluate our team's understanding of the basic concepts presented in RBE 1001. These concepts are evaluated by the following: a sensor-based autonomous program, non-trivial power transmission and lifting device, and a custom circuit. We will present these concepts through scoring points in the given challenge, shown in Figure 1, with both autonomous- and radio-controlled behavior. In this challenge, we will simulate tasks such as crossing uneven terrain, such as a ramp, picking up objects from the ground, and transporting and depositing the objects to specific locations. Our robot was created to simulate such tasks on the field in order to analyze how well a similar robot could perform in similar conditions.

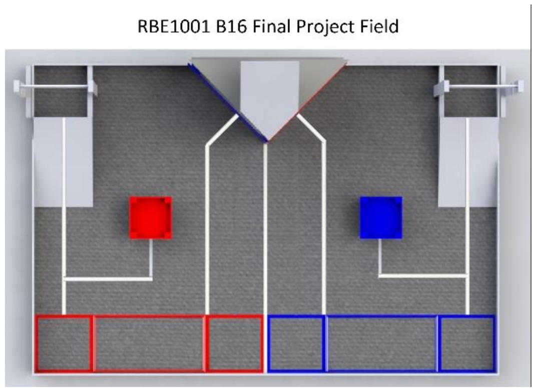


Figure 1

Preliminary Discussion

While analyzing the game, our team identified several unique ways to play. First, in autonomous to try and score EGGs, move the PEN, or drive onto a RAMP positioning needs to occur. Competing strategies are dead reckoning or sensor assisted, either by a line follower or range finding. During tele-op, scoring the EGGs is the main part of the game and there are several ways to do this. A robot could gather many eggs at once, then go and dump them in for a score. A second possible way to collect a few EGGs at a time but move quickly to score. In addition to the amount of eggs to score at a time, there are several scoring locations, the COOP, PEN, and NEST. Each is worth different amount of points, but those worth more are harder to score in. At the end of the game to score additional points a robot can get off the ground, either by driving onto a RAMP or by hanging from the PERCH.

Problem Statement

We determined that our ultimate goal would be to collect as many EGGs as possible and deposit them in the highest scoring location: the NEST. In order to accomplish this, we decided to gather many EGGs at once, then lift them all up and dump them into the NEST. We also decided that a very easy way to score points during the autonomous portion would be to simply push the PEN off of its starting location and then collect the EGGs that are underneath it. The final task that we wanted to accomplish was to lift our robot on the PERCH at the end of the match.

Preliminary Designs

We spent a considerable amount of time brainstorming the design of our robot. In order to lift the EGGs, we considered a linear slide as shown in Figure 2, a lifting arm, 4-bar linkage, or chain and sprocket to lift and dump the EGGs into the NEST. Because we could not get a linear slide working without much friction, we decided on a chain and sprocket in which the bucket will rotate with the sprocket once it reaches the top to dump the eggs. In order to get the EGGs into a bucket from the ground where many of them will be, we had a few options, including simply scooping them up using the walls to push in, or intake rollers attached to either the bucket or the chassis to "brush" the EGGs into the bucket. Using an intake to power them into the bucket seemed like the most efficient option and keeping it attached to the chassis as to keep the weight of the bucket low also seemed like the best idea.

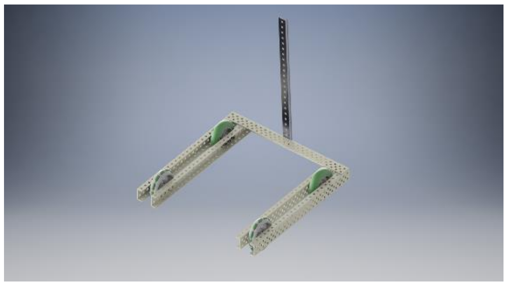


Figure 2

Another important aspect of our design is how it moves around the field. The two obvious choices were two wheel drive or four wheel drive. We also had the choice of omni wheels or regular wheels in different applications. We chose two wheel drive with Omni wheels on the non-driven wheels in order to make turns very easily. As we had limited room on our robot we could not afford the space for more motors to have a four-wheel drive robot. Since most of our weight will be over our rear wheels anyway due to our design we do not have to worry about losing traction on them.

For sensors to be used during autonomous, there were several options. We considered the use of line trackers to follow lines, Quadrature encoders to track wheel rotation, and Infrared or Ultrasonic range finders to determine the location of the robot. We eventually settled on only using Quadrature encoders for movement. We considered a line tracker but had no place to put them except for on the turning center of the robot which does not work well. With encoders we will be able to program distance moves to get where we need to go in autonomous accurately and consistently without the need for an additional distance sensor. Figure 3 shows how we intend to incorporate all of these mechanical aspects into our preliminary design.

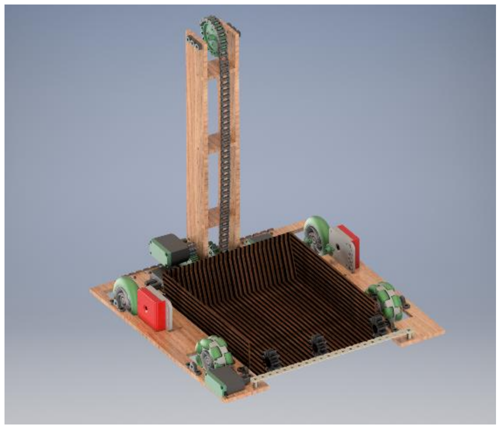


Figure 3

For the custom circuit we considered having a battery display, a robot mode display, or a custom line follow sensor. As mentioned above, the line follower would not be feasible on our robot, and we have no pressing need for a voltage readout, so we chose to use the LCD display to show which autonomous program we have selected. (I.e. which side, which starting location)

Selection of Final Design

After considering tradeoffs and better designs for parts in general, we made many changes to the design of our robot. These changes were made for a variety of reasons, such as better execution of tasks, more stability and strength of the robot, and additions to the design for more tasks to be completed. In our final adjustments to the design of the chassis we planned to laser-cut, we realized that there were quite a few locations, especially in the rear of the robot, in which the amount of material remaining after cutting was very thin and would be prone to breaking. To solve this, we added screw slots to the edges to add metal strips for reinforcement. We also added a metal band across the front to make the chassis less likely to flex and snap the back. When constructing our robot, we found quite a few small issues we didn't notice in our initial designing. We did not add any locations for placement of the battery, Arduino board, and custom circuit board. We solved this by adding a small mount on one of the arms of the main chassis for the battery, and attaching the boards to the side of the uprights used for the lift. Luckily, we were able to use existing screw holes in the chassis and uprights for mounting the battery and Arduino board. We used the existing sticky backing of the breadboard to mount our custom circuit board. When placing the motors and gears into the cutouts made for them in the laser-cut chassis piece, we found that a couple of the holes were slightly misplaced or sized too small. Therefore, we had to increase the size of the holes with a hacksaw in order for them to fit. Even after the robot was built completely, we made more changes. We decided to cut off the front outer corners of the chassis at an angle because the corner would hit the side of the BARN approximately 20% of the time in our testing. With the corners cut off at an angle, we would be much less likely to do so. Our final change was to add a hook 16 inches above our scoop in order to hang on the PERCH at the end of the tele-operated period of the demonstration. We had not planned to do so initially because we had thought it would interfere completely with dumping out the EGGs and our lift wouldn't be able to support the full weight of the

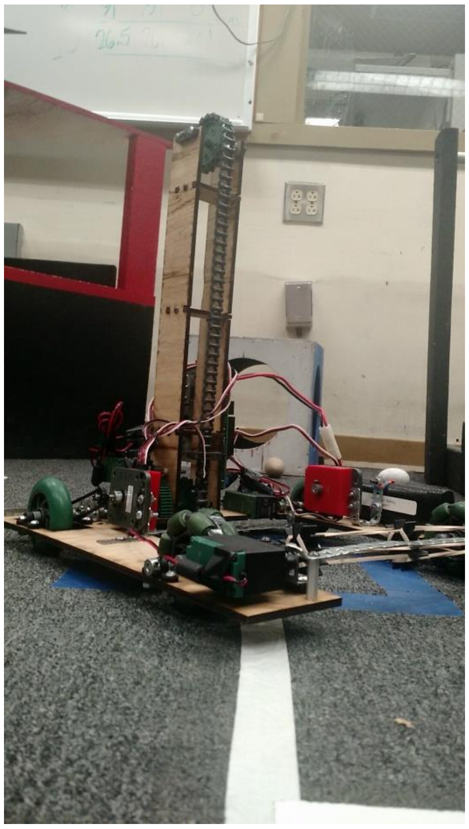


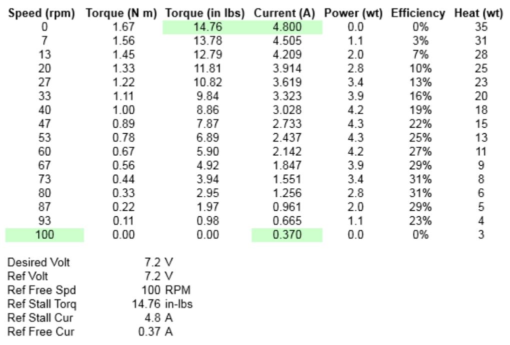
Figure 4: Our final robot

robot in order to hang effectively. However, we found that we only need to dump from approximately 8 inches further away from the BARN and add a larger surface for the back of the scoop for the EGGs to roll down into the NEST. We also found that, since we received a high-strength gear kit and high-strength chain and sprocket kit, our lift could lift the robot with no issue of back-driving the gears

Final Design Analysis

For this robot, we used functional programming, using several reusable functions. This allowed us to execute very complex moves by passing parameters to a sequence of simpler moves. We are not able to pause the execution of our code at any point because of the time for match length. Because of this, we used state programming to execute a sequential series of moves without having to stop the program. We also used a LCD display to display which autonomous mode we are running in.

On our robot we utilized two encoders, one for each of the driven wheels. Using these encoders to track wheel rotation, coupled with knowing the wheel circumference allows us to track with reasonable accuracy where on the field the robot is at all times.

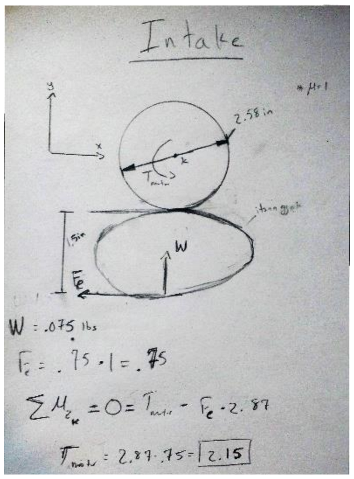


Speed (rpm)	Torque (N m)	Torque (in lbs)	Current (A)	Power (wt)	Efficiency	Heat (wt)
0	1.67	14.76	4.800	0.0	0%	35
7	1.56	13.78	4.505	1.1	3%	31
13	1.45	12.79	4.209	2.0	7%	28
20	1.33	11.81	3.914	2.8	10%	25
27	1.22	10.82	3.619	3.4	13%	23
33	1.11	9.84	3.323	3.9	16%	20
40	1.00	8.86	3.028	4.2	19%	18
47	0.89	7.87	2.733	4.3	22%	15
53	0.78	6.89	2.437	4.3	25%	13
60	0.67	5.90	2.142	4.2	27%	11
67	0.56	4.92	1.847	3.9	29%	9
73	0.44	3.94	1.551	3.4	31%	8
80	0.33	2.95	1.256	2.8	31%	6
87	0.22	1.97	0.961	2.0	29%	5
93	0.11	0.98	0.665	1.1	23%	4
100	0.00	0.00	0.370	0.0	0%	3

Desired Volt	7.2	V
Ref Volt	7.2	V
Ref Free Spd	100	RPM
Ref Stall Torq	14.76	in-lbs
Ref Stall Cur	4.8	A
Ref Free Cur	0.37	A

To ensure that the motors our robot uses would not be overpowered we did a few calculations to determine the load on the intake motor, the drive motors, and the lift motor. We used the motor specifications sheet shown in Figure 5 to determine that it would be best to run our motors below 1.5A in order to ensure maximum efficiency, as well as reduce the risk of overheating.

Figure 5



$W = .075$ lbs

$F_e = .75 \cdot 1 = .75$

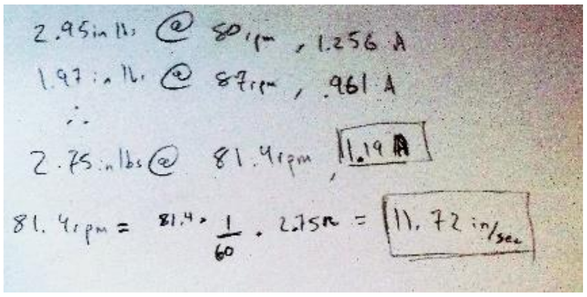
$\sum M_{z_k} = 0 = T_{motor} - F_e \cdot 2.87$

$T_{motor} = 2.87 \cdot .75 = \boxed{2.15}$

Figure 6

For the intake, we calculated the torque that the motor would have to exert by looking at the problem as it having to slide the egg along the ground as seen in Figure 6. This gave us a max torque on the motor of 2.15 in-lbs, which is well within our range.

To calculate the load on the drive motors, we assumed that 2/3 of the weight of the robot would be over the driven wheels. The calculations in Figures 7 and 8 show that the motor will need to exert 2.75 in-lbs. This translates to 1.19A, which is within our limit, and that the robot will move at 11.72 in/sec which is fast enough for us.



2.95 in lbs @ 80 rpm, 1.256 A

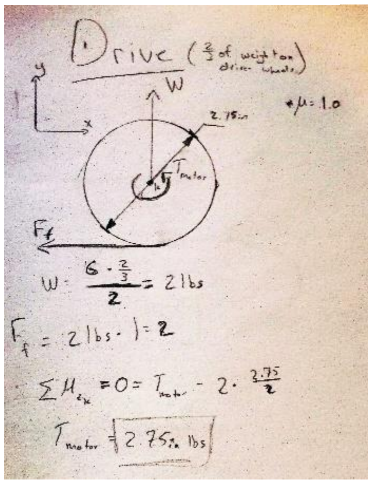
1.97 in lbs @ 87 rpm, .961 A

$\therefore$

2.75 in lbs @ 81.4 rpm, $\boxed{1.19 \text{ A}}$

$81.4 \text{ rpm} = 81.4 \cdot \frac{1}{60} \cdot 2.75\pi = \boxed{11.72 \text{ in/sec}}$

Figure 8



$W = \frac{6 \cdot \frac{2}{3}}{2} = 2$ lbs

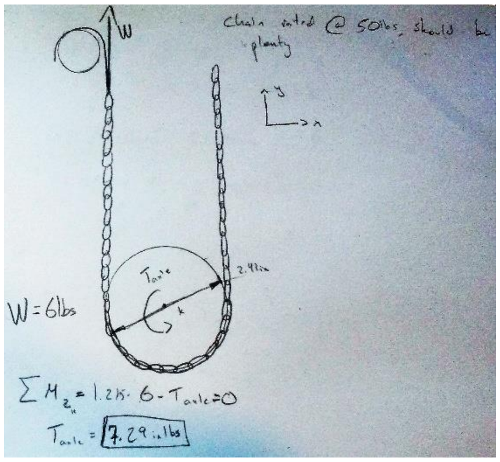
$F_f = 2 \text{ lbs} \cdot 1 = 2$

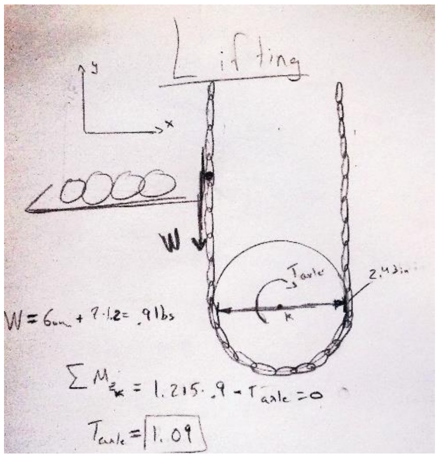
$\sum M_{z_k} = 0 = T_{motor} - 2 \cdot \frac{2.75}{2}$

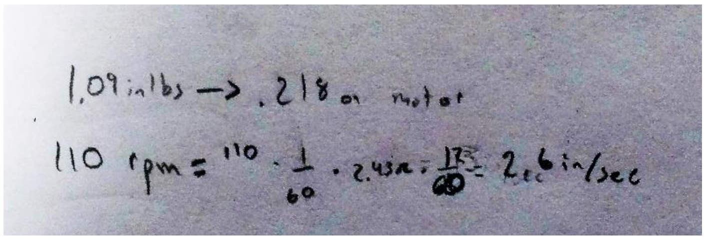
$T_{motor} = \boxed{2.75 \text{ in lbs}}$

Figure 7

The most important calculations we did were for the lifting mechanism. Because we planned on lifting not just EGGs with our bucket, but also the whole robot at the end of the match and it is safe to assume that the bucket weighs less than the entire robot, we only did torque requirement calculations using the weight of the whole robot, which we measured to be 6lbs. The first consideration was the chain strength, however vex high strength chain is rated for 50 lbs, so we were all set. Next we considered motor load. The calculations in Figure 10 show that the sprocket would need 7.29 in-lbs to lift the robot. While this is doable with the vex 393s, we chose to use a 12:60 gearing in order to keep the motor running below 1.5A. Using the calculations shown in Figures 9 and 11, we determined that with this gearing the motor have no trouble at all lifting the bucket, needing to exert only 0.218 in-lbs of torque. In this scenario, the bucket lifts at 2.6 in/sec which we determined was a controllable speed.


W
Chain rated @ 50lbs, should be plenty
y
x
T_{axle}
2.43in
W=6lbs
k
$\sum M_{z_k} = 1.215 \cdot 6 - T_{axle} = 0$
$T_{axle} = \boxed{7.29 \text{ in lbs}}$
Figure 10


Lifting
y
x
W
T_{axle}
2.43in
W = 6 + 7·1.2 = .9lbs
k
$\sum M_{z_k} = 1.215 \cdot .9 - T_{axle} = 0$
$T_{axle} = \boxed{1.09}$
Figure 11


1.09 in lbs → .218 on motor
$110 \text{ rpm} = \frac{110}{} \cdot \frac{1}{60} \cdot 2.43\pi = \frac{175}{60} = 2.6 \text{ in/sec}$
Figure 9

Summary/Evaluation

During our evaluation in the lab period, our robot performed spectacularly. It scored nearly all of the points that we wanted it to score during the autonomous section, collecting EGGs from under the PEN while moving the PEN off of its starting box, then proceeding to score all the EGGs in the NEST. This was only one of multiple autonomous modes that we could select by moving jumper cables between different pins on the arduino shield. Our custom circuit, an LCD display, whose wiring diagram can be seen in Figure 12, which we forgot to mention during the demo, displayed which mode was selected. Our only complaint about the autonomous portion of our demonstration was that the robot did not succeed in actually getting a few of the EGGs into its bucket, costing us a few points. During the driver controlled portion we succeeded in dumping two more bucket-fulls of EGGs into the NEST, and, at the end, managed to hang off of the PERCH.

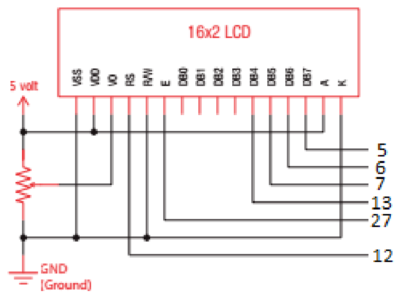


Figure 12

Despite how well our robot ended up working, there were still a few things we would have liked to have done better. First and foremost, from a mechanical perspective, we would like to have improved out bucket slightly. We could have added an acrylic strip behind the chain, to act as a rail that would keep the bucket from flopping down as much. We also think it would have been good to go with a design with slightly more ground clearance, as the gear that turned our lift was very near to the ground and would bottom out when we tried to go up the ramp, necessitating a complex maneuver to get the robot up the ramp. From an electrical and coding standpoint, it would have been nice to have a button to use to cycle through our autonomous modes, as well as a toggle-able intake. This would have made driving easier, allowing us to potentially score more points.

Appendix

```
#include <DFW.h> // DFW include
#include <Servo.h> // servo library
#define ENCODER_OPTIMIZE_INTERRUPTS //optimize interrupts
#include <Encoder.h> //Encoder library
#include <LiquidCrystal.h> //lcd library


int ledpindebug = 13; //Wireless controller Debug pin. If lit then there is no communication.

int autoState = 0; //autoState var

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
Servo rightmotor; // Servo object
Servo leftmotor; // Servo object
Servo liftmotor; //Servo object
Servo intakemotor; //Servo object

Encoder leftE(2,25);//2,3
Encoder rightE(3,26);//18,19

LiquidCrystal lcd(12,27,13,7,6,5); //pinout for lcd

void setup() {
  Serial.begin(9600); // Serial output begin. Only needed for debug
  dfw.begin(9600, 1); // Serial1 output begin for DFW library. Buad and port #."Serial1 only"

  intakemotor.attach(11, 1000, 2000); //intakemotor pin declaration
  liftmotor.attach(8, 1000, 2000); //liftmotor pin declaration
  leftmotor.attach(9, 1000, 2000); // left drive motor pin#, pulse time for 0,pulse time for 180
  rightmotor.attach(10, 1000, 2000); // right drive motor pin#, pulse time for 0,pulse time for 180
  //Serial.println("start");

  lcd.begin(16, 2); //write wait for connect
  lcd.print("Wait");

  while (dfw.start() == 0) { //waits for controller to init
    Serial.println("init");
    dfw.update();
    delay(20);
  }
  // put your setup code here, to run once:
  //default lcd display/mode
  lcd.clear();
  lcd.print("Auto Mode:");
  lcd.setCursor(0,1);
  lcd.print("Red Goal");

  pinMode(28, INPUT_PULLUP); //pin for jumper for auto selection
  pinMode(29, INPUT_PULLUP); //pin for jumper for auto selection
}

void autonomous(volatile unsigned long time) // function definition
{

  while (dfw.start() == 1) { // waits for start button
    Serial.println("waiting for start");
    dfw.update();

    //select and update autoState based on placement of jumpers
    if((digitalRead(28) == 1) && (digitalRead(29) == 1)){
      if(autoState != 0){ //if not already this mode change to this mode
        lcd.clear();
        autoState = 0;
        lcd.print("Auto Mode:");
        lcd.setCursor(0,1);
        lcd.print("Red Goal");
      }
    }
    else if ((digitalRead(28) == 0) && (digitalRead(29) == 1)){
      if(autoState != 18){ //if not already this mode change to this mode
        lcd.clear();
        autoState = 18;
        lcd.print("Auto Mode:");
        lcd.setCursor(0,1);
        lcd.print("Blue Goal");
      }
    }
    else if ((digitalRead(28) == 1) && (digitalRead(29) == 0)){
      if(autoState != 38){ //if not already this mode change to this mode
        lcd.clear();
        autoState = 38;
        lcd.print("Auto Mode:");
        lcd.setCursor(0,1);
        lcd.print("Ramp");
      }
    }
    else if ((digitalRead(28) == 0) && (digitalRead(29) == 0)){
      if(autoState != 41){ //if not already this mode change to this mode
        lcd.clear();
        autoState = 41;
        lcd.print("Auto Mode:");
        lcd.setCursor(0,1);
        lcd.print("Stop");
      }
    }

    delay(20);
  }

  unsigned long startTime = millis(); // sets start time of autonomous
  time = time * 1000; // modifies milliseconds to seconds
  while ((millis() - startTime <= time) && (dfw.select())) // compares start time to time entered in
the autonomous function and
  {
    // The select button can be used to skip the autonomous code.
    // Enter Autonomous User Code Here

    switch(autoState){

      //red goal
      case 0: driveE(20); //drive forward 20 in
              autoState++;
              break;
      case 1: turnLeft(89); //turn left 89 degrees
              autoState++;
              break;
      case 2: driveE(19); //drive forward 19 in
              autoState++;
              break;
      case 3: turnRight(105); //turn right 105 degrees
              autoState++;
              break;
      case 4: intakeOn(); //turn intake on
              driveE(24); //drive forward 24 in
              autoState++;
              break;
      case 5: driveER(5); //drive backwards 5 in
              autoState++;
              break;
      case 6: turnRight(90); //turn right 90 degrees
              autoState++;
              break;
      case 7: driveE(24); //drive forward 24 in
              autoState++;
              break;
      case 8: liftUp(); //start lift going up
              turnLeft(44); //turn left 44 degrees
              autoState++;
              break;
      case 9: driveE(48); //drive forward 48 in
              autoState++;
              break;
```

```
 case 10: intakeOff(); //turn intake off
         turnRight(90); //turn right 90 degrees
         autoState++;
         break;
 case 11:delay(1800); //wait for 1.8 sec for bucket to reach the top
         autoState++;
         break;
 case 12:liftStop(); //stop the bucket
         autoState++;
         break;
 case 13:driveER(2); //drive backwards 2 in
         autoState++;
         break;
case 14: delay(500); //wait 0.5 sec
         autoState++;
         break;
case 15: driveE(2); //drive forward 2 in
         autoState++;
         break;
case 16: liftDown(); //run lift down
         autoState++;
         break;
case 17: delay(500); //wait 0.5 sec
         autoState++;
         break;
case 18: liftStop(); //stop lift
         autoState = 41; //enter wait for auto end
         break;

         //blue goal
 case 19: driveE(20); //drive forward 20 in
          autoState++;
          break;
 case 20: turnLeft(88); //turn left 88 degrees
          autoState++;
          break;
 case 21: driveE(20); //drive forward 20 in
          autoState++;
          break;
 case 22: turnRight(96); //turn right 96 degrees
          autoState++;
          break;
 case 23: intakeOn(); //turn intake on
          driveE(24); //drive forward 24 in
          autoState++;
          break;
 case 24: driveER(5); //drive backwards 5 in
          autoState++;
          break;
 case 25: turnLeft(93); //turn left 93 degrees
```

```
        autoState++;
        break;
case 26: driveE(24); //drive forward 24 in
        autoState++;
        break;
case 27: liftUp(); //start lift up
        turnRight(44); //turn right 44 degrees
        autoState++;
        break;
case 28: driveE(46); //drive forward 46 in
        autoState++;
        break;
case 29: intakeOff(); //turn intake off
        turnLeft(90); //turn left 90 degrees
        autoState++;
        break;
case 30: delay(1800); //wait 1.8 sec for the bucket to reach the top
        autoState++;
        break;
case 31: liftStop(); //stop lift
        autoState++;
        break;
case 32: driveER(2); //drive backwards 2 in
        autoState++;
        break;
case 33: delay(500); //wait 0.5 sec
        autoState++;
        break;
case 34: driveE(2); //drive forward 2 in
        autoState++;
        break;
case 35: liftDown(); //run lift down
        autoState++;
        break;
case 36: delay(500); //wait 0.5 sec
        autoState++;
        break;
case 37: liftStop(); //stop lift
        autoState = 41; //enter wait for auto end
        break;

      //ramp
case 38: backupRamp(35); //backup 35 in
        autoState++;
        break;
case 39: liftUp(); //start lift up to assist ramp climb
        backupRamp(20);//backup 20 in
        autoState++;
        break;
case 40: liftStop(); //stop lift
```

```
            turnLeft(150); //turn left 150 degrees
            driveE(2); //drive forward 2 in
            autoState = 41; //enter wait for auto end
            break;

            //Wait
      case 41: break; //wait for auto end
        }

    Serial.println("Autonomous"); //prints Autonomous over serial (usb com port)

    dfw.update();//used for autonoumous skip
    delay(20); //delay to prevent spamming the serial port and to keep servo and dfw libraries
happy
  }
}

void teleop(unsigned long time) { // function definition
  unsigned long startTime2 = millis(); // sets start time of teleop
  time = time * 1000; // modifies milliseconds to seconds
  lcd.clear();
  while (millis() - startTime2 <= time) // compares start time to time entered in the teleop function
  {
    //tank drive code next 4 lines
    dfw.update();// Called to update the controllers output. Do not call faster than every 15ms.
    rightmotor.write(180 - dfw.joystickrv()); //invert joystick for rightmotor
    leftmotor.write(((dfw.joysticklv() - 90) * (2.0/3.0)) + 90);  //scale leftmotor to right motor's
output
    delay(20); // Servos do not like to be called faster than every 20 ms. Also the Serial data is
sent every 15ms

    // Enter Teleop User Code Here

    if(dfw.l1() == 0){ //turn intake on
      intakeOn();
    }
    if(dfw.l2() == 0){ //reverse intake
      intakeReverse();
    }
    if(((dfw.l1() == 1) && (dfw.l2() == 1)) || ((dfw.l1() == 0) && (dfw.l2() == 0))){ //if both pressed or
none are pressed turn intake off
      intakeOff();
    }
    if(dfw.r1() == 0){ //lift up
      liftUp();
    }
    if(dfw.r2() == 0){ //lift down
      liftDown();
    }
    if(((dfw.r1() == 1) && (dfw.r2() == 1)) || ((dfw.r1() == 0) && (dfw.r2() == 0))){ //lift stop if not
going up or down, or both are pressed
      liftStop();
    }

    Serial.println("TeleOp"); //prints Teleop over serial (usb com port)
    delay(20); //delay to prevent spamming the serial port

  }
  exit(0); // exits program
}


void loop() {

  autonomous(20); //time in seconds to run autonomous code

  teleop(180); //time in seconds that teleop code will run
}


void driveE(int inches){
  leftE.write(0); //reset encoders
  rightE.write(0);
  //360 ppr
  //Good pins: 2,3,18,19,20,21
  int pulsesNeeded = (inches/8.5) * 360.0; //calc number of pules to go x inches;(number of
inches to go)/(approximate wheel diameter) * ppr; 8.639
  while(leftE.read() < pulsesNeeded || rightE.read() > -pulsesNeeded){ //while not at target keep
driving
    if(leftE.read() < pulsesNeeded){ //if left is short of target drive else stop
      leftmotor.write(160);
    }
    else leftmotor.write(90);
    if(rightE.read() > -pulsesNeeded){ //if right is short of target drive else stop
      rightmotor.write(0);
    }
    else rightmotor.write(90);
  }
  leftmotor.write(90); //stop both motors after target is reached
  rightmotor.write(90);
}
void driveER(int inches){
  leftE.write(0); //reset encoders
  rightE.write(0);
  //360 ppr
  //Good pins: 2,3,18,19,20,21
  int pulsesNeeded = (inches/8.5) * 360.0; //calc number of pulses to go x incher; (number of
inches to go)/(approximate wheel diameter) * ppr;  8.639
```

```
  while(leftE.read() > -pulsesNeeded || rightE.read() < pulsesNeeded){//while not at target keep
driving
    if(leftE.read() > -pulsesNeeded){ //if left is short of target drive else stop; not at full speed as
to drive straight
      leftmotor.write(30);
    }
    else leftmotor.write(90);
    if(rightE.read() < pulsesNeeded){ //if right is short of target drive else stop
      rightmotor.write(180);
    }
    else rightmotor.write(90);
  }
  leftmotor.write(90); //stop both motors after target is reached
  rightmotor.write(90);
}
void turnLeft(int degrees){ //turn robot left for x degrees
  leftE.write(0); //reset encoders
  rightE.write(0);
  int pulsesNeeded = ((((degrees/360.0) * 69.115)/8.3) * 360)/2.0; //calc number of pulses to
rotate x degrees
  while(leftE.read() > -pulsesNeeded || rightE.read() > -pulsesNeeded){ // while not rotated far
enough continue rotating
    if(leftE.read() > -pulsesNeeded){ // if left is short of target keep going else stop
      leftmotor.write(24);
    }
    else leftmotor.write(90);
    if(rightE.read() > -pulsesNeeded){ //if right is short of target keep going else stop
      rightmotor.write(0);
    }
    else rightmotor.write(90);
  }
  leftmotor.write(90);// stop motors after rotation is reached
  rightmotor.write(90);
}
void turnRight(int degrees){
  leftE.write(0); //reset encoders
  rightE.write(0);
  int pulsesNeeded = ((((degrees/360.0) * 69.115)/8.3) * 360)/2.0; //calc number of pulses to
rotate x degrees
  Serial.println(pulsesNeeded);
  while(leftE.read() < pulsesNeeded || rightE.read() < pulsesNeeded){ // while not rotated far
enough continue rotating
    if(leftE.read() < pulsesNeeded){ // if left is short of target keep going else stop
      leftmotor.write(156);

    }
    else leftmotor.write(90);
    if(rightE.read() < pulsesNeeded){ //if right is short of target keep going else stop
      rightmotor.write(180);
      Serial.println(rightE.read());
    }
    else rightmotor.write(90);
  }
  brakeMotors(1);
}
void intakeOn(){ //turn intakemotor on at full speed forward
  intakemotor.write(180);
}
void intakeOff(){ //turn intakemotor off
  intakemotor.write(90);
}
void intakeReverse(){ //turn intakemotor on at full speed reverse
  intakemotor.write(0);
}
void liftUp(){ //lift bucket up at full speed
  liftmotor.write(180);
}
void liftDown(){ //put bucket down at full speed
  liftmotor.write(0);
}
void liftStop(){ //stop lift
  liftmotor.write(90);
}
void backupRamp(int inches){ //backup ramp with chaged speeds compared to normal backing
up
  leftE.write(0);
  rightE.write(0);
  //360 ppr
  //Good pins: 2,3,18,19,20,21
  int pulsesNeeded = (inches/8.5) * 360.0; //encoder pulse calculation
  while(leftE.read() > -pulsesNeeded || rightE.read() < pulsesNeeded){ // while not rotated far
enough continue rotating
    if(leftE.read() > -pulsesNeeded){ // if left is short of target keep going else stop
      leftmotor.write(20);
    }
    else leftmotor.write(90);
    if(rightE.read() < pulsesNeeded){ //if right is short of target keep going else stop
      rightmotor.write(180);
    }
    else rightmotor.write(90);
  }
  leftmotor.write(90);
  rightmotor.write(90);
}
void brakeMotors(int direction){ //run motors reversed to brake after a turn
  if(direction == 1){ //brake if given 1
    leftmotor.write(80);
    rightmotor.write(80);
    delay(100);
    leftmotor.write(90);
    rightmotor.write(90);
  }
  if(direction == 0){ //brake if given 0
    leftmotor.write(100);
    rightmotor.write(100);
    delay(100);
    leftmotor.write(90);
    rightmotor.write(90);
  }
}
```