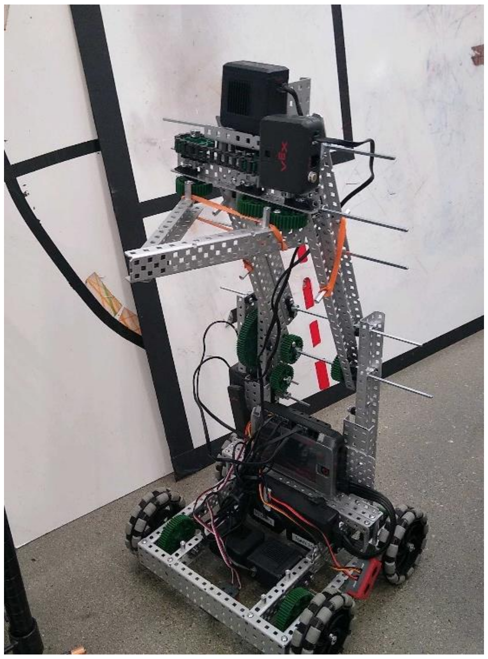
RBE 1001 B21, *Introduction to Robotics*

Final Project Report



"Pizza Time"

Team 14

Member	Signature	Lab Contribution (%)
Wyatt Harris	[illegible]	50 %
Bryce McKinley	[illegible]	50 %

Member	Signature	Project Contribution (%)
Wyatt Harris	[illegible]	50 %
Bryce McKinley	[illegible]	50 %

Grading:	
Report Mechanics	___/10
Mechanical Analysis	___/30
Sensor Analysis	___/10
Programming Competency	___/25
Design Reviews	___/25
Total	___/100

Table of Contents

Table of Contents i
List of Figures ii
Section 1: Introduction 1
Section 2: Preliminary Discussion 2
Section 3: Problem Statement 3
Section 4: Preliminary Design 4
Section 5: Selection of Final Design 6
Section 6: Final Design Analysis 9
Section 6.1: Mechanical Analysis 9
Section 6.2: Electrical Analysis 13
Section 6.3: Programing Analysis 15
Section 7: Summary/Evaluation 16
Section 8: Appendix 17
Section 8.1: Autonomous Program 17
Section 8.2: Remote Control Program 21

List of Figures

Figure 1- A diagram of the game field 1
Figure 2- An early blueprint for an alternate design 4
Figure 3- Front view of acquisition mechanism 5
Figure 4- side view of the robot, demonstrating the lift mechanism 7
Figure 5- decision matrix for the robot 8
Figure 6- close-up of the drive train 9
Figure 7- Free Body Diagram of the robot climbing a ramp 9
Figure 8- Free Body Diagram of robot moving horizontally 10
Figure 9- diagram of the lift at its lowest point 11
Figure 10- Diagram of the lift moving from its highest point to straight forward 11
Figure 11- Gear transmission of the four-bar 11
Figure 12- Photo of line sensors mounted to the base of the robot 14
Figure 13- Photo of the distance sensor mounted to the side of the robot 114

Section 1- Introduction

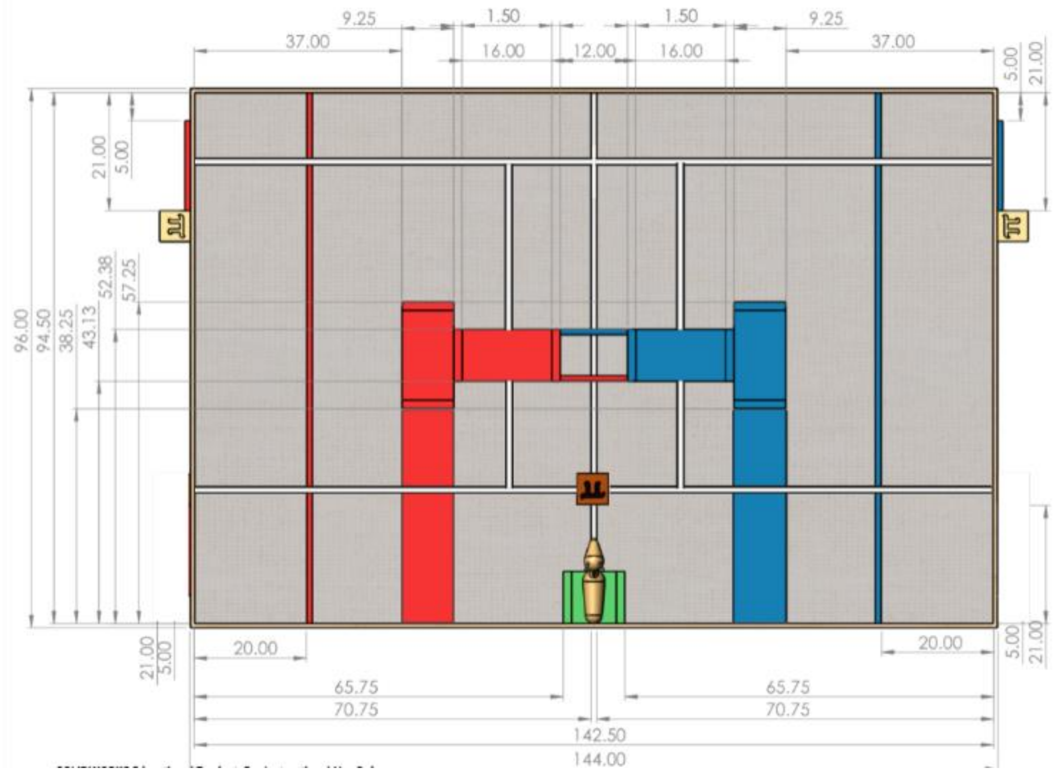


Figure 1- A diagram of the game field

In this project, we were challenged to create a robot that is capable of grabbing “pizzas” that are small 5.5” x 5.5” x .75” wooden blocks, picking them up, and moving them to different heights of “Faraday” and “Messenger”. Faraday and Messenger are the red and blue shelf structures shown in figure 1. Higher floors are worth more points, except for the top floor, and points are awarded for getting pizzas on all floors, from the achievement “Happy Dorm”. At the beginning of the CDR the robot starts at the bottom of a ramp and must perform tasks autonomously. In the OED, the robot starts on the field, but only has 30 seconds to complete autonomous tasks. During autonomous, it may get points for going over the bump, and delivering pizzas to Faraday Hall, which is the hall farther from the ramp. After autonomous, there is a remote-control session that lasts two minutes, where the robot may score points by continuing to deliver pizzas. During endgame, the last 30 seconds of the remote-control session, additional points may be earned by, lifting itself up on the aerial delivery bar, and pushing Gompei, the wooden structure in the construction zone. This report will explain our thought process in designing a robot in order to complete the given tasks.

Section 2- Preliminary Discussion

In order to win, we decided that being capable of completing the aerial delivery, and delivering pizzas were the most important tasks. Pizza delivery is the only task the robot can earn points for throughout the majority of the remote-control period. That alone makes it a very valuable task to be able to complete. Being able to reach all levels of the dorm achieves the “happy dorm” requirement, getting 30 points for a dorm. That is significantly more than the 24 points that may be earned for completing the entirety of “Messenger Hall”. Because it’s more worthwhile to get points at all levels, we made it a goal to be able to deliver pizzas at any height. We also noticed that both the aerial delivery and pushing Gompei are highly valuable in terms of points, at 35 for the aerial delivery, and 20 to 40 depending on whether the golden pizza is used for Gompei. However, only 30 seconds are given to complete both tasks, so we decided to focus primarily on the lift, especially since we already intended to have a lift on our robot in order to deliver pizzas.

Our initial concept was a robot with some sort of lift mechanism that could both support the robot and be used to bring an acquisition method up and down. Our acquisition method would be attached to the robot lift and controlled separately, so it could pick up and drop the Pizzas at any height. During autonomous and the non-endgame section of remote control we planned to pick up and deliver pizzas, and during endgame, we planned to complete the aerial delivery.

Section 3- Problem Statement

Based on our gameplan, we thought of some design goals for our robot. Our robot needed to be able to lift itself up, pick up pizzas, reach all floors of the residence halls, and go over the speed bump to enter the construction zone. Using those goals, we can create a priority list:

Priority List-

1. Pick up pizzas to all floors of Faraday Hall
2. Complete the Aerial delivery
3. Pick up pizzas to all floors of Messenger Hall
4. Go over the speed bump
5. Pick up pizzas off the ground
6. Move Gompei

Completing all the goals would be ideal, but only the first three were especially important to us, as our ideal gameplan primarily involved them. Picking up pizzas off the ground may help in some situations where pizzas are dropped, or are closer than the Bakery, but getting pizzas from the bakery is an alternative that requires the same design as delivering pizzas to residence halls. Going over the speed bump can be necessary for maneuvering around the game field, so it was somewhat important to us, but not nearly as important as other point scoring functions. Moving Gompei would be nice for added flexibility during endgame, but the aerial delivery isn’t too many less points, and is achievable if we combine the robot lift and acquisition lift functions, so moving Gompei wasn't a primary goal of ours.

Section 4- Preliminary Design

Initially, we analyzed various lifting grabbing methods we had learned of in class, online, or anywhere else. A rack and pinion was a method we heavily considered using, due to it's simplicity in design. In fact, our first design used a rack and pinion for both the lift and the acquisition, as shown in figure 2 below.

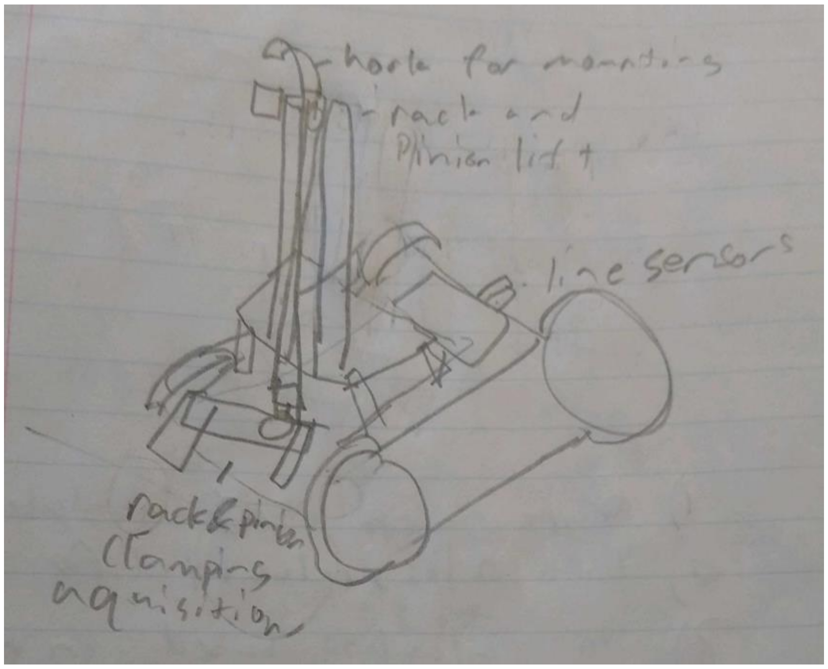


Figure 2- An early blueprint for an alternate design

Although simple, rack and pinions tend to lose a lot of power to friction, and the kit we were given didn't have enough resources to make multiple rack and pinions of considerable length. For our lift, we also considered a chain system, however decided against it since chains can break and may be a pain to conduct maintenance on and assemble. We eventually landed on a four-bar design for our lift mechanism, after a lot of consideration and experimentation. It is not as difficult to set up as a chain and sprocket lift, while also typically being efficient in power transmission. Although not capable of reaching the floor, we still planned on being able to put pizzas in the floor of the bottom dorm by pushing them in using the drive train. Picking up pizzas of the floor was low on our priority list, so it was a sacrifice we were willing to make in exchange for simplicity and consistency. In order to decide

For our acquisition, less power is needed to clamp down on a pizza than to lift a robot, so a rack and pinion may have been viable. However, when experimenting with dropping pizzas out of C channel, we found that sometimes the pizzas would get stuck if only one C channel moves. Because of that, we wanted a design that would allow both C channel bars to clamp down, and back off. We then landed on a design that would use a motor to rotate the bars, and a system of chains and gears shown in figure 3 below to rotate both grabbing arms simultaneously from one motor.

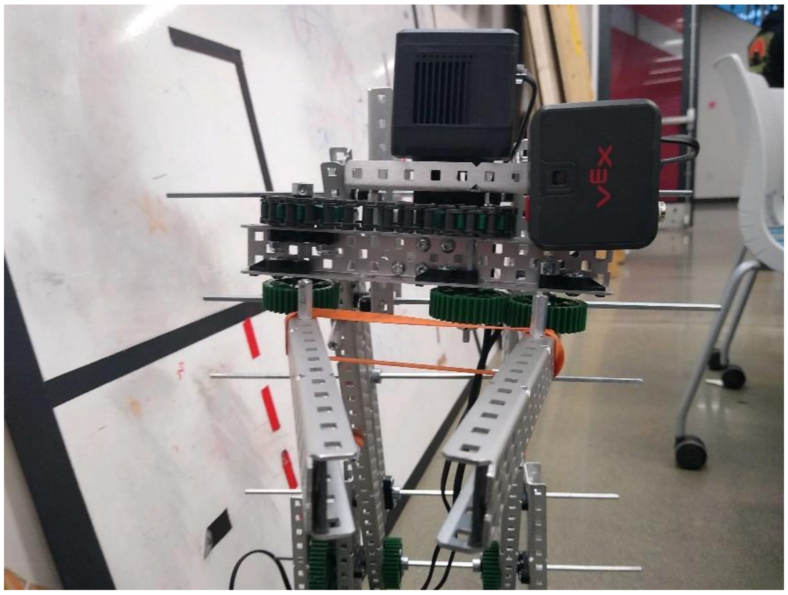


Figure 3- Front view of acquisition mechanism

Section 5- Selection of Final Design

The acquisition method we initially created worked quite well, and we were very satisfied with its consistency. Regardless, we still decided put some finishing touches on it. Although the grabber didn’t seem to drop the pizza when driving around, in order to ensure that it would not we added a rubber band between the arms. This made the arms naturally close in on the pizza, creating even more friction force. We experimented with different locations to place the rubber band at, because if placed at the tip of the arms, it would prevent them from fully opening, but putting them too close would reduce the added friction force. In order to further increase friction, we added material inside the C channel that was very sticky compared to the aluminum. The dark gray slides shown in figure 3 significantly improved the friction that the pizza had with the aquistion arms. Overall, this design was very consistent, and functioned as intended.

For our lift, we tested out multiple designs in order to decide on a final one. Our first build was a rack and pinion, as originally planned. It worked well, but wasn’t able to reach the very highest floor of the dorms. The rack and pinion also had a few other issues. The most important one is that whenever force was applied to the motor at an angle, the motor would stall and the mechanism would stop working. This is because applying force that isn’t directly parallel with the rack causes an increase in the force on it’s bearing, and therefore an increase in friction force. Because the lift wasn’t directly over the robots center of mass, it would oftentimes get stuck while lifting itself. It is possible that we may have been able to fix this issue by better positioning the lift, however when we tried out a four-bar mechanism, it worked very well.

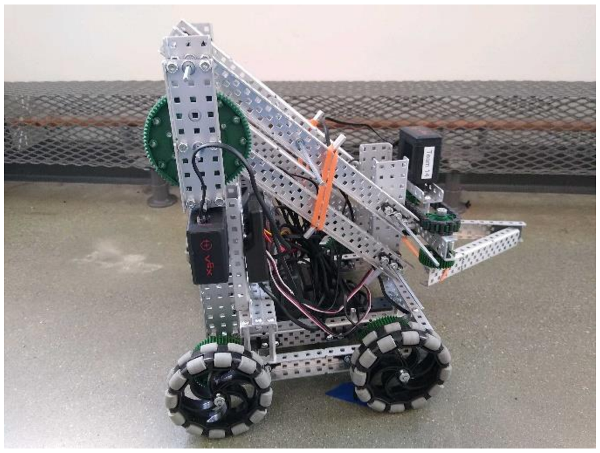
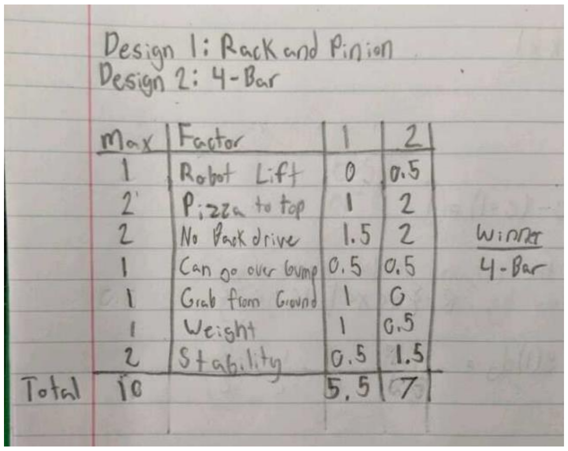


Figure 4- side view of the robot, demonstrating the lift mechanism

Our four-bar led to the center of weight of the robot being closer to it's middle, so it seemed to be more stable than the rack and pinion design, which had the majority of the weight on one side of the robot. It also had more reach than the rack and pinion, being just capable of reaching the highest dorm. We set up the mechanism with a very high gear ratio, which will be further explained in our mechanical analysis. When first controlling the arm with this powerful ratio, it still moved fast enough that it didn't seem to be a major hinderance, while also being capable of lifting up our robot without backdriving. Overall, it was a very effective lift mechanism for both the aerial delivery and delivering pizzas. It wasn't capable of picking pizzas up from the floor, but that was a sacrifice we could make in exchange for it's consistency and strength.



Design 1: Rack and Pinion
Design 2: 4-Bar

	Max	Factor	1	2
	1	Robot Lift	0	0.5
	2	Pizza to top	1	2
	2	No Back drive	1.5	2
	1	Can go over bump	0.5	0.5
	1	Grab from Ground	1	0
	1	Weight	1	0.5
	2	Stability	0.5	1.5
Total	10		5.5	7

Winner
4-Bar

Figure 5- decision matrix for the robot

In order to compare the two designs, we ended up creating a decision matrix, shown in figure 5. The four-bar lift came out on top, due to a couple of key factors. The fact that it's center of mass was in the middle of the robot opposed to the edge gave it a lead in stability. The extra height that it could reach gave it better delivery capabilities, and the robot lift's strength gave even more of a lead over the rack and pinion. Those factors were all things we decided were very important and helped finalize our decision in a four-bar mechanism.

Section 6- Final Design Analysis

Section 6.1- Mechanical Analysis:

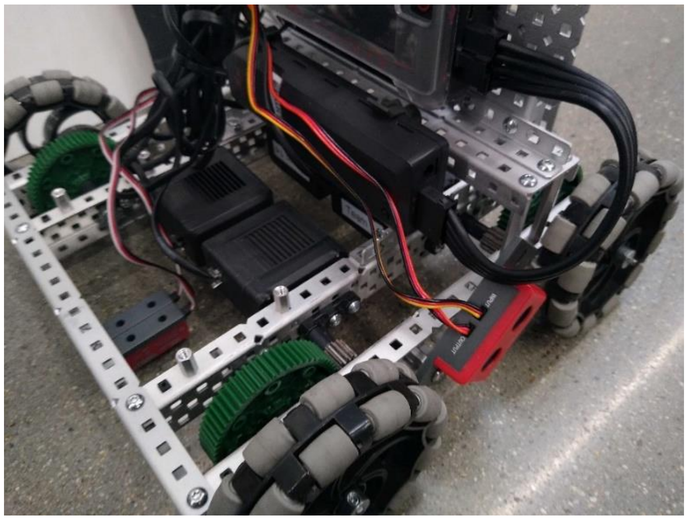


Figure 6- close-up of the drive train

Our drive train used four 4-inch diameter omni wheels, each powered by their own respective motor, operating at a gear ratio of 1 to 5, making it very powerful base. The robot ended up weighing about 9.1 pounds, a little more than our predicted 9 pounds. We didn't intend to make a particularly fast robot, because the distance between the dorms and the bakery is quite small. Since the distance the robot needed to travel wasn't very high, we felt comfortable with a slower robot, as long as the drive was consistent and precise.

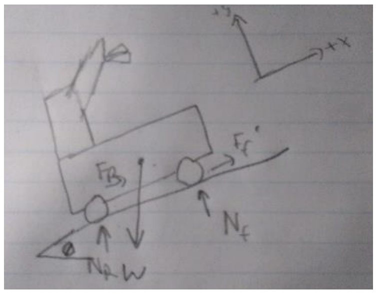


Figure 7- Free Body Diagram of the robot climbing a ramp

To calculate the traction the robot needs to get up the ramp, we used the free body diagram in figure 7. Assuming it's at a constant velocity, the free body diagram shows that $\sum F_x = 0 = F_f + F_r - W * \sin\theta$. The ramp is approximately at a 30-degree incline, allowing us to substitute in values to find total friction, $F_t = 9.11\ lbs * \sin 30$, and $F_t = 4.56\ lbs$. Dividing that upon the four motors, a typical load for the motors is 1.14 lbs, and at a radius of 2 inches. Using $\tau = F * r$, we get $\tau = 1.14 * 2$, and $\tau =$

2.28 in*lbs. The motors need 2.28 inch pounds of torque to climb the ramp each. The stall torque on the motors is 1.05 N*m, or 9.29 in*lbs, and when multiplied by the 5 to 1 gear ratio, and a transmission efficiency of .9, they should only stall if $\tau = 9.29 * 5 * .9 = 41.8$ in*lbs are required. Even if only two motors are generating the friction, they will need at most 4.56 in*lbs of torque if the robot is just about to tip. That is only about 11% of the stall torque, so the motors will have plenty of traction to climb the ramp.

To find the speed of the drive train on the ramp, we can use the equation $\omega = \omega_{freeload} * \left(1 - \frac{T}{T_{stall}}\right)$. Plugging values in from the vex motors data, and the typical required torque of each motor, $\omega = 200\ RPM * \left(1 - \frac{2.28\ in*lbs}{41.8\ in*lbs}\right)$, $\omega = 189\ RPM$, which is almost full speed. When accounting for e_s, 1/5, we find the RPM to be 38. To find the linear speed, we multiply the angular speed of the wheel in radians, 38 revs/min * $2\pi \frac{rad}{rev} * \frac{1\ min}{60\ seconds} = 3.96 \frac{rad}{sec}$, by the circumference of the wheel, and find $v = \omega * r = 3.96\ rad/sec * 2\ in = 7.9 \frac{in}{sec}$.

To find the speed when driving horizontally, we use the free body diagram in figure 8. The force the motors exert is equal to the force of friction caused by the robots weight, so the equation $\sum F_x = 0 = F_m - F_f$, and also $\sum F_y = 0 = W - F_n$, so $F_f = F_n * \mu = W * \mu$. Experimentally, we determined the coefficient of friction of the carpet on omni wheels to be .75, so $F_f = 9.1 * .75$ = 6.83 lbs, or 1.71 lbs/motor at a radius of 2 inches, using $\tau = F * r$, we find torque to be 1.71 * 2 = 3.42 in*lbs. Once again, using $\omega = \omega_{freeload} * \left(1 - \frac{T}{T_{stall}}\right)$, we find $\omega = 200\ RPM * \left(1 - \frac{3.42\ in*lbs}{41.8\ in*lbs}\right)$, $\omega = 184\ RPM$, which when multiplied

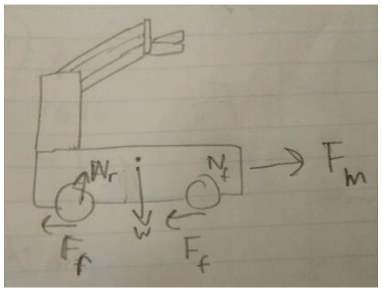


Figure 8- Free Body Diagram of robot moving horizontally

by e_s is 36.7 revs/min. Converted to radians/second it is 36.7 * $2\pi \frac{rad}{rev} * \frac{1\ min}{60\ seconds} = 3.85 \frac{rad}{sec}$. The velocity of the wheels should equal $v = \omega * r = 3.85\ rad/sec * 2\ in = 7.7 \frac{in}{sec}$, very similar to it's speed on the ramp.

Our four-bar lift mechanism used 12 inch long bars to get enough reach to deliver pizzas to the top shelf, 21 inches off of the ground. It weighed approximately 1.2 pounds with a center of mass around 13 inches for the point of rotation, since the acquisition and its motor was the majority of the weight. In order to power the lift, we used a 21 to 1 gear ratio, which allowed us to climb with it as well. In fact, we were the only team to perform an aerial delivery during the OED competition, because our gear ratio was strong enough to hold the robot up without back driving, even when power to the motor was cut off.

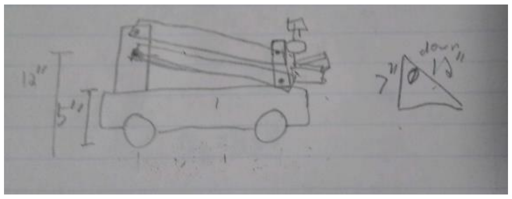


Figure 9- diagram of the lift at its lowest point

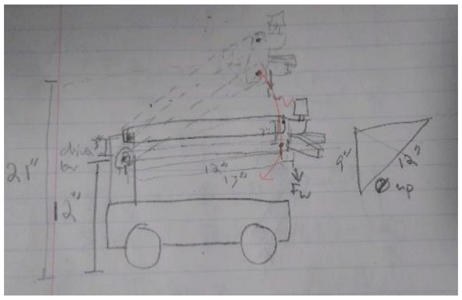


Figure 10- Diagram of the lift moving from its highest point to straight forward

Figures 9 and 10 show the range of the kinematic motion of our four-bar mechanism, which can bring the grabber of our acquisition between 21 and 5 inches. The diagram also shows how far the weight of the arm is applied from the motor. The arm weighs 1.2 pounds, and the force acts 13 inches from the pivot point, meaning that $\tau = F * r = 1.2 * 13 = 15.6$ in*lbs, therefore 15.6 in*lbs of torque are required to move the arm. The motor uses a two-stage transmission, a 12 to 84 gear ratio, and a 12 to 36 gear ratio. To find the e_s, multiply the gear

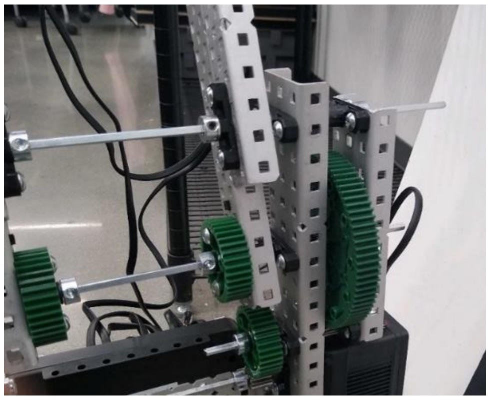


Figure 11- Gear transmission of the four-bar

ratios, $\frac{12}{84} * \frac{12}{36} = \frac{1}{21} = e_s$. To find the output torque, $\tau_{out} = \frac{\tau_{in} * n^2}{e_s}$. Assuming the input torque is the stall torque of the motor given by Vex, 9.29 in*lbs, and efficiency is 90%, $\tau_{out} = 9.29 * .9^2 * 21 = 158$ in*lbs. This is significantly more than the 15.6 in*lbs required to lift the four-bar.

To calculate the torque required for the aerial delivery, we simply apply the weight of the robot at the location of the lift mechanism, which is 12 inches from the motor, and on the four-bar mechanism. The torque the motor needs to apply to lift the robot is then $\tau = F * r = 9.1 * 12 = 109$ in*lbs. This is about 69% of the motors stall torque, which is quite high for continuous operation, but should work fine for short tasks like lifting the robot. In fact, in practice, it could lift the robot and preventing the motor from back driving even when power to the motor was cut off, proving the 21 to 1 gear ratio to be very valuable.

To calculate the speed of the delivery mechanism, we can once again use the equation $\omega = \omega_{freeload} * \left(1 - \frac{T}{T_{stall}}\right)$. To find it's speed when delivering pizzas and during regular use, we use the torque found that is required to lift the four-bar and acquisition. Plugging values in, we get $\omega = 200\ RPM * \left(1 - \frac{15.6\ in*lbs}{158\ in*lbs}\right) = 180.3\ RPM$, and including the gear ratio found previously, we find the arm to spin at $180.3\ RPM *\ e_s = 180.3 * \frac{1}{21} = 8.58\ RPM$. Converting this to radians per second, we find 8.58 revs/min * $2\pi \frac{rad}{rev} * \frac{1\ min}{60\ seconds} = .90 \frac{rad}{sec}$. To find the velocity of the acquisition as the motor drives it, use the equation $v = \omega * r = .9 \frac{rad}{sec} * 12\ in = 10.8 \frac{in}{sec}$.wever, it may be more important to find the time it takes for the arm to go from its bottom height to its top height. Going back to figures 9 and 10, the maximum angle the arm reaches is $\sin^{-1}\frac{9}{12} = 49°$ above the x axis, while going as low as $\sin^{-1}\frac{7}{12} = 35°$ below the x axis. The range of motion is then 84°, and the arm travels across it at .9 rad/sec, which can be converted to degrees per second, $\omega = \ .9\,\frac{rad}{second} * 360\,\frac{degrees}{2\pi\, rad}$ = 51.6 degrees/second. This allows it to go from minimum to maximum height in $t = \frac{84°}{51.6°/sec} = 1.63\ seconds$. The time isn't negligible, but it's short enough that it won't hold the robot back significantly in competition considering there is 30 seconds for autonomous and 2 minutes for remote control.

Section 6.2- Electrical Analysis

Before we began writing code for the autonomous program of the robot, our group first met up and discussed which sensors we wanted to use. Line trackers proved to be extremely helpful because of the numerous straight lines on the course, which we determined we could use both for following and as markers for when to preform or end certain functions. We considered also using a line tracker for ascending the ramp but decided that a ultrasonic sensor would be more apt for the situation, as it could detect the top of the ramp reliably, which line trackers might have struggled with as they would have been going from a black strip of tape to a close to black table surface. We also experimented with a bump switch to determine when the arm was at its lowest setting so that we could easily determine its location without having to reset the arm to a known height before each match. We would have preferred to use a limit switch for this purpose, and the bump switch proved too hard for the robot to reliably trigger so we removed it from the final design. One of the most important sensors on our robot was the camera, which we used track the size and position of the tape strips, which were mounted on both the dorm halls and the pizzeria.

We mounted the line followers to the underside of the front of the base as shown in *Figure 12*, as we discovered during lab one that having line followers closer to the direction the robot was driving in made a line tracking function considerably smoother. We initially struggled to mount the line follower to the side of the base because we had less room available than we had in the past, due to our switch to four-wheel drive. Eventually we found a way to mount it to the outermost left C-channel using 3/4th inch standoffs, as shown in *Figure 13*. This ensured that it was at a suitable height to see the retaining wall as we drove up the ramp. The camera was mounted sideways on the grabber mechanism so that it could be used to detect not only the horizontal but also vertical position of the arm. We ended up not using this feature as the encoder in the arm proved far more reliable once the arm had raised above the reflective tape.

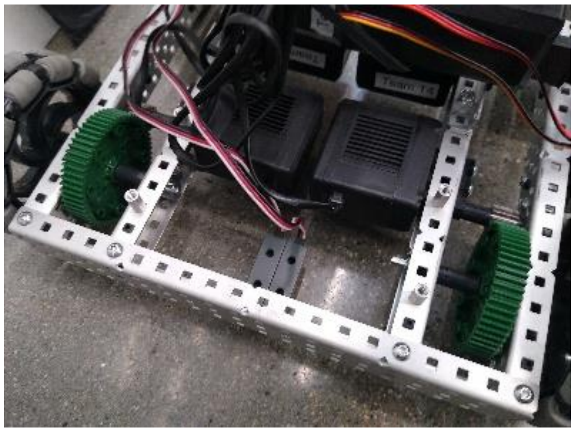


Figure 12- Photo of line sensors mounted to the base of the robot

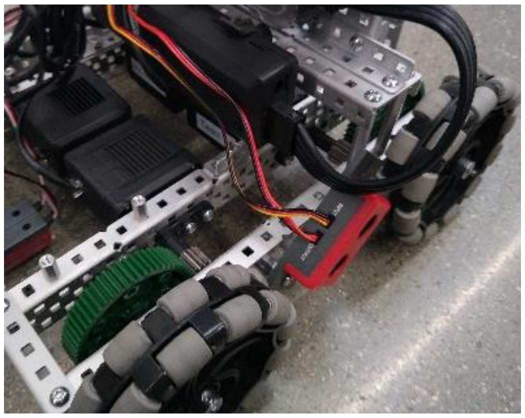


Figure 13- Photo of the distance sensor mounted to the side of the robot

Each of the vex V5 motors that we are using draws 11 watts of power when running continuously. If we take 11 watts for each of the six motors on our robot and divide it by the 12 volts available from our battery we have a maximum current draw of 5.5 Amps, assuming that the sensors have a negligible draw.

Section 6.3- Programming Analysis

One of our main goals during the writing of our autonomous code was to avoid having large areas of dead reckoning, as we discovered in the first lab that it was far from reliable over long periods. In order to avoid slowly losing our place with dead reckoning, we had a number of functions, such as DriveToLine, TurnToLine, and Center which would use landmarks on the field to ensure that even if the robot went slightly off course, it would be able to self-correct, at least to a degree. Because we were unconfident in our designs ability to make it over the bump into the construction zone wed decided to instead focus on delivering two pizzas into separate floors of faraday hall. To begin we spent time compiling useful functions that we had already written for other labs, such as Drive and LineTrack, and modifying others, such as the Center Function. This was actually one of the most challenging to write but ended up being one of our most efficient functions, that managed to center the robot on both the pizzeria and the faraday dorms reliably and quickly as long as the reflective tape was within frame. Our autonomous program used a combination of sensing distance, detecting lines, and vision processing to remain on course and efficient in its navigation of the playing field. We had two main programs: one fore the autonomous section and one for the remote control. The reason that these were separate was because in order to map the robot onto the remote control in a way that felt natural to drive we needed to define a drive train in the code, as opposed to the two separate motor groups that we found more elegant when coding proportionally controlled movements in the autonomous code. Within the remote control program we also implemented the use of the buttons to have preset heights for the arm, which saved a huge amount of time that would have been spent trying to raise the arm to specific locations. In our code we used not only all of the sensors that we attached but also the encoders built into the Vex V5 motors. These proved to be extremely helpful in accurately moving the height of the arm, as well as toggling the state of the grabber.

Section 7- Summary/Evaluation

Our robot preformed extremely well in the CDR. Although it veered slightly off course after turning too far during the initial TurnToLine function it was able to quickly correct itself during the LineTrack function. It executed the entire autonomous code in just under fifty seconds on the first attempt, scoring both pizzas that we intended it too. It had no issue ascending the ramp due to the gear ratio we used in the drivetrain and our arm’s resistance to being back driven contributed to us being the only team that managed to complete an arial delivery after power was cut in the OED. Though we initially messed up in the remote control portion of the CDR after dropping a pizza in front of the dorm building and not being able to push it into the bottom floor, on our second attempt we not only completed a happy dorm but also executed an arial delivery, easily earning that maximum points available in the CDR.

We would state that our robot was a success. It executed every task that we determined was a high priority and earned us full points on the CDR, in addition to doing well in the OED. It was within the weight and size constraints defined by the project. It could reliably move pizzas from the pizzeria to the top three floors of the dorm, as well as the bottom floor with slightly less reliability. It could also ascend the ramp without any tipping or sliding and complete an arial delivery even without power, a feat that no other team achieved.

Section 8- Appendix

Section 8.1- Autonomous Program:

```
/*----------------------------------------------------------------------------*/
/*                                                                            */
/*    Module:       main.cpp                                                  */
/*    Author:       C:\Users\bryce                                            */
/*    Created:      Tue Nov 30 2021                                           */
/*    Description:  RBE 1001 Final Project                                    */
/*                                                                            */
/*----------------------------------------------------------------------------*/

// ---- START VEXCODE CONFIGURED DEVICES ----
// Robot Configuration:
// [Name]               [Type]        [Port(s)]
// RangeFinder          sonar         A, B
// LeftLine             line          E
// RightLine            line          F
// Arm                  motor         8
// Grabber              motor         5
// Camera               vision        7
// LeftMotors           motor_group   1, 4
// RightMotors          motor_group   2, 3
// Bump                 bumper        C
// ---- END VEXCODE CONFIGURED DEVICES ----

#include "vex.h"
using namespace vex;

// KP constants
const float Kp = 2;
const float Kp2 = 0.5;
const float KpX = 1.1;
const float KpW = 4;

// Camera Variables
int xcord = 0;
int width = 0;
```

```cpp
// Turn function, + is left turn, - is right turn
void Turn(double Degree) {
  double rotationDegrees = Degree * 5 * 12 / 4.125;
  RightMotors.setVelocity(100, percent);
  LeftMotors.setVelocity(100, percent);
  RightMotors.spinFor(forward, rotationDegrees, degrees, false);
  LeftMotors.spinFor(reverse, rotationDegrees, degrees, true);
}

// Basic blind drive function based on set distance
void Drive(float inches) {
  float rotationDegrees = 27 * inches * 5;
  RightMotors.setPosition(0, degrees);
  LeftMotors.setPosition(0, degrees);
  LeftMotors.rotateFor(rotationDegrees, rotationUnits::deg, 200,
                       velocityUnits::rpm, false);
  RightMotors.rotateFor(rotationDegrees, rotationUnits::deg, 200,
                        velocityUnits::rpm, true);
}

// Line Following Function based on time
void LineTrack(int loopCount) {
  while (loopCount > 0) {
    int left = LeftLine.reflectivity();
    int right = RightLine.reflectivity();
    int error = left - right;
    LeftMotors.setVelocity(75 - error * Kp2, percent);
    RightMotors.setVelocity(75 + error * Kp2, percent);
    LeftMotors.spin(forward);
    RightMotors.spin(forward);
    loopCount--;
    wait(0.1, seconds);
  }
}

// Camera centering
void Center(float height) {
  while (true) {
    Camera.takeSnapshot(Camera__TAPE);
    if (Camera.objectCount > 0) {
      printf("x: %d, y %d\n", Camera.largestObject.centerX,
             Camera.largestObject.centerY);
      // Takes important dimensions of the object (inverted because the camera
      // is sideways)
      xcord = Camera.largestObject.centerY;
      width = Camera.largestObject.height;
      // Moves the motors based on the location and size of the reflective
      // strips
      float errorX = 95 - xcord;
      float errorW = 55 - width;
      LeftMotors.setVelocity(0 + (errorX * KpX + errorW * KpW), percent);
      RightMotors.setVelocity(0 + (-errorX * KpX + errorW * KpW), percent);
      LeftMotors.spin(forward);
      RightMotors.spin(forward);
      if (errorX < 10 && errorX > -10 && errorW < 5 && errorW > -5) {
        LeftMotors.setVelocity(0, percent);
        RightMotors.setVelocity(0, percent);
        break;
      }
    }
  }
  // Raises the arm to a given height
  Arm.setVelocity(100, percent);
  Arm.spinToPosition(height, turns);
}

// Distance sensor function which goes until the top of the ramp
void Dist() {
  while (RangeFinder.distance(inches) < 15) {
    float dist = RangeFinder.distance(inches);
    float error = 6 - dist;
    Brain.Screen.clearScreen();
    LeftMotors.setVelocity(95 + error * Kp, percent);
    RightMotors.setVelocity(95 - error * Kp, percent);
    LeftMotors.spin(forward);
    RightMotors.spin(forward);
  }
}

// Turn till a line is found function, + is left turn, - is right turn
void TurnToLine(int toggle) {
  RightMotors.setVelocity(70 * toggle, percent);
  LeftMotors.setVelocity(-70 * toggle, percent);
  RightMotors.spin(forward);
  LeftMotors.spin(forward);
  while (true) {
    if (LeftLine.reflectivity() > 18) {
      RightMotors.setVelocity(0, percent);
      LeftMotors.setVelocity(0, percent);
      break;
    }
  }
}

// Drive till a line is found +toggle is forward, -toggle is backwards
void DriveToLine(int toggle) {
  RightMotors.setVelocity(100 * toggle, percent);
  LeftMotors.setVelocity(100 * toggle, percent);
  RightMotors.spin(forward);
  LeftMotors.spin(forward);
  while (true) {
    if (LeftLine.reflectivity() > 20) {
      RightMotors.setVelocity(0, percent);
      LeftMotors.setVelocity(0, percent);
      break;
    }
  }
}

int main() {
  vexcodeInit();

  //***************** TWO PIZZA PLAN *********************\\
  //Points Possible: Pizza1 (6) + Pizza2 (6) + Both (5) + Multpi-Auto (2) = 19
  // Pros: If we mess up its till 8 points, no bump needed
  // Cons: more than twice max points, time

  // Go to top of ramp and prep/lower the arm
  Arm.setPosition(0, degrees);
  Grabber.setPosition(0, degrees);
  Arm.spinToPosition(6.2, turns, false);
  wait(1.75, seconds);
  Dist();

  // Drive to pizzeria
  DriveToLine(1);
  Drive(3);
  TurnToLine(1);
  Arm.spinToPosition(4.5, turns, false);
  LineTrack(38);
  // Deliver First Pizza
  Turn(90);
  Drive(37);
  Turn(90);
  Center(3.7);
  Drive(10);
  Grabber.spinToPosition(45, degrees);
  // Return to pizzeria and grab pizza
  DriveToLine(-1);
  Drive(3);
  TurnToLine(1);
  Arm.spinToPosition(5, turns, false);
  LineTrack(5);
  Grabber.spinToPosition(0, degrees, false);
  Center(4);
  Drive(8);
  wait(1, seconds);
  // Deliver second pizza
  Drive(-42);
  Arm.spinToPosition(5.2, turns, false);
  Turn(-90);
  Center(5.2);
  Drive(10);
  Grabber.spinToPosition(45, degrees);
  Drive(-8);
}
```

Section 8.2- Remote Control Program:

```
/*----------------------------------------------------------------------------*/
/*                                                                            */
/*    Module:       main.cpp                                                  */
/*    Author:       C:\Users\bryce                                            */
/*    Created:      Fri Dec 03 2021                                           */
/*    Description:  V5 project                                                */
/*                                                                            */
/*----------------------------------------------------------------------------*/

// ---- START VEXCODE CONFIGURED DEVICES ----
// Robot Configuration:
// [Name]               [Type]        [Port(s)]
// Drivetrain           drivetrain    1, 4, 2, 3
// Grabber              motor         5
// Arm                  motor         8
// Controller1          controller
// ---- END VEXCODE CONFIGURED DEVICES ----
```

```cpp
#include "vex.h"

using namespace vex;

int main() {
  // Initializing Robot Configuration. DO NOT REMOVE!
  vexcodeInit();
  Drivetrain.setDriveVelocity(100, percent);
  // Prep arm and Grabber
  Arm.setPosition(0, degrees);
  Grabber.setPosition(0, degrees);
  Arm.setVelocity(70, percent);
  Grabber.setVelocity(60, percent);
  // Pre-prepared Arm heights and grabber positions to make reaching certain
  // floors easier
  while (true) {
    // Bring Arm to pizzeria height
    if (Controller1.ButtonUp.pressing()) {
      Arm.spinToPosition(4, turns, false);
    }
    // Bring Arm to highest position
    if (Controller1.ButtonDown.pressing()) {
      Arm.spinToPosition(0, turns, false);
    }
    // Bring Arm to 3rd floor height
    if (Controller1.ButtonLeft.pressing()) {
      Arm.spinToPosition(3.7, turns, false);
    }
    // Bring Arm to 2nd floor height
    if (Controller1.ButtonRight.pressing()) {
      Arm.spinToPosition(5.2, turns, false);
    }
    // Opens Grabber
    if (Controller1.ButtonX.pressing()) {
      Grabber.spinToPosition(45, degrees, false);
    }
    // Closes Grabber
    if (Controller1.ButtonY.pressing()) {
      Grabber.spinToPosition(0, degrees, false);
    }
  }
}
```