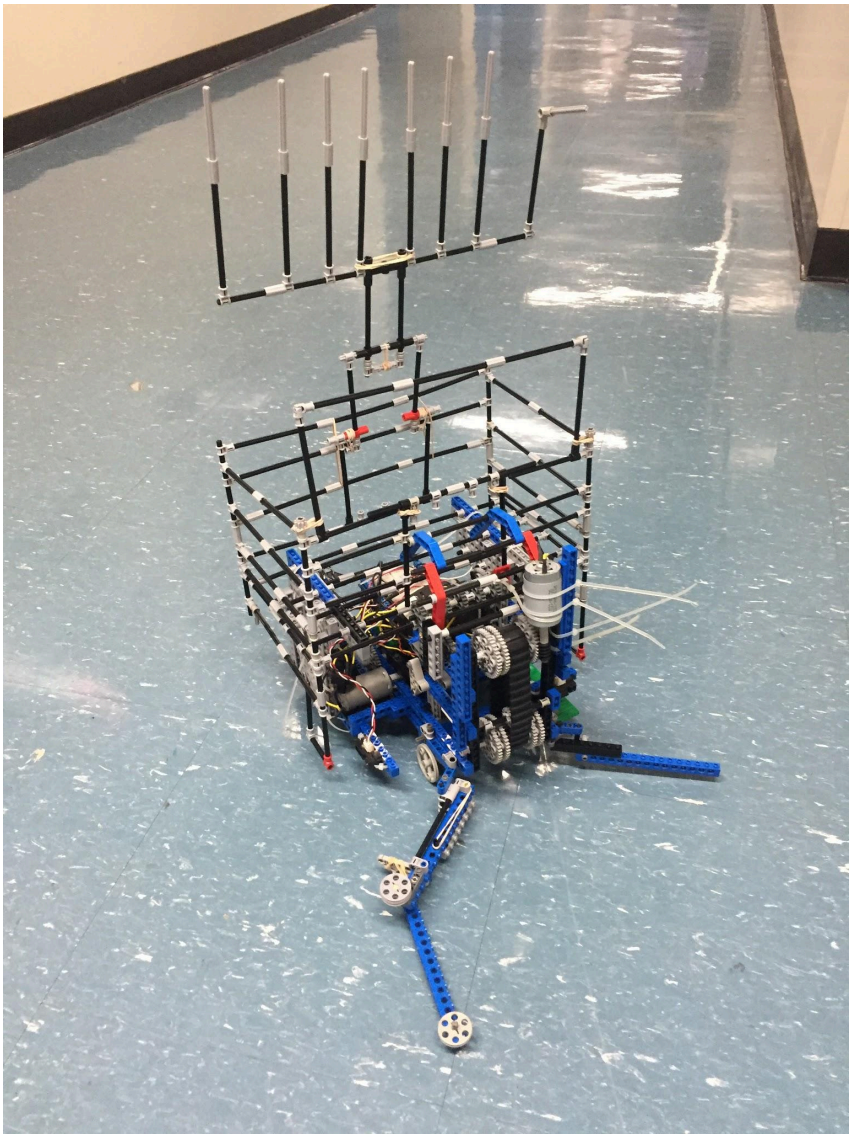
RoboRat Final Report



Ryan Tsukamoto
Austin Semmelroth

ME 179L
June 10, 2017

I. Introduction and Objectives

The main idea behind the RoboRat race was to collect as many points as possible through collecting “cheese” using automated robots. The interesting part about this competition is that the cheeses are worth varying amount of points depending upon their location on the board. The competition field layout and point scales for the different cheeses can be seen in *Figure 1* and *Table 1* respectively.

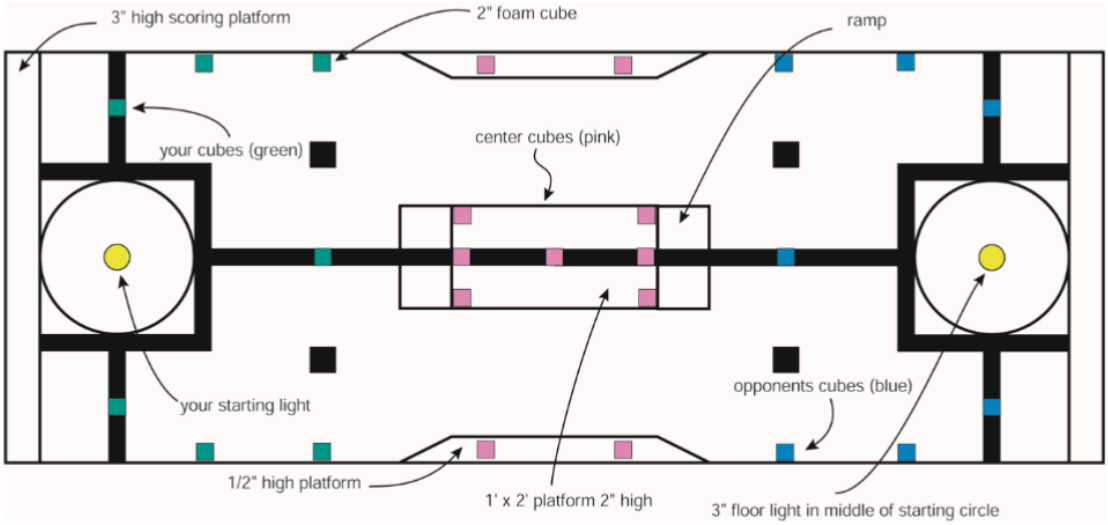


Figure 1: RoboRat Competition Field Showing Cheese Location

Figure 1 only shows the cheese location on the competition field itself. There are also nine total hanging cheese spread evenly throughout the field roughly two feet above the stage. There is a single golden cheese that is worth the most amount of points. It is located hanging above the center platform in the exact center of the field.

Deposit location \ cheese type	Your cheese	Middle cheese	Opponent's cheese	Golden cheese
On your robot	2	3	4	20
On your wall	8	12	16	80

Table 1: Table Showing Cheese Value Based on Location

The point scale seen above mentions that more points can be received if the cheese is deposited on your wall. This means that the cheese is collected on the robot, then is left freestanding on the wall the robot started the competition on.

The robot design constraints are also taken into consideration for this competition. There are quite a few of these constraints, but the most important is that the robot must be able to fit within a 1’ cube box. After the robot successfully fits within this volume, accessories can be deployed and the robot is free to open up and spread out to a size that may be larger than 1’ in either direction. This action may be achieved through any means possible from rubber bands to servo motors.

Other constraints are the robot must only be made from a combination of LEGO parts, arduino board and other supplied accessories. Zipties may be used to attach these accessories such as mounting motors and also to compress LEGO parts. Otherwise, no adhesives or other ways to fix LEGOs or other parts is not permitted. Finally, a kill switch must be included in the design of the robot.

II. Available Components

Arduino Stack

An Arduino Uno microcontroller, powered with a 9V battery, and Arduino IDE software were used to control the robot. The sensors and servo motor were attached to Arduino pins on the prototyping shield and wired to power using the breadboard. The three DC motors were connected to the Arduino through the motor shield layer and were powered with a AA battery pack.

Sensors

The sensors we chose to include on our robot included encoders, range sensors, and a photoresistor. Each encoder was composed of a break-beam sensor and a LEGO pulley piece. A short range sensor was used to follow the wall while a long range sensor was used to signal a turn. Both range sensors were calibrated in the wall follower assignment by measuring sensor voltages at various distances. Finally, a photoresistor was used with another resistor as a switch that sensed the starting light.

Actuators

Two brushed DC motors were used for the drivetrain and one was used for the cheese collecting mechanism. A servo motor was also used on the robot in order to actuate the basket drop, a feature that was never fully implemented.

LEGO

Aside from the electrical components,the entire robot was constructed out of LEGO parts. The most frequently used pieces included beams, gears, axles, couplings, wheels, and spacers. The LEGO belt piece was crucial to our cheese collecting mechanism (see *Figure 2*).

Other Materials

Rubber bands were used to spring-load various parts and helped our robot collapse into the 1’ cube size requirement. Zipties were used to attach motors and sensors as well as hold LEGO together under compression.

III. Design Description

There are two main design approaches for this competition. The first is to design a robot that will gather the cheese found on the stage of the competition field or “ground cheese”. The second approach is to try to collect the hanging cheese or “sky cheese”. We decided to incorporate both these strategies in our robot design that would gather she ground and sky cheese simultaneously. By doing this, the most amount of cheese would be collected as quickly and efficiently as possibly.

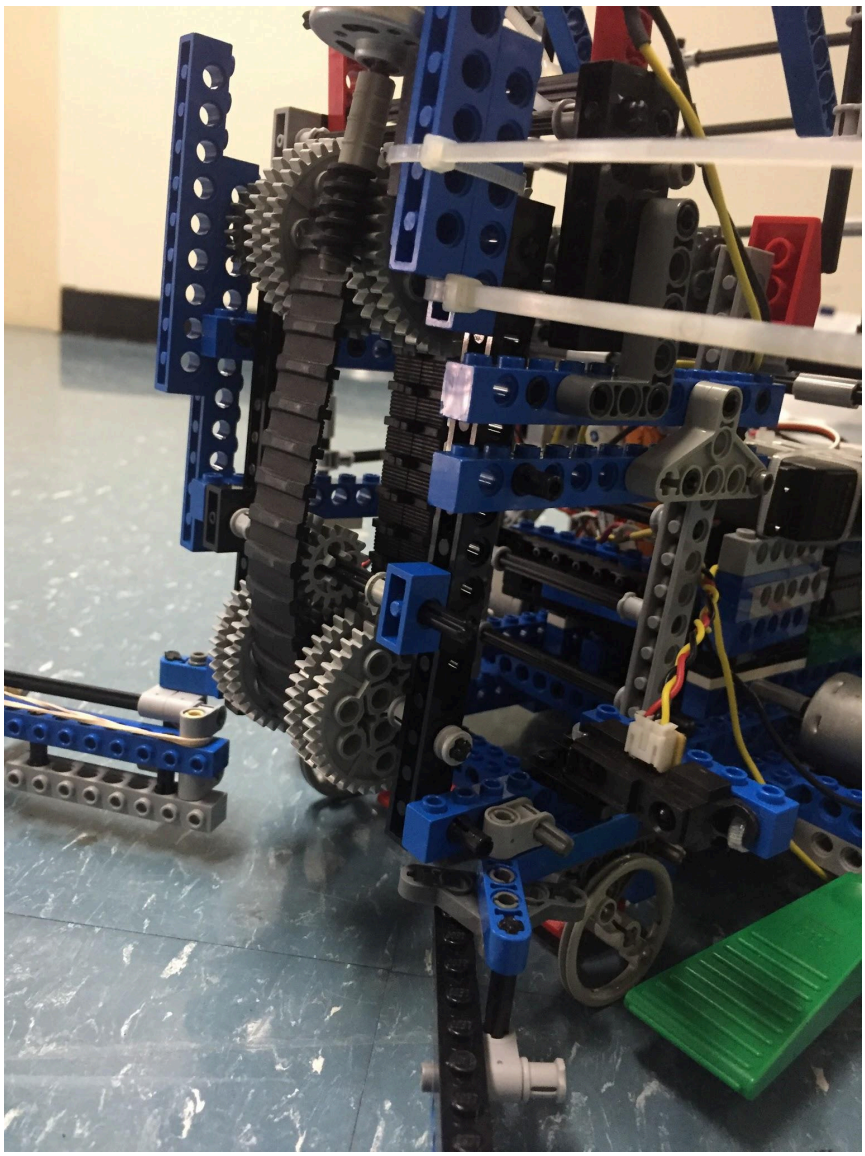


Figure 2: The “Pitching Machine”: Ground Cheese Collecting Mechanism

Figure 2 shows the mechanism used to collect the ground cheese and store it on an onboard basket on the robot. The inspiration behind this concept was a baseball pitching machine. As the belt in the image above rotates in a counterclockwise direction, the cheese will get sucked up against the cage behind it. This causes the cheese to spin its way up the chute and into the storage basket.

It proved to be a very reliable design as it was able to handle multiple cheeses at once without stalling the motor or becoming jammed. This was partly due to the use of a worm gear in the drive train. Worm gears provide an extremely favorable gear reduction with a limited number of parts. This means there are less places for the design to fail and can be relied upon.

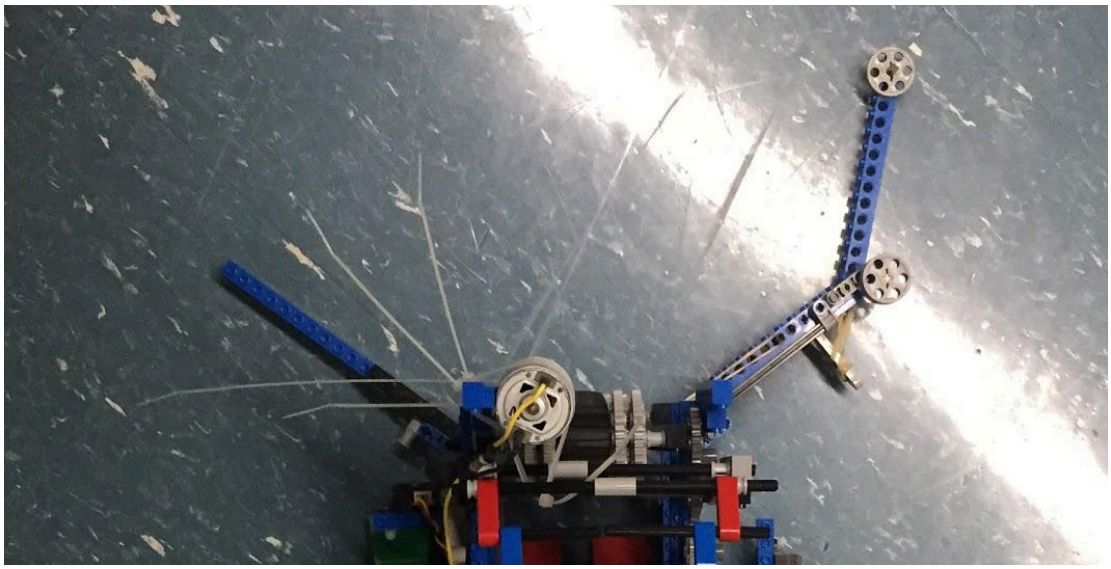


Figure 3: Cheese Funneling System

One disadvantage of the “pitching machine” concept is that the cheese must enter the chute directly under the belt, otherwise it won't get “sucked up”. Therefore, a funneling system had to be implemented to guide the cheese the the entrance as seen in *Figure 3*. Due to the layout of the cheese on the competition field, and the robots must travel in a counterclockwise rotation, the right side of this funnel is much more complex.

The funneling arm uses two different types of motion, linear and rotational, for two main reasons. The first of which is that the robot must be able to fold into the 1’ cube design constraint. The second is so it can swipe off the cheese along the wall without getting snagged or caught on the edges of the competition field. When traveling along the board, the arms automatically adjust to the difference in wall shape, and the error in the robot's ability to accurately travel a constant distance from the wall. The arms are actuated through the use of rubber bands.

As seen in the image above, rollers were added to the joints. This allowed the arms to smoothly travel along the wall with minimal friction and binding. The left arm does not follow along a wall so no automatic adjustment was necessary. This arm imply folds up to fit within the size constraint, and falls through the use of gravity.

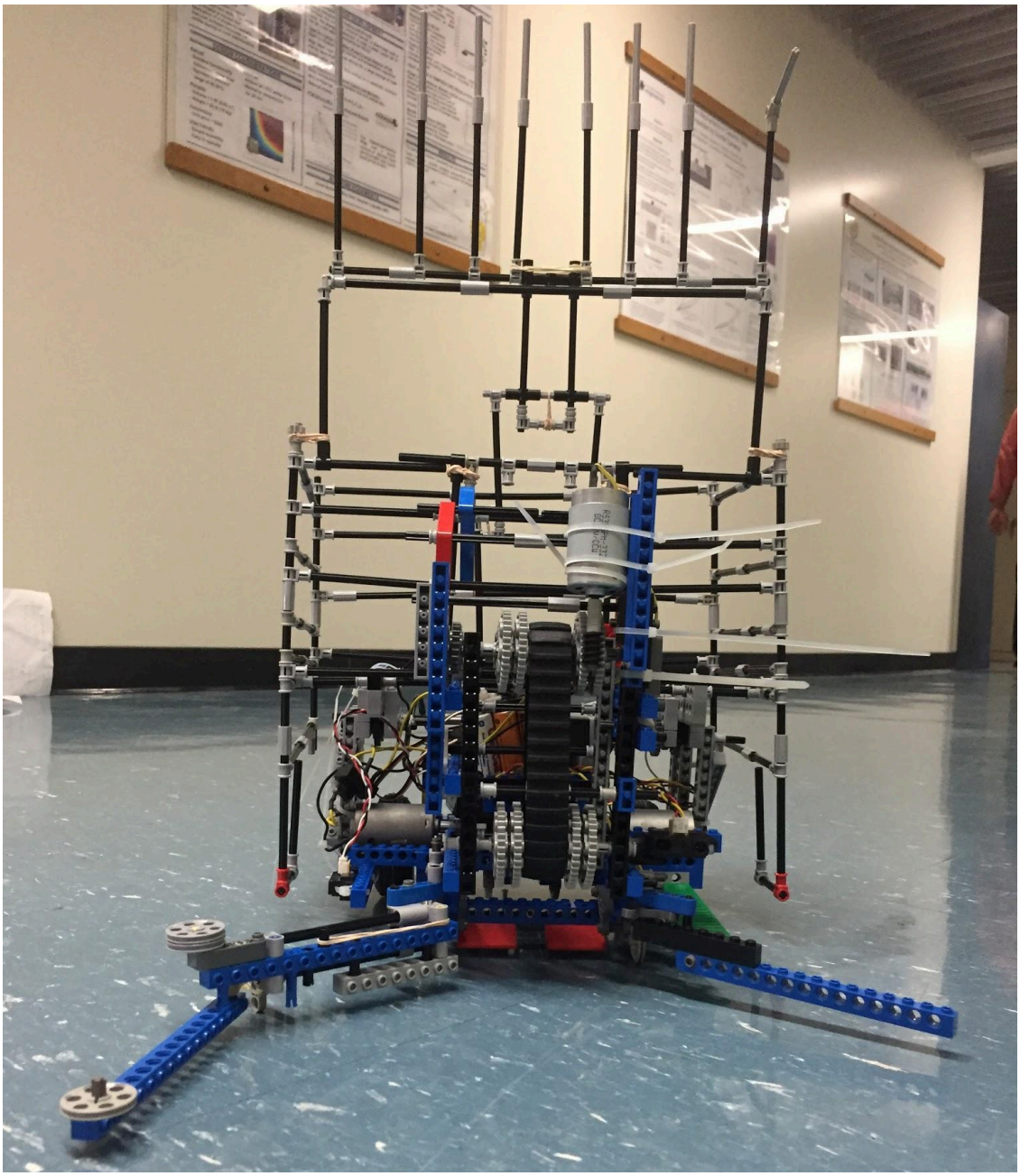


Figure 4: The “Rake AKA Pitchfork”: Ground Cheese Collecting Mechanism

To collect the sky cheese, a rake system was implemented. This idea behind this feature was it would simply rake the sky cheese off their hangers and have them fall into the storage basket below. Dampeners were included in this design so the cheese would fall smoothly below, and not bounce out or miss the basket. The most difficult aspect of this design was how to make it collapsible and fit into the allowable volume. Rubber bands, hinges, and makeshift linear bearings were used to successfully meet this.

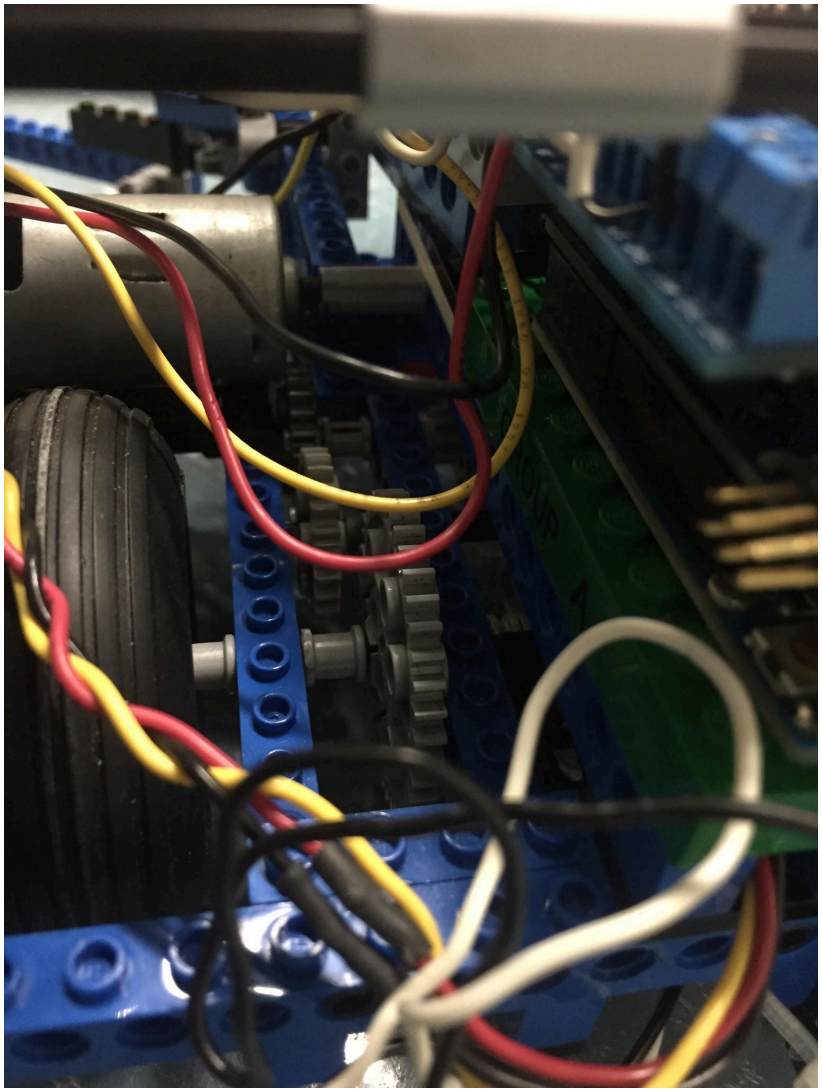


Figure 5: Drivetrain

Figure 5 shows the drivetrain of the robot. When designing the drivetrain, we wanted to ensure there would be no gears slipping. In order to ensure gears meshed will, an extender was added to the shaft of the motor. At the end of this extender, a pinion gear (not seen) is used and supported by two LEGO pieces on either side (not seen). The pinion gear is now only limited to rotation, and will always be perfectly messed to the gear below it (not seen).

The “gear box” provides a two stage reduction from motor to wheel. This is a total gear reduction of 50:1. Through our previous experience in the class, we decided this was the optimal gear reduction for torque and speed. The necessary torque to the wheels is important to propel the robot forward and not stall the motors. This torque is especially prevalent while turning due to the added friction of the front wheel and pivoting the drive wheels. However, speed is also something to take into consideration. We wanted to robot to still be able to move quickly enough to collect the cheese as fast as possible before the opponent is able to do so. As a result, an optimal gear ratio was found for necessary torque and a decent top speed.

We discussed DC motor power curves in class and learned that as the speed of the motor is increased, the power output of the motor increases as well. With this information in mind, we ran our motors as close to top speed as possible. For example, the motor driving the “pitching machine” belt was ran at its top speed. In addition, the motors driving the wheels were run at full speed and the robot was turned simply by slowing the correct wheel. This also ensured our robot would operate as fast as possible.

To drive our robot, we used a wall following technique with a simple P-controller to keep the robot at a desired distance from the wall it was traveling along. Range sensors were used at the front, and right side of the robot. Through past experience in the course, we learned that a full PID-controller added unnecessary complexity to the benefits it provided. Therefore, a P-controller would work well enough for our implementation and be stable enough. To do this, an error was calculated between the desired wall distance, and the distance it is currently at. The distance from the wall to the robot was multiplied by a proportional constant of 0.8. Through many trials, this constant was desired because it corrected the robot direction quickly enough without creating an underdamped system.

To determine when to turn the robot, a range sensor was used pointing forward. This sensor was placed high enough to see above cheese, yet see the top of the wall in front of it. When the robot was within a certain distance of the wall, the left wheel would run in reverse while the right wheel remained stationary. This would cause the robot to pivot hard to gather any cheese that may be in that corner. The robot would then turn until the front sensor recognized the robot has sufficiently turned far enough to complete the corner. The the robot is hard coded to run forward for a tenth of a second before the side sensor is used again and the P-controller takes over. Overall, the robot is designed to keep running in a circle around the competition field in a counterclockwise rotation gathering as much cheese as possible before the time runs out or a collision with another robot occurs.

IV. Competition Performance

Our robot swept the competition and we went undefeated as we took first place in the RoboRat competition. Our highest score of all our runs was 32 total points. Relative to our competition, we did very well and didn't have any close races. This was due to the robustness, reliability, and consistency of our design. Throughout the entire competition we never had to make any repairs to our robot. We feel this is what gave us the edge to win. Other than our first race, our scores remained fairly constant ranging between 25 and 32 points. The first race we received 9 points because we followed too closely to the wall and did not gather any sky cheese. Following this run we changed to code to stay further away from the wall, which allowed us to gather the sky cheese greatly improving our overall score.

V. Lessons Learned

The most important lesson we learned is to just have fun and enjoy what you are doing. It is important to laugh and “mess around” with one another. Team dynamic is such an important part of being an engineer and succeeding at life in general. By throwing around “dumb” ideas or trying to think of different ways about going about an issue, some ideas will stick, and might actually end up working very well. This happened with us many times over the past couple weeks. We would like to tell next year's students to try and get to know their partner as much as possible. This way you can understand each other's

strengths and weaknesses and build upon those to create the best overall design in the end.

We made heaps of changes to our robot and went through countless iterations. First of all, we changed the drivetrain of the robot 4 total times until reaching our final design. The first concepts used u-joints, but they were noisy, and created too much backlash in the system rendering it unstable. Once we switched to a straight gearbox and drivetrain we reduced to overall footprint and went through a couple designs to limit friction and improve robustness.
Other improvements revolved around the funnel system. This was difficult because we had to find a way to make it collapse, yet be strong enough to guide the cheese to the chute. We tried configuring it to fold every which direction until coming to our final design. The rollers on the joints were a last minute addition because we noticed the arm would sometimes catch alongside the wall.

The most important change we made, however, was the motor attachment to the pitching machine system. We had many issues attaching this motor because the worm gear would become disengaged and not be in contact all the time. Once we fixed this, the mechanism worked perfectly and we had no more issues.

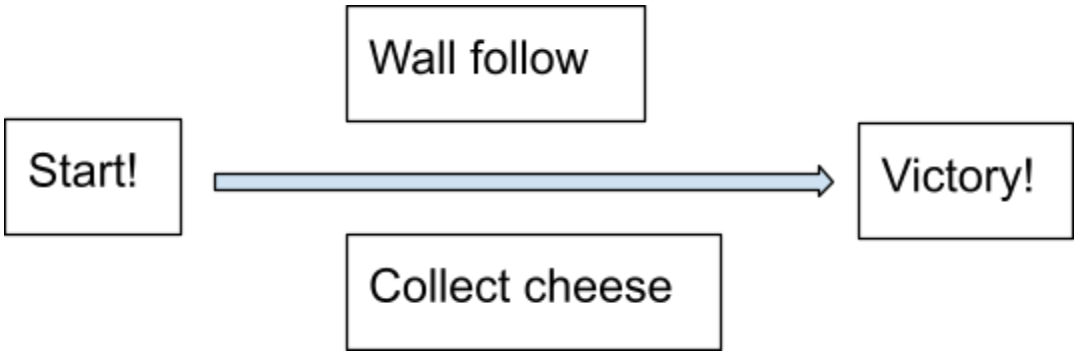
There were obviously the changes made to the code and P-controller to make it work properly. The main skeleton was not changed significantly and the changes were mostly tuning and changing around constants.

If we were to repeat the competition in two weeks, we would not significantly change our design, as we performed well. However, there are two main things we would add. The first of which would be a way to gather the cheese on the ramp as we pass by it. These cheeses were typically left unbothered and would be a quick way to gather points. This could be achieved through the use of anther arm off the left side of the robot to knock the cheese off the ramp and into the funnelling system to get sucked up with the pitching machine system.
Another addition to the robot would be a way to deploy the basket at the end of the run. If this could be successfully achieved, we could quadruple our point count. We played around with this system a bit and had it roughly working. However, it did not work well enough to use in the competition so we decided to not attempt it as it would be too risky and accidentally drop the basket elsewhere on the competition field.

Our design process was fairly straightforward and effective. We would each pick a certain mechanism we would want to work on and go off individually and start building a prototype. Once that prototype was built, we would reconvene and discuss it. Post discussion, improvements to the design would be made and we would run it through its courses and trials. Iterations of discussions, testing, and reengineering would occur till we felt confident with the device. Next time we approach design problems we would try to communicate better before the initial prototype. We rarely discussed how we were going to build said prototype and typically just jumped right into it. If we could spend a couple minutes and draw a few sketches first, we could expedite the design and iteration process. This would result in a more refined design in the end and allow for more time to make improvements elsewhere.

VI. Appendix

A. FlowChart



B. Arduino sketch

```
//ROBORAT
//RYAN T. and AUSTIN S.
//THIS CODE WILL WIN THE COMPETITION
//DON'T BELIEVE ME? JUST WATCH...

//Include the needed libraries
#include <AFMotor.h>
#include <SoftwareSerial.h>
#include <math.h>
//Define Constants
```

```
#define DEBUG 1  //By setting DEBUG to either 0 or 1, the program will exclude or include the print statements when it compiles
#define SwitchPin1 0   // switch is connected to this pin
#define SwitchPin2 9
#define encoderPin1 2   // Encoder i.e. break-beam sensor (2 or 3 only, to allow hardware interrupt).
#define encoderPin2 3
#define LCDTxPin 13   // LCD connected to this pin (14 is analog 0)
#define DummyRxPin 4  // Not used by LCD Board, can be any unused pin
#define SPEED 255     // Set Speed to be used for motors
#include <Servo.h>

Servo servo;  // create servo object to control a servo
SoftwareSerial mySerial =  SoftwareSerial(DummyRxPin, LCDTxPin);  //Change Tx and Rx Pins to pins of our choosing
AF_DCMotor Right_Motor(4, MOTOR34_1KHZ); // create right motor on port 3, 1KHz pwm
AF_DCMotor Left_Motor(3, MOTOR34_1KHZ); // create left motor on port 4, 1KHz pwm
AF_DCMotor Front_Motor(1, MOTOR34_1KHZ); // create front motor on port 1, 1KHz pwm
//Options for PWM, from quiet but power hungry to loud but lower power cost: MOTOR12_64KHZ, MOTOR12_8KHZ, MOTOR12_2KHZ, or MOTOR12_1KHZ
//(Only for ports 1 and 2, ports 3 and 4 are(and can only be) set to MOTOR34_1KHZ

const int analogInShort = A1; //Short-range distance sensor for wall following
const int analogInLong = A0; //Long-range distance sensor for turning
int LightPin = A2; // select the input pin for LDR
int LightValue = 0; // variable to store the value coming from the sensor

///SET UP///

double LongSensor = 0;
double LongVolt = 0;
double ShortSensor = 0;
double ShortVolt = 0;
double Wall_Error = 0;
double Front_Error = 0;
//double P = 0;
double Right_Speed = 0;
double Left_Speed = 0;
```

```
volatile int encoderCount=0;   // Use "volatile" for faster updating of value during hardware interrupts.
int currentencodercount = 0;
int encoderCountGoal = 10;
int Lap = 999999999999; //600;
int Angle = 0;



//FOR ADJUSTMENT:
//const int TURNTIME = 800;
double WallDist = 225; //200; //Distance from wall to be followed
double FrontDist = 210; //Distance from wall to TURN
//const double kp = 100; //Proportional Control
//double tolerance = 15;
//const double StartTurn = 2.0  ; //Voltage from distance to front wall
//int ControlTime = 250;



void setup(){
  pinMode(LCDTxPin, OUTPUT);
  mySerial.begin(9600);  //Set baud rate for the LCD serial communication
  mySerial.print("?f"); //Sends clear screen command to LCD
  pinMode(SwitchPin1, INPUT); //Makes Switch Pin an input
  pinMode(SwitchPin2, INPUT); //Makes Switch Pin an input
  digitalWrite(SwitchPin1, HIGH);   //Enables the pull-up resistor on Switch Pin
  digitalWrite(SwitchPin2, HIGH);   //Enables the pull-up resistor on Switch Pin
  Right_Motor.setSpeed(SPEED);
  ////Motor Things////
  Left_Motor.setSpeed(SPEED);
  Front_Motor.setSpeed(SPEED);
  ////encoder things////
  int interruptPin1 = encoderPin1 - 2;   // Hardware interrupt pin (0 or 1 only, to refer to digital pin 2 or 3, respectively).
  int interruptPin2 = encoderPin2 - 2;
  attachInterrupt( interruptPin1, Increment, FALLING);   // Attach interrupt pin, name of function to be called
        // during interrupt, and whether to run interrupt upon voltage FALLING from high to low or ...
  // Setup encoder i.e. break-beam:
  attachInterrupt( interruptPin2, Increment, FALLING);
  pinMode( encoderPin1, INPUT);
  digitalWrite( encoderPin1, HIGH);
```

```
  pinMode( encoderPin2, INPUT);
  digitalWrite( encoderPin2, HIGH);
  ////LCD Screen Things////
  mySerial.print("?f"); //Sends clear screen command to LCD
  mySerial.print("?x00?y0");  //Move cursor to position x=0 and y=0 on the LCD
display
  servo.attach(10);  // attaches the servo on pin 10 to the servo object
  servo.write(Angle); //sets servo to initial position
}

void loop(){

//+++Light sensor says..... GO!!!!+++//
//_______Get that young Cheese_______//

while(LightValue<200)
  LightValue = analogRead(LightPin); // read the value from the sensor

//+++++++++ WALL FOLLOWER ++++++++++//
//____________BEGINNING_________________//

  //P Controller
   ShortVolt= analogRead(analogInShort);
   LongVolt= analogRead(analogInLong);
   Wall_Error = WallDist - ShortVolt*.8;

   //New Motor Speeds
   Right_Speed = 255-Wall_Error; //-Front_Error;
   Left_Speed = 255+Wall_Error; //+Front_Error;
   if (Right_Speed>255)
      Right_Speed = 255;
   if (Right_Speed<0)
      Right_Speed = 0;
   if (Left_Speed>255)
      Left_Speed = 255;
   if (Left_Speed<0)
      Left_Speed = 0;

  //Turn Robot
  if( LongVolt>FrontDist){

    while(LongVolt>FrontDist-100){
      Right_Motor.setSpeed(0);
```

```
      Left_Motor.setSpeed(255);
      Right_Motor.run(FORWARD);
      Left_Motor.run(BACKWARD);
      LongVolt= analogRead(analogInLong);

     Front_Motor.setSpeed(255);
     Front_Motor.run(FORWARD);

      delay(100);
    }
    Right_Motor.setSpeed(200);
    Left_Motor.setSpeed(255);
    Right_Motor.run(FORWARD);
    Left_Motor.run(FORWARD);

    Front_Motor.setSpeed(255);
    Front_Motor.run(FORWARD);

    delay(500);

  }


  //Drive Roborat
  Front_Motor.setSpeed(255);
  Front_Motor.run(FORWARD);

  Right_Motor.setSpeed(Right_Speed);
  Left_Motor.setSpeed(Left_Speed);
  Right_Motor.run(FORWARD);
  Left_Motor.run(FORWARD);

  delay(20);

//_________________END_________________//
//+++++++++ WALL FOLLOWER ++++++++++//

}

void Increment() {
  ++encoderCount;
}
```

//TOLD YOU WE’D WIN