RBE 1001 A ’17, Introduction to Robotics
Final Project

“Lifty McLifterFace”
Team 2

Member	Signature	Contribution
Arjun Gandhi	_____	____45%_____
Amrit Parmanand	_____	____40%_____
Matt Ferreira	_____	____15%_____

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

TABLE OF CONTENTS

TABLE OF FIGURES

INTRODUCTION

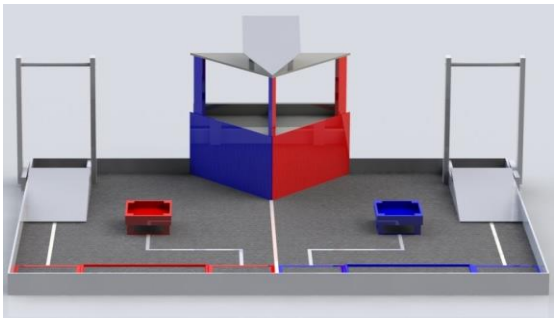


Figure 1: Picture of the Field

The final lab tasked our team with constructing a robot that is both autonomous and controlled via remote and is capable of competing in each complex scenario. Requirements with regards to the robot include: sensor-based autonomy, a non-trivial transmission, a custom circuit, and a lifting device that can affect the outcome challenge. The field is setup up so that two teams will be operating at the same time throughout the two minute twenty second match. Each robot starts in the pre-designated start zone and is given one of two COOPs (Coalitions Ordinary Object Plane). Each COOP contains a PEN (Portable EGG Nest) which is an elevated container that can be moved by the robot. In the center of the field towards the back is the BARN (Big Assembly Retaining NESTs) which holds each team's NEST (Nested Elevated Scoring Triangle). These two containers are elevated fifteen inches off the ground. On either side of the field there is a RAMP (Ramp Assisting Movement to PERCH) and a PERCH (Pretty Elevated Robot Communal Hanger). Fifty EGGs (EGGcellent Game Gizmo) will be placed throughout the fields and will be used for scoring. The value of each egg is dependent upon where and when it is placed by our team's robot. Other contributors to the score are whether are not the robot can be supported by both the RAMP and the PERCH. In the first twenty seconds of the match the robot must function autonomously. Then in the next minute one teammate will take control, then in the following minute another teammate will operate. Within the first two matches, it is required that all group members take control of the robot. The remainder of this report discusses the manners in which

our group tackled this complex challenge including: sketches, calculations, priority characteristics, and how we decided upon our final design.

PRELIMINARY DISCUSSION

After going through the rules and procedures for the game. We broke down the game into 3 main scoring procedures, Autonomous, TeleOp, and End Game. We immediately realized that a combination of these zones would be necessary for success in the game. We then proceeded to construct scoring sheet outlining the point values shown in Figure 3 to the left. After creating a scoring sheet, we constructed a chart of possible combinations of strategies, and assigned a difficulty to each strategy shown in Figure 2 below. Then by dividing total points by difficulty, we assigned a cost benefit ratio to each strategy, as shown in Figure 2. It is obvious that Strategy 5 would be the ideal strategy. The strategy consists of moving the pen, collecting eggs, and dumping them into the high goal during autonomous, while focusing on the collection of eggs during TeleOp and hanging from the perch in the end game.

Autonomus							
Location	Pre-Loaded Eggs	Eggs on floor	Points from Eggs	Points from Moving Pen	Combined Points	Points from Climbing Ramp	Max Points
COOP	1	2	12	5	17	5	22
PEN	3	6	36	5	41	5	46
NEST	6	12	72	5	77	5	82
TeleOP							
Location	Eggs Scored	Points per Egg	Points from Eggs				
COOP	0	1	0				
PEN	0	3	0				
NEST	10	6	60				
		Total Points	60				
End Game							
	Total Points						
Off Carpet	5						
Hanging by Perch	35						

Figure 3: Scoring Sheet

Assume all Strategies have 10 eggs in the scoring zone			
Strategy	1	2	3
Auto	PreLoaded eggs in the COOP	Move Pen and place preloaded eggs in it	Move pen collect floor eggs and put all in pen
Teleop	Push eggs on the floor into COOP	Put eggs in pen	Put eggs in pen
End game	Drive on Ramp	Drive on Ramp	climb
Point total	19	52	106
Difficulty value	3	6	10
Point Total/Difficulty	6.333333333	8.666666667	10.6

4	5	6
Move Pen dump preload eggs in Nest	Move Pen dump collected eggs in Nest	Move Pen dump collected eggs in Nest And climb ramp
Put eggs in Nest	Put eggs in Nest	Put eggs in Nest
Climb	Climb	Climb
124	172	177
12	13	17
10.33333333	13.23076923	10.41176471

Figure 2: Strategy Sheet

PROBLEM STATEMENT

After receiving and analyzing the challenge, the team created a list of prioritized objectives for the robot. It is as follows:

1. Consistent system for dumping eggs into the NEST
2. Being able to hang at the end of the match
3. Hanging system integrated into the dumper
4. Consistent intake to create large loads of eggs to dump.
5. Be able to follow the lines in autonomous to dump eggs into and push the PEN out of its initial position.
6. Use bump switches in order to sense when pressed against either the PEN or BARN
7. Have enough power to consistently lift large loads of eggs high enough to dump into the BARN, and lift our entire robot off of the ground.
8. Have high maneuverability in the drivetrain

Our initial design makes this possible in a streamlined and simple way. The use of multiple line-following sensors will make our autonomous objectives very easy to implement. However, possible design flaws will likely affect being able to fill up our bin with a large amount of eggs due to the way that they are being loaded into the bottom. The use of the four bar linkage for moving the bin with eggs up into the NEST, while simplifying our design, may limit how high the bin can be lifted.

PREMLIMINARY DESIGNS

After our Preliminary Design review, it was evident that our design needed to be simplified. Therefore, we compromised with the design seen in Figure 4. The design consists of center wheel drive with encoder feedback, a line tracker array for PID line following, bump sensors for wall location and accuracy, a 4-bar egg storage lift with PID pot control for accurate positioning, a custom red blue changer circuit, and an expanding lifting device from the bottom of the 4-bar lift.

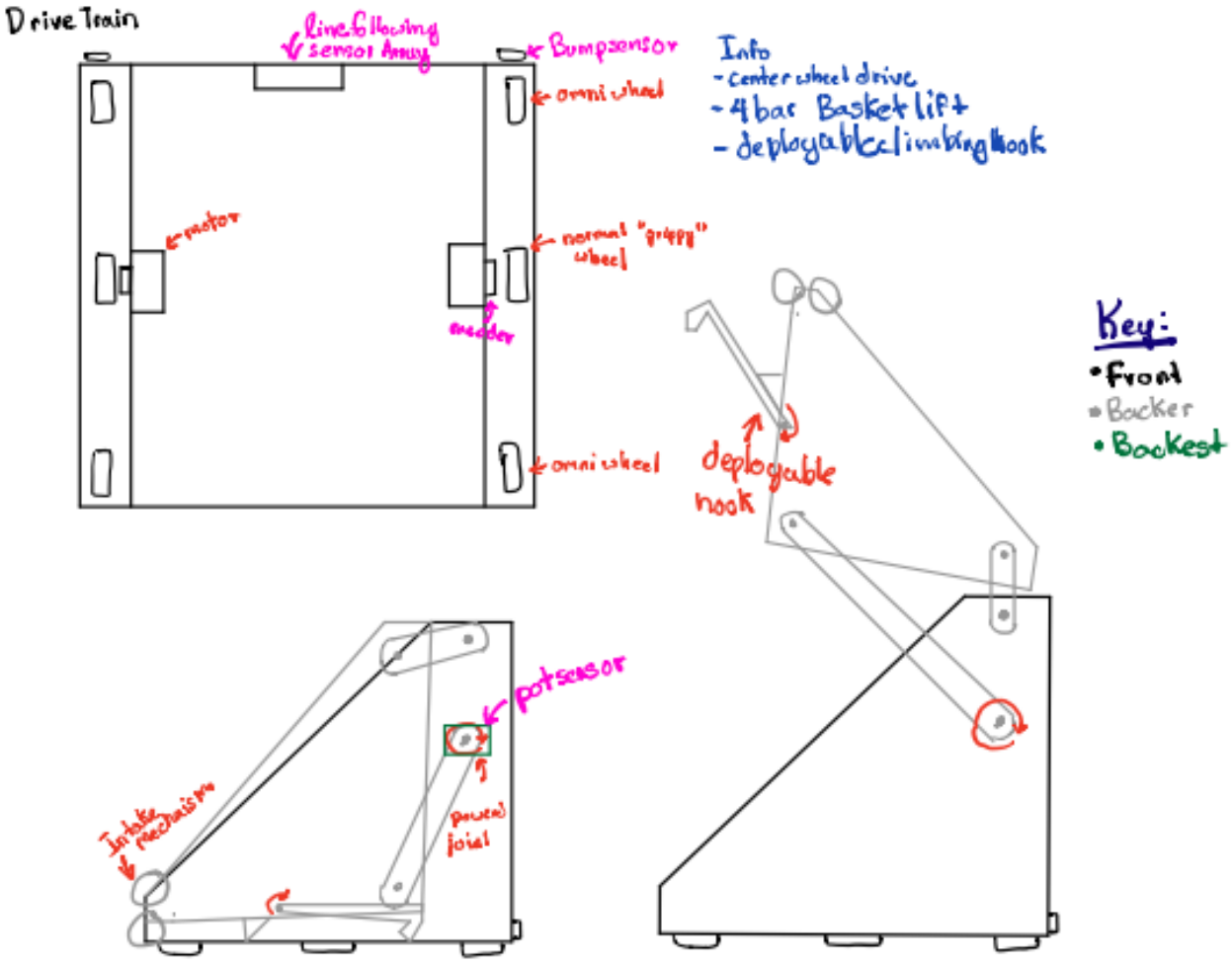


Figure 4: Robot Sketch

SELCTION OF FINAL DESIGN

The design of our robot has changed dramatically from its inception to its final model. The wheel design alone went through three very different iterations, starting at first with a four cornered omni-directional drive train, to a six wheel design, to a more conventional four wheeled back driven layout. The omni wheels will still be used in order to reduce sliding friction when the robot is turning. The motors and major electronics will be placed above the rear wheels to allow for more traction between the ground and the driving wheels. The parts for constructing the robot will be laser cut from quarter inch board. The EGG Basket is operated by a four-bar linkage and contains retractable hooks that will allow the robot to lift itself on the PERCH. The EGG dumping functionality is powered directly by a motor connected to the four bar linkage, A simple mechanical hard stop will prevent back driving of the 4 bar motors while the robot is hanging. Our simple circuit consists of a three-way switch that lets the robot's autonomous functions know whether it is starting on the red or blue side, and lights up the appropriately colored LED.

DESIGN ANALYSIS

MECHANICAL ANAYLSIS

WHEEL BASE

After observing the performance of the base bot, we decided that the performance of the base bot was satisfactory for our needs. The base bot drive train was satisfactory for our needs.

Calculations for the linear speed of our robot is shown below in Figure 5. Due to an unknown weight. We assume the robot weight at a max of 10lb. From previous labs coefficient of friction assumed to be 1.0. As shown speed of the robot is estimated to be 7 in/s.

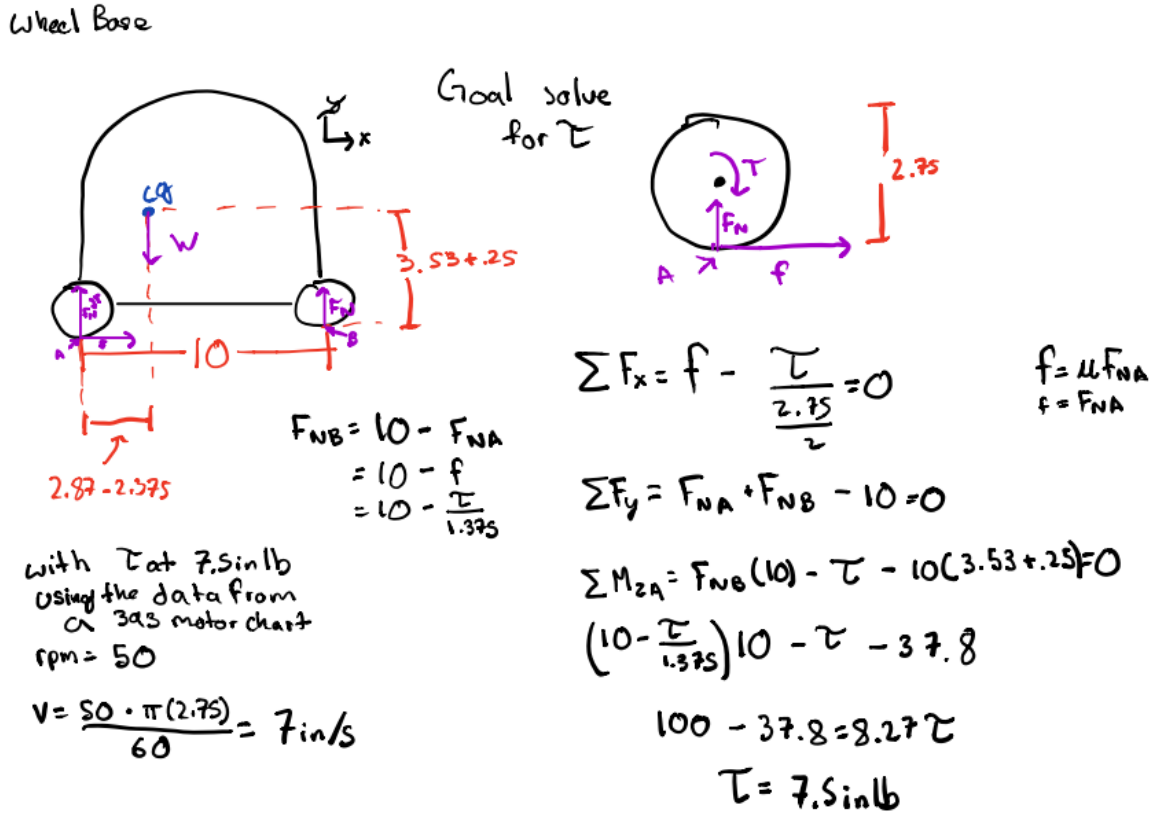
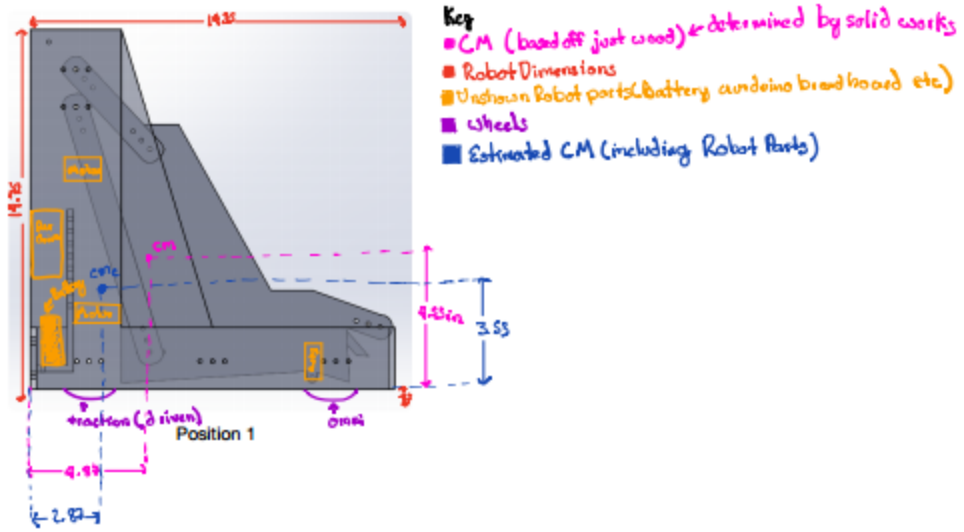


Figure 5: Drive Train Calculations

CENTER OF MASS

As shown in the Figure 6 estimated center of mass for the robot in both positions of the four-bar are relatively inline in both x and y and thus tipping is not a worry even with an extended arm.



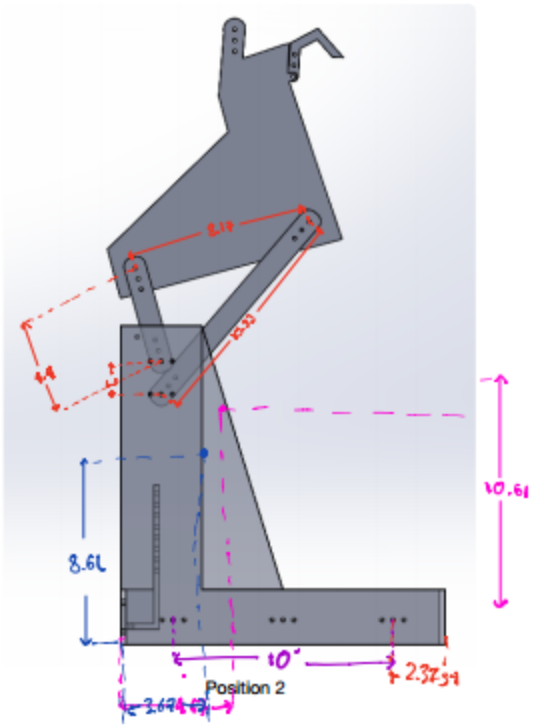


Figure 6: Center of Mass

Figure 6: Center of Mass

RAMP

After looking at the location of our center of mass it is immediately clear that the robot would tip if we would attempt to climb up the ramp forward. We know this as the weight vector would be

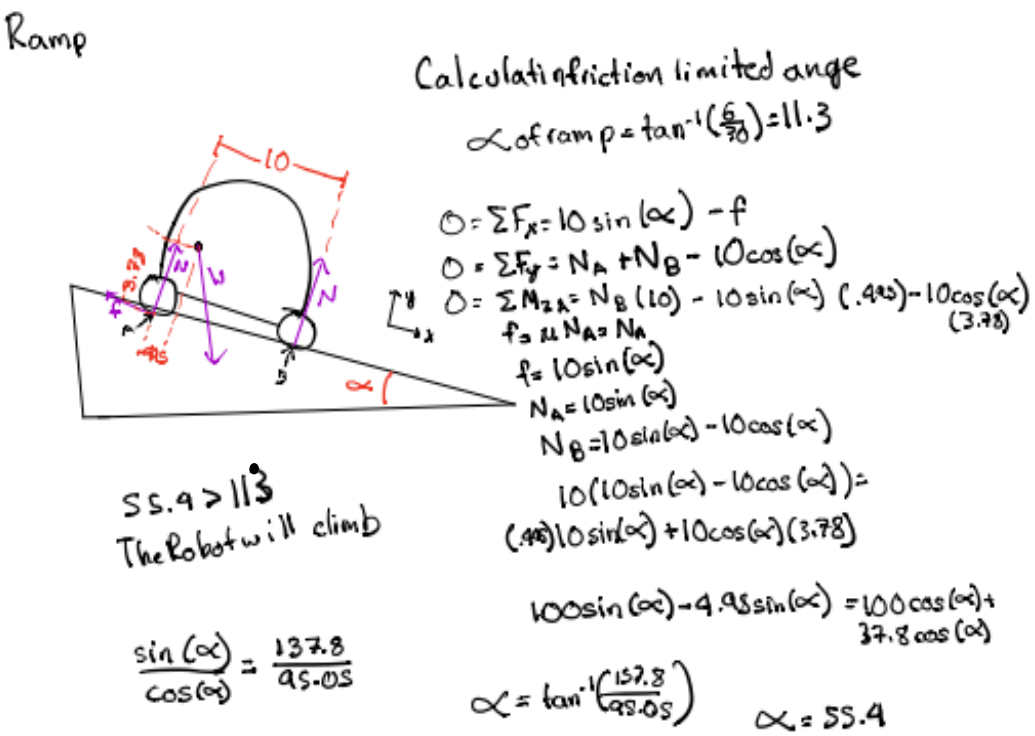
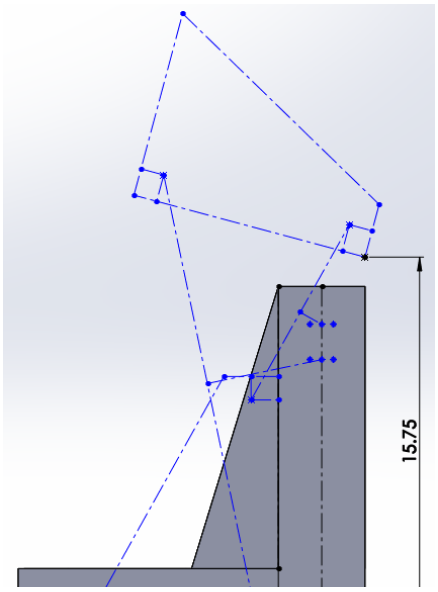


Figure 7: Ramp Calculations

the rear wheel to generate traction is shown in Figure 7.

outside the wheel base. Therefore, we determined it would be best to drive up the ramp backwards thus making it more difficult for the robot to tip. Calculations proving the robot will still have enough force on

FOUR-BAR



The four-bar mechanism was designed in solid works and modeled onto the robot as shown in Figure 8. The four-bar mechanism is designed to hold about 15 eggs. The eggs were found to weigh 1.2 lb. using a scale. The target speed for the arm was 5 rpm and the estimated weight of the empty arm is around 2 lb. with a center of mass located about 2 inches into the basket. As shown in Figure 9 approximately 1.57 W are needed to perform the lift.

Figure 8: Four Bar Design

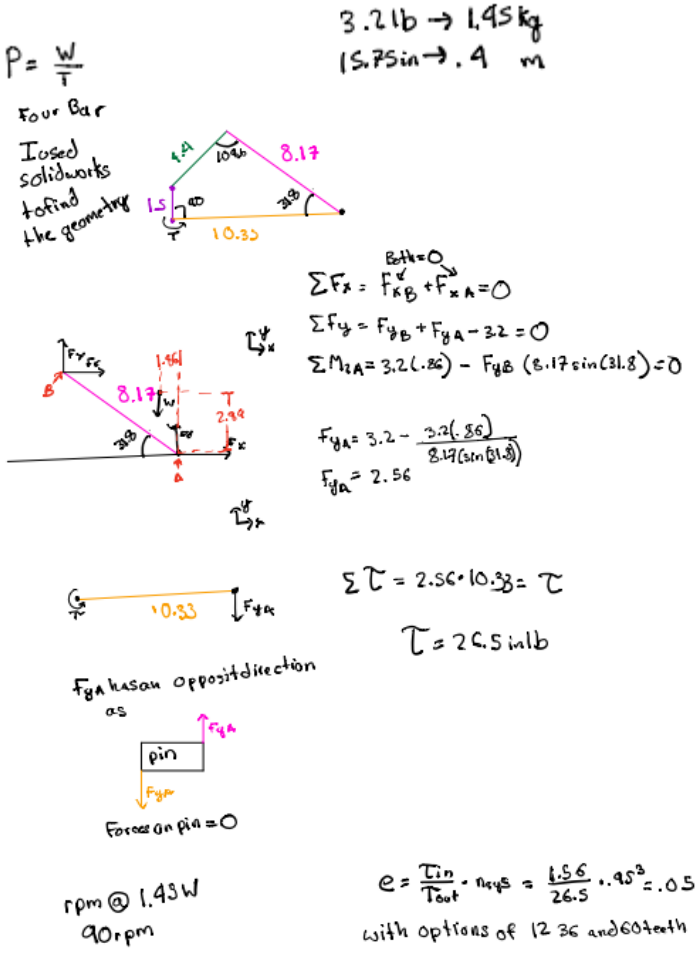


Figure 9: Power and Gearing Calculations

while a 269 vex motor would be sufficient, the final weights of the arm are still not known yet and thus we opted for a larger 393 motor. This would allow our motor to be able to handle the potential higher loads from a heavier arm. The next step would be to calculate the appropriate gear ratio for our robot. The robot would be at max torque when the crank is perpendicular to the robot. The calculations for gear ratios are shown in Figure 10. Of the available gear ratios, we decided that 2

reductions would be necessary. From 12 to 60 twice this would give us a reduction of 1/25 which is sufficient.

ELECTRICAL ANAYLSIS

Our custom circuit was a 3-position switch with two options a red pathway and blue pathway which tells the robot which autonomous to use and lights an appropriate LED. A diagram is shown in Figure 11.

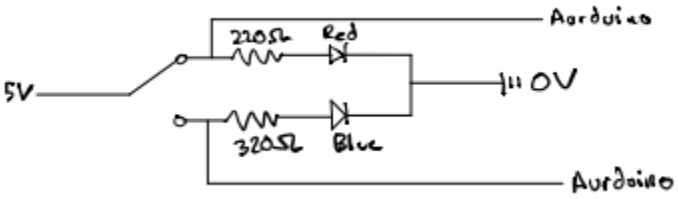


Figure 10: Electrical Diagram

PROGRAMMING ANAYLSIS

SENSOR INTEGRATION:

On our robot, we have four different kinds of sensors for the stability and functionality of the robot itself. We used a potentiometer, three VEX line trackers, and two VEX bump switches. These sensors will be useful for both autonomous and teleoperated alike.

The potentiometer is used on the four-bar linkage to gauge position of the bin we intend to implement. With this, we can use proportional control to reach the setpoints necessary for our bot to function well.

The line trackers are going to be used almost exclusively for our autonomous program. We will use the three sensors to figure out how far off the robot was from the line. The tracking will be useful for the state-machine structure of the autonomous program.

The bump switches are placed on the back of the robot, where the eggs will be dumped out from the robot. These switches are to verify that the bot is fully pressed against the BARN, which we intend to dump the eggs into. They can also be used during autonomous to detect whether the bot has reached the PEN and if it is pressing against it evenly. This will be key in our strategy, as the bin must empty very close to the BARN itself.

PROGRAMMING METHODOLOGY:

The programming for this robot is quite easy to understand, and generally explains itself through the function names. During autonomous, the bot uses a simple state machine to follow the line to the PEN and once the bump switches on the front of the bot are triggered, the bot pushes the pen forward and then dumps. The code for the line tracking is similar to the code written for the lab, with the operation being ended by the bump switches rather than a horizontal line.

The teleoperated is simple. There are three functions for the main objective: raise, lower, and intake. Both raise and lower use a potentiometer on the four-bar linkage to find the correct position. These will be controlled through proportional control for speed and precision. The intake function would just run the motor controlling the intake roller. Additionally, there is a function to deploy a small hook from the hopper, to climb.

SECTION 7: SUMMARY AND ANAYLSIS

During the CDR the robot performed well for what we were trying to accomplish. We were successfully able to intake eggs quickly and could dump loads of up to 16 eggs. There were two main issues with our robot during the CDR. The first would be that due to the base of the robot being lower to the ground than expected we were unable to climb the ramp. The second

issue was our code had no autonomous functionality. During the CDR we demonstrated our robot's ability to quickly obtain and dumb eggs by scoring 41 points into the NEST.

After the CDR we decided that working on autonomous was the best path option and worked on developing PID control for our 4-bar mechanism and attempted to develop a line tracking path. Unfortunately, while we did get the autonomous egg dumping to work we were unable to successfully test and create a line tracking program.

During the competition we had several unexpected errors occur. We had two cases of axles falling out due to a lack of proper securing methods and we had a battery die after the first match. Despite all this we came 4th out of 10 in the competition and scored a total of 217 points.

All in all, the team feels this class left us with a great wealth of knowledge in the field of robotics. Even though the class was a large time commitment and difficult in many aspects. The class was truly enjoyable and excites us for future robotics classes. We also feel that this class is truly a class where you get out what you put into it. The more work you put into it, the more you learn.

APPENDIX

FULL ROBOT CODE

RBEFinaL.ino

```
/* This is the RBE 1001 Template as of
*
* 3/28/17
*
* This Template
* is designed to run the autonomous and teleop sections of the final
* competition. Write and test your autonomous and teleop code on your
* own and place the code in auto.cpp or teleop.cpp respectively.
* The functions will be called by the competition framework based on the
```

```
* time and start button. DO NOT change this file, your code will be called
* by the framework. The framework will pass your code a reference to the DFW
* object as well as the amount of MS remaining.
*/
#include <DFW.h>
#include "MyRobot.h"
MyRobot lifty;
DFW dfw(&lifty); // 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. Only needed for debug
dfw.begin(); // Serial1 output begin for DFW library. Buad and port #."Serial1 only"
lifty.initialize();
lifty.dfw=&dfw;
}
void loop() {
dfw.run();
}
```

MyRobot.cpp

```
#include "MyRobot.h"
#include "Arduino.h"
/**
These are the execution runtions
*/
MyRobot::MyRobot() {}
void MyRobot::initialize() {
leftMotor.attach(4, 1000, 2000);
rightMotor.attach(5, 1000, 2000);
liftMotor.attach(6, 1000, 2000);
feedMotor.attach(7, 1000, 2000);
pinMode(circuitPin, INPUT);
}
enum autoStates {
STARTING,
FIRSTLINE,
SECONDLINE,
THIRDLINE,
PUSH,
DUMP,
RIGHTTURN,
LEFTTURN
} autostate;
```

```
void MyRobot::feed() {
feedMotor.write(180);
}
void MyRobot::backFeed() {
feedMotor.write(0);
}
void MyRobot::feedOff() {
feedMotor.write(90);
}
void MyRobot::driveStraight() {
leftMotor.write(45);
rightMotor.write(135);
}
void MyRobot::followRight() {
leftMotor.write(35);
rightMotor.write(135);
}
void MyRobot::followLeft() {
leftMotor.write(45);
rightMotor.write(145);
}
void MyRobot::brake() {
leftMotor.write(90);
rightMotor.write(90);
}
void MyRobot::lineFollow() {
int leftTrack = analogRead(leftTrackPin), midTrack = analogRead(midTrackPin), rightTrack =
analogRead(rightTrackPin);
if (leftTrack > lightThreshold && midTrack < lightThreshold && rightTrack > lightThreshold)
driveStraight();
else if (leftTrack < lightThreshold && midTrack < lightThreshold && rightTrack > lightThreshold)
autostate = LEFTTURN;
else if (leftTrack > lightThreshold && midTrack < lightThreshold && rightTrack < lightThreshold)
autostate = RIGHTTURN;
else if (leftTrack > lightThreshold && midTrack > lightThreshold && rightTrack < lightThreshold)
followRight();
else if (leftTrack < lightThreshold && midTrack > lightThreshold && rightTrack > lightThreshold)
followLeft();
}
void MyRobot::moveTo(int position) {
int potVal = analogRead(potPin);
int error = potVal - position;
if (90 + KpUp * error > 180) {
```

```
liftMotor.write(0);
Serial.println("full up");
}
else if (90 + KpDown * error < 0) {
liftMotor.write(180);
// Serial.println("full down");
}
else if (error < 10) {
liftMotor.write(90);
// Serial.println("off");
}
else if (error < 0){
liftMotor.write(90 + KpUp * error);
// Serial.print("pid");
// Serial.println(90 + Kp * error);
}
else if (error > 0){
liftMotor.write(90+ KpDown * error);
}
}
/**
Called when the start button is pressed and the robot control begins
*/
void MyRobot::robotStartup() {
}
/**
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) {
// Serial.print("\r\nAuto time remaining: ");
// Serial.println(time);
int leftTrack = analogRead(leftTrackPin), midTrack = analogRead(midTrackPin), rightTrack =
analogRead(rightTrackPin);
int colorInput = digitalRead(circuitPin);
// Serial.println(time);
if(time > 11000){
moveTo(upPos);
}
else{
liftMotor.write(90);
```

```
}
// driveStraight();
// delay(startingDelay);
// brake();
// switch (colorInput){
// case 1:
// autostate = STARTING;
// switch (autostate) {
//
// case FIRSTLINE:
// lineFollow();
// break;
//
// case SECONDLINE:
// lineFollow();
// break;
//
// case THIRDLINE:
// lineFollow();
// break;
//
// case LEFTTURN:
// leftMotor.write(180);
// rightMotor.write(180);
// if(leftTrack > lightThreshold && midTrack < lightThreshold && rightTrack > lightThreshold){
// autostate = SECONDLINE;
// }
// break;
// case RIGHTTURN:
// leftMotor.write(0);
// rightMotor.write(0);
// if(leftTrack > lightThreshold && midTrack < lightThreshold && rightTrack > lightThreshold){
// autostate = THIRDLINE;
// }
// break;
//
// case PUSH:
// driveStraight();
// delay(250);
// autostate = DUMP;
// break;
//
// case DUMP:
```

```
// moveTo(upPos);
// break;
// }
// break;
//
// case 0:
// autostate = STARTING;
// switch (autostate) {
// case STARTING:
// driveStraight();
// delay(startingDelay);
// autostate = FIRSTLINE;
// break;
//
// case FIRSTLINE:
// lineFollow();
// break;
//
// case SECONDLINE:
// lineFollow();
// break;
//
// case THIRDLINE:
// lineFollow();
// break;
//
// case LEFTTURN:
// leftMotor.write(0);
// rightMotor.write(180);
// if(leftTrack > lightThreshold && midTrack < lightThreshold && rightTrack > lightThreshold){
// autostate = THIRDLINE;
// }
// break;
// case RIGHTTURN:
// leftMotor.write(180);
// rightMotor.write(0);
// if(leftTrack > lightThreshold && midTrack < lightThreshold && rightTrack > lightThreshold){
// autostate = SECONDLINE;
// }
// break;
//
// case PUSH:
// driveStraight();
```

```
// delay(250);
// autostate = DUMP;
// break;
//
// case DUMP:
// moveTo(upPos);
// break;
// }
// break;
// }
}
/**
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
*/
enum teleStates {
DRIVING,
FEEDING,
BACKFEEDING,
RAISING,
LOWERING
} telestate;
void MyRobot::teleop(long time) {
// Serial.print("\r\nTeleop time remaining: ");
// Serial.print(time);
// Serial.println(analogRead(potPin));
telestate = DRIVING;
if (dfw->l1() == 1) {
telestate = FEEDING;
} if (dfw->one() == 1) {
telestate = RAISING;
} if (dfw->three() == 1) {
telestate = LOWERING;
} if (dfw-> r1 () == 1) {
telestate = BACKFEEDING;
}
switch (telestate) {
case DRIVING:
liftMotor.write(90);
feedOff();
rightMotor.write(180 - dfw->joystickrv()); //DFW.joystick will return 0-180 as an int into
```

```
rightmotor.write
leftMotor.write(dfw->joysticklv());
break;
case FEEDING:
feed();
rightMotor.write(180 - dfw->joystickrv()); //DFW.joystick will return 0-180 as an int into
rightmotor.write
leftMotor.write(dfw->joysticklv());
break;
case BACKFEEDING:
backFeed();
rightMotor.write(180 - dfw->joystickrv()); //DFW.joystick will return 0-180 as an int into
rightmotor.write
leftMotor.write(dfw->joysticklv());
break;
case RAISING:
feed();
liftMotor.write(0);
break;
case LOWERING:
liftMotor.write(170);
break;
default:
rightMotor.write(180 - dfw->joystickrv()); //DFW.joystick will return 0-180 as an int into
rightmotor.write
leftMotor.write(dfw->joysticklv());
break;
}
}
/**
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");
liftMotor.write(90);
feedMotor.write(90);
leftMotor.write(90);
rightMotor.write(90);
}
```

MyRobot.h

```
#pragma once
#include "Servo.h"
#include <DFW.h>
#include <AbstractDFWRobot.h>
class MyRobot :public AbstractDFWRobot{
public:
MyRobot();
DFW * dfw;
/**
* Called when the start button is pressed and the robot control begins
*/
void robotStartup();
void driveStraight();
void brake();
void lineFollow();
void followRight();
void followLeft();
/**
* 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 autonomous( long 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 teleop( long time);
void feed();
void backFeed();
void feedOff();
/**
* Called at the end of control to reset the objects for the next start
*/
void robotShutdown(void);
/**
* Return the number of the LED used for controller signaling
*/
int getDebugLEDPin(void){return 13;};
```

```
void initialize();
void moveTo(int position);
~MyRobot(){};
private:
Servo feedMotor, leftMotor, rightMotor, liftMotor;
const int leftTrackPin = 0, midTrackPin = 1, rightTrackPin = 2, potPin = 3, circuitPin = 22;
int inval, error;
const int lightThreshold = 400;
const float KpUp = 1.1, KpDown = 0.4;
const int upPos = 40, downPos = 640;
const int startingDelay = 50;
};
```