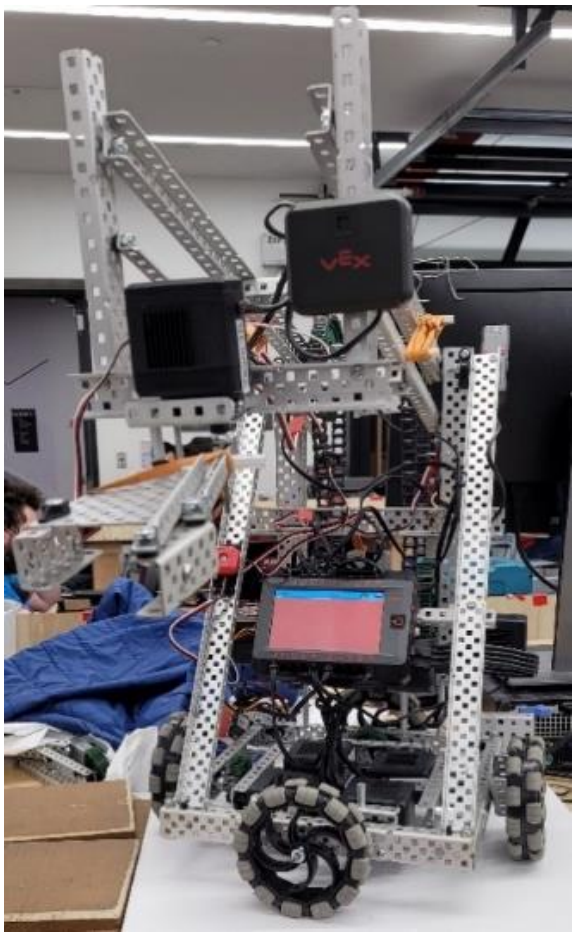
RBE 1001 B21, *Introduction to Robotics*
Final Project Report



"The R.A.D (Robot Automated Delivery) bot"

Shivangi Sirsiwal, Cameron Wian, Jai Jariwala

Table of Contents

List of Figures

Introduction:

There's nothing WPI students love quite like pizza and robots. What could be better for our campus community than combining the two? This beautiful concept is not without challenges, though, as a robot aiming to complete such a heroic mission as delivering pizzas to WPI dorms must brave challenges such as climbing the hill, collecting pizzas from various locations, making sure pizzas are properly delivered to various floors in the dorm buildings, driving over speed bumps guarding the 86 Prescott Street construction site, and even holding itself in the air to complete a coveted aerial delivery. These difficulties are simulated on the challenge field, and our robot was created to overcome these simulated challenges, perhaps as a first step toward a glorious future union of pizza and robots for WPI students to enjoy for years.

Preliminary Discussions:

In order to decide what challenges to attempt, we created a table ranking the number of points earned by each challenge and the difficulty of each challenge, then ranked the challenges in order from highest score to lowest.

Figure B:

Challenge	Number of Points	Challenge Difficulty (Easiest [10] - Hardest [0])	Totals
Gompei	35	8	43
Aerial Delivery	35	0	35
Happy Dorm	30	5	35
Auto Pizza deilvery Farady floor 2	12	4	16
Auto Pizza deilvery Farady floor 3	12	4	16
Tele Op delivery Farady floor 2	6	8	14
Tele Op delivery Farady floor 3	6	8	14
Tele Op delivery Messenger floor 2	6	8	14
Tele Op delivery Messenger floor 3	6	8	14
Tele Op delivery Messenger floor 4	6	8	14
Tele Op delivery Messenger floor 5	6	8	14
Tele Op delivery Farady floor 4	2	10	12
Auto Pizza deilvery Farady floor 1	4	7	11
Tele Op delivery Farady floor 1	2	9	11
Tele Op delivery Messenger floor 1	2	9	11
Auto Pizza deilvery Farady floors 4	4	6	10
Auto enter into Construction zone	5	2	7

Table ranking challenges based on their point value and difficulty.

Using the information from this table we developed our game strategy. For the AUTO period, we elected to focus on delivering pizzas to floors 2 and 3 of the FARADAY building, as this seems doable and results in the most points for the AUTO period. For the TELE period we decided to focus on achieving a HAPPY DORM in FARADAY, as this results in a high number of points and is relatively easy should the autonomous placement of the two pizzas in FARADAY be successful. For the ENDGAME period we decided to make our top priority AERIAL DELIVERY as this results in the highest number of points, but we determined it could be useful to also be able to get GOMPEI into our COALITION's zone in case both AERIAL DELIVERY are occupied.

Problem Statement:

Design a robot to hold, transport, and deliver pizzas to various floors of dorm buildings.

<u>Design Specifications:</u>

- High Priority (goals that are essential to a robot design our team would qualify as successful)
  - A strong, stable chassis with a drivetrain that can change direction quickly and efficiently
  - A pizza grabber that can hold pizzas securely
  - A lift mechanism that can lift and lower the grabber mechanism from the top to bottom floors of the DORMS
  - Ramp climbing
  - Weight under 10 lbs.
  - Fits with 15.25'' by 15.25'' perimeter
- Medium Priority (design goals that serve to maximize our competition score)
  - Autonomous delivery to the 2nd and 3rd floors of FARADAY
  - Some sort of hooking mechanism that can be attached to our lift for AERIAL DELIVERY
  - Autonomous ramp climbing
  - A way to push GOMPEI into our COLATION's zone
- Low Priority (design goals that would be nice to have but are not essential to what our team considers a competitive robot)
  - A way to cross the SPEED BUMPS into the CONSTRUCTION ZONE
  - Autonomous delivery to other floors of FARADAY
  - Autonomous delivery to MESSENGER
  - Large enough range with the lift mechanism to pick up pizzas off the ground

Preliminary Designs:

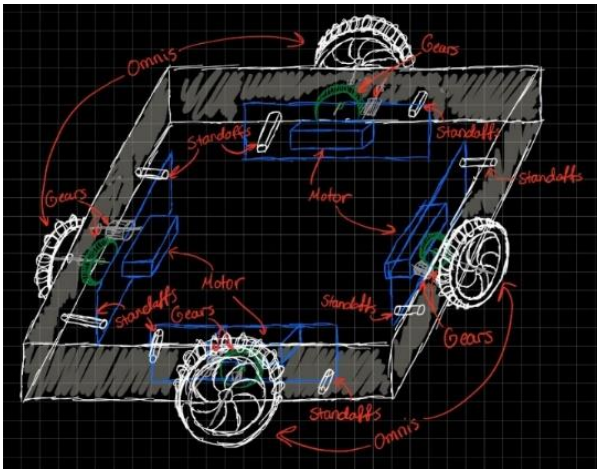
We first decided to design the chassis. We had to first discuss what wheel type to use, and after some deliberation, we decided to use omni wheels as opposed to traction wheels so as to widen our range of motion and ability to change directions quickly. We created a chassis design with wheels on all four sides, as depicted in the sketch in figure C1.

We next brainstormed ways to pick up and hold pizzas. We had two main ideas – the roller design (see figure C2) and the grabber design (see figure C3). The roller design has the advantage of being able to collect pizzas from any angle (thanks to the wide roller) and hold them very securely (thanks to the traction from the rubber bands). The roller design has the disadvantage of being very wide and less controlled, as the roller shoots the pizzas into the dorms. The grabber design has the advantage of having a lower reach, meaning this deign would make it considerably easier to pick up pizzas from the ground, and it also has more control over the delivery of pizzas since it places the pizzas exactly. We decided to prototype both designs to get a better idea of which design would be more practical to execute and which one is more efficient at its job. After creating the prototypes and seeing how they interact with the various field elements, we decided to go with the grabber mechanism as the control made the mechanism more consistent. We then created a rendition of the grabber mechanism that lowered grabber arms from above the pizza as opposed to the sides (see figure C4 for image), but this design had issues with maneuverability in the dorms. So, we decided to switch to a grabber that approaches the pizza from the sides (as initially depicted in the figure C4 sketch). In order to automate this mechanism, we decided we could mount a camera on the grabber mechanism in order to use the orange and yellow stripes on the DORMS to align the grabber

mechanisms with the dorms to deposit the pizzas and to align the grabber mechanism with the pizzeria to pick up a second pizza.

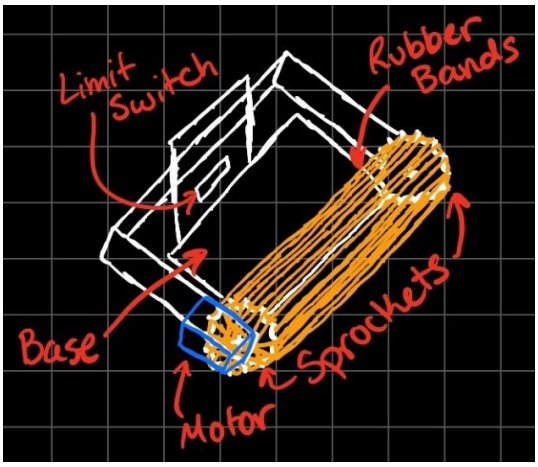
We then discussed our lift mechanism to and add height range to our pizza delivery mechanism and lift our robot during AERIAL DELIVERY. We first considered a two-stage rack and pinion lift design (see figure C5 for sketch), which would be able to lift and lower quickly and would also be relatively easy to manufacture and implement. However, rack and pinion is not a very strong or stable lift mechanism, and for our group it was imperative that our lift be strong as we were counting on it to support our whole robot. Next, we designed and prototyped a scissor lift (see figure C6 for image) and four-bar (see figure C7 for image). We decided against the scissor lift design because of its inefficiency and went with the four-bar because of its strength, stability, and consistency. The four-bar also allows for a large enough range of motion to allow the robot to pick up a pizza off the ground.

Figure C1:



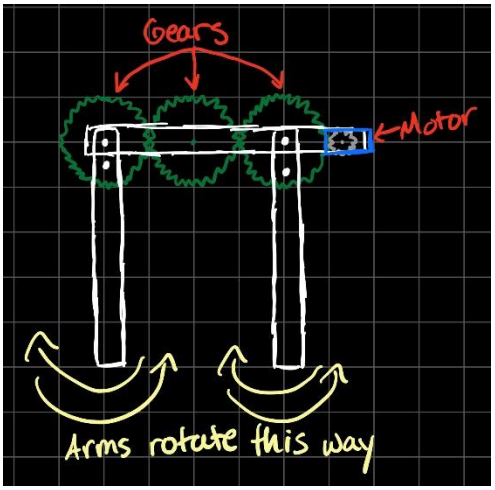
An initial sketch of our chassis design with four omni wheels on all four sides of the robot.

Figure C2:



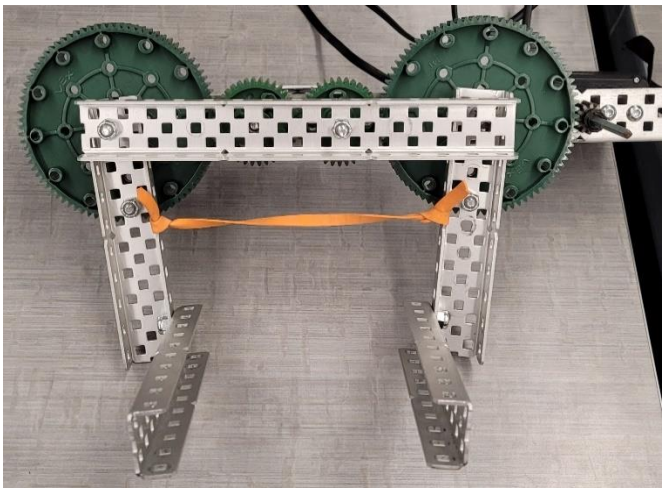
A sketch of the roller design with a motorized roller made with rubber bands and sprockets.

Figure C3:



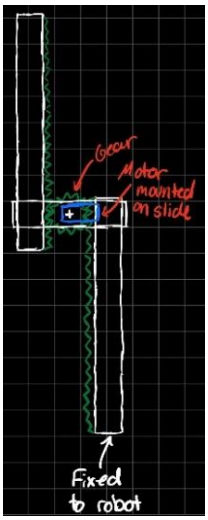
A sketch of the grabber design with two arms driven by a motorized geartrain.

Figure C4:



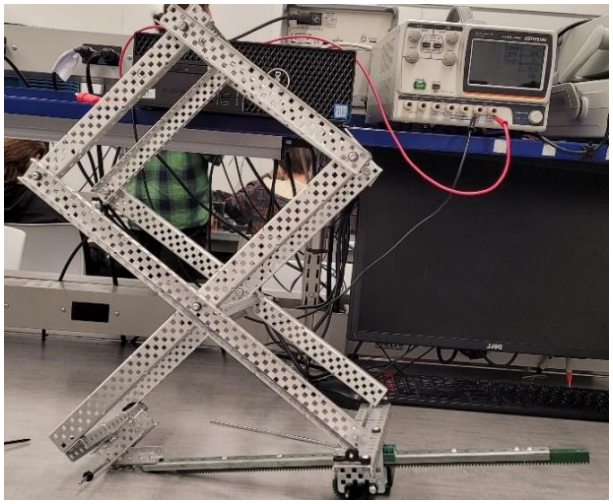
An image of our initial grabber design that approaches the pizzas from the sides.

Figure C5:



Sketch of our initial two-stage rack and pinion

Figure C6:



An image of our scissor lift prototype.

Figure C7:

An image of our four-bar lift mechanism.

Selection of Final Design:

Throughout the final design process, we had to make some compromises and changes to our initial designs in order to fulfill the design specifications we decided on during preliminary

discussions while also complying with the weight and perimeter restrictions. One of the largest changes we made was with our grabber design. After prototyping our preliminary grabber design and mounting it on our robot, we decided to weigh the robot and found it was a couple pounds above the weight limit, largely due to our nearly one-pound grabber design (see figure D1 for image). We eventually simplified the design down to a baseplate that holds the pizza from below and a motor that uses a small plate with a screw extending from it to force open one mobile arm that opens and closes around the pizza with the help of a rubber band, pushing the pizza in between a stationary arm and a mobile arm (see figure D2 for image and figures D3 & D4 for diagram). We also ended up adding a hook to the back of the four-bar in order to complete the AERIAL DELIVERY objective (see figure D4 for diagram) as opposed to building a separate mechanism to complete this task. We also found it made our AERIAL DELIVERIES more consistent when we extended the back bar out to the right and left so the two extensions press against the two FARADAY dorms and keep the robot from pitching backward and falling.

Figure D1:

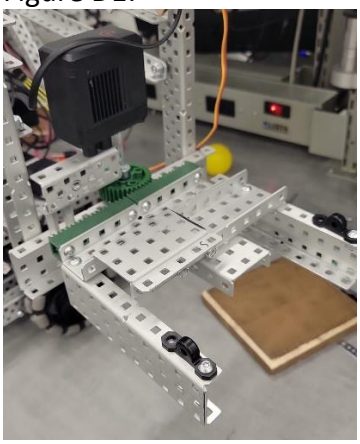


Image of our initial, nearly one-pound grabber mechanism.

Figure D2:

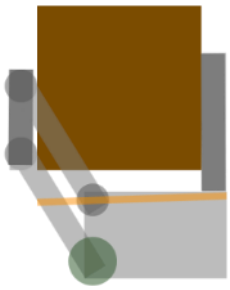
Image of our final, simplified, and significantly lighter grabber mechanism.

Figure D3:

Diagram of our final grabber mechanism.

Figure D4:

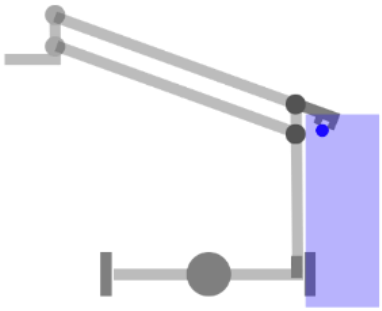
Gripper with PIZZA



A diagram showing our grabber mechanism when it has a pizza in it.

Figure D5:

Aerial Delivery



A diagram showing how our hook for AERIAL DELIVERY works.

Sensor Selection and Programming Implementation:

Sensors used:

- Vision (camera)
- Ultrasonic rangefinder
- IR reflectivity sensor ("line detector")
- Switches/buttons (limit switch and bump switch)
- Internal motor encoders (optical, high resolution)

Our primary focus when selecting sensors was reliability and type of information gathered by them. For this reason, we invested most of our time into working with the vision sensor (camera). The information gathered by the camera is far greater any other single sensor, and for this application, the very regulated operating conditions led to an acceptable level of reliability. By using color signatures from the vision targets present on the field (see figure D6), we were able to use just the one sensor to determine the approximate distance and offset of our robot from most of the points of interest on the field. Combined with our unique drivetrain, this gave us a distinct advantage in programming the autonomous section of the competition

code. Our robot's ability to move easily in any direction while maintaining a fixed orientation relative to the field allowed us to move in the general direction of a vision target and count on our proportional control loop to move the robot to nearly the exact same location every time the code was run, thus removing any potential error in the deduced reckoning code that may have occurred when traveling between targets. This allows all vision targets to be treated as a "reset point" preventing compounding of errors in positioning.

Another capability of the camera that we considered when choosing it was the possibility of automated pickup of pizzas from the field surface. Code was written to align the robot with a pizza below it, and a servo motor was utilized to change the camera position, controlled with a pwm_out object. Due to both a lack of the function's necessity and weight restrictions, this functionality was cut from the final design. However, proof of concept tests proved that it was certainly feasible and another advantage to focusing on the vision sensor.

Apart from the camera, we used both the ultrasonic range finder and the infrared "line detector" sensors. Due to design considerations like the chassis height requirements to climb the SPEED BUMP, we determined that line following sensors could not be mounted on the chassis close enough to the ground to be effective. We instead decided that it was simpler to utilize the ultrasonic sensor for robot alignment during the ramp climb. Another reason for this was the way our drivetrain handled the ramp. Due to the use of only two omni wheels for traction on the ramp, it tended to be jerky in motion, which could move the line sensors far enough to be ineffective quicker than could be corrected for. The ultrasonic sensor, being a range sensor, had no such issue with jerky movements and proved quite effective in helping course-correct after unpredictable motion using a proportional control loop. One major flaw,

however, was the tendency for the robot to be at an unpredictable angle when reaching the top of the ramp. Given more time, we hoped to correct for this problem using previously written code that recorded the travel distance of each motor during a proportional control loop and attempted to correct the final angle after it exited.

The infrared “line detector” sensor was utilized as a presence sensor, inspired by and similar to those found in many automatic dispenser type devices. Infrared sensors of this type use an IR LED to shine infrared light in front of them and measure the amount that bounces back with a photoresistor sensitive to IR light. This is used for line tracking because the reflectivity of a surface is related to its color, and therefore a line can be distinguished from the surface it is on. In our case, reflectivity was merely based on whether there was an object to reflect light back (see figure D7). This allowed for contactless sensing of a pizza in the gripper mechanism, reducing the required number of moving, consistent parts and the weight of the grabber mechanism overall. Our autonomous code makes use of this instead of a delay to make picking up from the PIZZERIA less affected by human error.

Finally, we used two switches for sensing. One, the limit switch, was used to sense the arm in its uppermost position. From that, we were able to use the motor encoders to set the arm consistently to predefined positions. The other, a “bump” switch (spring-loaded button requiring higher force to throw) was used to switch autonomous programs during testing and the CDR.

Figure D6:

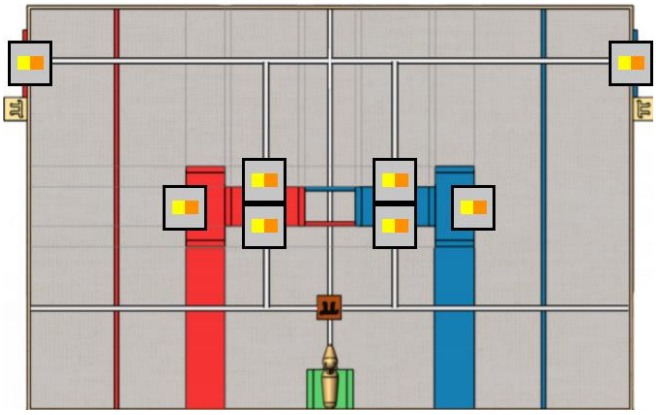
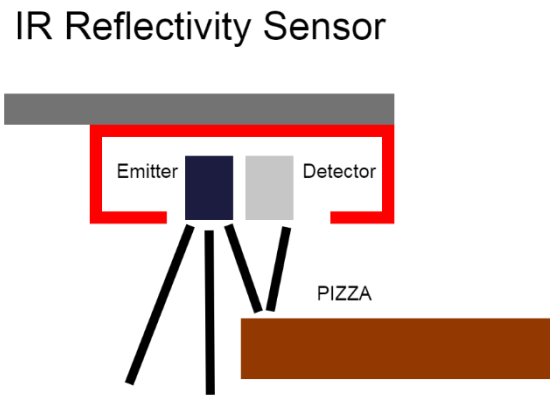


Diagram of challenge field with vision targets marked.

Figure D7:



IR Reflectivity Sensor sensor diagram.

Electrical System:

Limited documentation can be found easily on the VEX website, but from what is provided, it can be estimated that the motors can draw up to about 2A at the minimum rated battery voltage of around 10V. Assuming this is a reasonably close estimate, the battery can theoretically power 10 motors at 2A each for a total of 20A, the rated maximum output of the battery. Motors use the most power of any device, and our robot uses 6. Seeing as there are rarely more than 4 running at full speed (not necessarily full power), it is reasonable to assume our robot is within the limitations of the electrical system. Of course, testing confirmed this, as all of the devices have features built in to prevent them from operating at unacceptable levels of power. Based on a typical battery life of at least one hour of operation, and a battery rated for 1100mAh or 1.1Ah, it can be determined that our robot draws a maximum average of 1.1A for a typical usage session.

Specifications from documentation:

Brain: 12.8V

Battery: 1100mAh, 12.8V to 10V, 20A max

Motors: 11W max

Programming Methodology:

For our programming methodology and sensor integration, we tried to stick to principles of reliability, repeatability, and genericism. Especially for programming, it is important to make the process of doing an action as concise and general as possible, because this both removes room for error and allows for reuse of code. This is covered more in-depth in the commented code found in the Appendix and the Sensor Selection and Programming Implementation section.

Mechanical Analysis:

<u>Robot stability</u>

In order to find the center of gravity, we measured the weight of the robot from all sides and determined the coordinates by using the ratio of weights in accordance with the wheelbase of the robot.

Wheelbase = 10" x 10"
Weight of robot = 10.0 pounds
Weight of left side = 5.23 pounds
Weight on right side = 4.77 pounds
Weight in the front = 4.86 pounds
Weight in the rear = 5.14 pounds
Using ratio and proportion to find out the position of the center of gravity-

$$Distance\ of\ CG\ from\ left = \frac{4.77\ lbs}{10\ lbs} \times lenght\ of\ robot = \frac{4.77\ lbs}{10\ lbs} \times 10\ in = 4.77in$$

Repeating the formula for all sides we get:

D_x = 4.77 in from the left

D_y = 5.14 in from the front

D_x = 3.11 in from the bottom

Using these values, we calculated the angular stability:

$$Angle\ of\ stability = \tan^{-1}\frac{length\ from\ left}{length\ from\ bottom} = \tan^{-1}\frac{4.77}{3.11} = 56.9°$$

Repeating this formula for all sides we get an average angle of stability of 57.2 degrees

Arm design

We designed the arm to carry at least 4.5 pounds of weight as this was the weight of the arm itself along with the pizza collecting contraption. To accomplish this we needed to use moments and torque.

If the length of the arm is 17" and the weight of the load is at least 4.5 pounds:

$$torque = 17\ in\ \times\ 4.5\ lbs = 76.5\ in\ lbs$$

To find the needed gear ratio we have to divide the torque out by the torque in

$$\frac{torque\ out}{torque\ in} = \frac{76.5\ in\ lbs}{torque\ of\ motor} = \frac{76.5\ in\ lbs}{9.3\ in\ lbs} = 8.22{:}\ 1$$

We tested different ratios to get as close to this ideal ratio value as possible and ended up using a 24T to 30T chain and sprocket transmission and a 12T to 84T gear transmission.

With these values:

$$\frac{torque\ out}{torque\ in} = \frac{30}{24} \times \frac{84}{12} = 8.75{:}\ 1$$

This value was sufficient for the arm as it was more than the required torque but did not reduce the speed too much.

For arm speed:

$$Power = \frac{Torque \times Speed}{4}(of\ motor) = \frac{9.3 \times 200}{4} = 465W$$

Also

$$Power = \frac{Wieght \times Height}{time}$$

$$time = \frac{Wieght \times Height}{Power} = \frac{4.5\ lbs \times 20\ in}{465\ W} = 0.2\ s$$

In theory, our lift should take only 0.2 seconds to reach the 20 in mark, but this calculation was done for a lossless transmission. The actual robot had a low transmission efficiency and therefore took somewhere around 1.2- 1.5 seconds to reach the 20 in mark.

Robot Speed

The speed of the robot is fairly easy to calculate as it is:

$$v = \frac{r \cdot d \cdot i \cdot \pi}{o}$$

Where:

(r) is Revolutions per Minute = 200 rpm

(d) is Diameter of Wheel = 4 in

(i) is Reduction Input = 1

And (o) is Reduction Output = 3

Therefore

$$v = \frac{200 \cdot 4 \cdot 1 \cdot \pi}{3} = 837.33\frac{in}{min} = 0.354\frac{meter}{sec}$$

Summary/Evaluation:

During the CDR our design performed very well. It executed its autonomous functions perfectly, climbing the ramp and successfully placing two pizzas in FARADAY. During the TELE period we achieved a HAPPY DORM in FARADAY, and we picked up a pizza off the floor, and we attempted an AERIAL DELIVERY, but we were unsuccessful because we did not line up the robot properly.

During the OED our robot did not perform as well as we had hoped. This was due largely to the rushed modifications to our autonomous code as we had not realized that there would not be a ramp for the competition, and we had not adapted the code beforehand. We did not have time for proper testing on both sides of the field prior to the OED, leading to a lack of de-bugging before competing. The robot also underperformed due to a serious lack of driver's practice. This led to spending a lot of time trying to figure out how best to maneuver the bot around the field during the OED. Had there been more driving practice, we would have been able to spend less time fumbling with controls and more time scoring points.

All in all, this project was an incredibly enriching experience for our team. We learned much from our successes and even more from our failures, and we look forward to the rest of our robotics engineering journey here at WPI.

Appendix:

Code:

```
/*----------------------------------------------------------------------------*/
/* */
/* Module: main.cpp */
/* Author: C:\Users\camca */
/* Created: Thu Nov 18 2021 */
/* Description: V5 project */
/* */
/*----------------------------------------------------------------------------*/


// ---- START VEXCODE CONFIGURED DEVICES ----
// Robot Configuration:
// [Name] [Type] [Port(s)]
// Vision5 vision 5
// Controller1 controller
// frontMotor motor 1
// leftMotor motor 10
// backMotor motor 11
// rightMotor motor 20
// LimitSwitchA limit A
// ArmMotors motor_group 15, 16
// LimitSwitchB limit B
// clawMotor motor 3
// LineTrackerC line C
// LineTrackerD line D
// ---- END VEXCODE CONFIGURED DEVICES ----
```

```
#include "cmath"
#include "vex.h"

using namespace vex;

// Define constants and initial values.

//----------------------------------------------

#define ARMSPEED 100
#define CLAWSPEED 20

int velocity = 100;

bool closed = false;

bool slowMode = false;

float velocityPercent = 1;

bool colorToggle = false;
```

```
//---------------------------------------------


void clawOpen() {
if (closed) {
clawMotor.spinFor(-100, degrees);
closed = false;
}
}
void clawClose() {
if (!closed) {
clawMotor.spinFor(100, degrees);
closed = true;
}
}
void teleOp() {


// Reads controller values and sets motor speeds accordingly. Designed to be run in a loop to allow other actions to occur simultaneously.


if (slowMode) {
velocityPercent = .5;
} else {
velocityPercent = 1;
}
```

```
frontMotor.setVelocity((Controller1.Axis3.position(percent) +
Controller1.Axis1.position(percent)) * velocityPercent,percent);

rightMotor.setVelocity((Controller1.Axis3.position(percent) -
Controller1.Axis1.position(percent)) * velocityPercent,percent);

backMotor.setVelocity((Controller1.Axis4.position(percent) -
Controller1.Axis1.position(percent)) * velocityPercent,percent);

leftMotor.setVelocity((Controller1.Axis4.position(percent) +
Controller1.Axis1.position(percent)) * velocityPercent,percent);

frontMotor.spin(forward);

leftMotor.spin(forward);

backMotor.spin(forward);

rightMotor.spin(forward);



clawMotor.setVelocity(CLAWSPEED * velocityPercent, percent);



if (Controller1.ButtonL1.pressing() && !LimitSwitchB.pressing()) {

ArmMotors.setVelocity(ARMSPEED * -velocityPercent, percent);

ArmMotors.spin(forward);



} else if (Controller1.ButtonL2.pressing() && !LimitSwitchA.pressing()) {

ArmMotors.setVelocity(ARMSPEED * velocityPercent, percent);

ArmMotors.spin(forward);

} else {

ArmMotors.stop();

}



if (Controller1.ButtonLeft.pressing()) {

Brain.Screen.clearScreen();
```

```
Brain.Screen.setCursor(1, 1);
Brain.Screen.print(ArmMotors.position(degrees));
}
if (Controller1.ButtonR1.pressing()) {
clawMotor.spin(reverse);


} else if (Controller1.ButtonR2.pressing()) {
clawMotor.spin(forward);
} else {
clawMotor.stop();
}
}
void armUp() {


// Sets arm to top position, determined with a limit switch, and zeroes encoders.


ArmMotors.spin(reverse);
while (!LimitSwitchB.pressing()) {
}
ArmMotors.stop();
ArmMotors.resetPosition();
}


// Set arm to a specified encoder position. Must be called after armUp() to ensure accurate positioning.
```

```
void setArm(int pos) { ArmMotors.spinTo(pos, degrees); }


bool targetFollow(int pos, vex::vision::code sig) {
// Function that aligns the robot with a vision target defined by the
// signature passed as the second parameter. Uses P control for x position and
// PD for y position, which is based on size of the target. int pos defines
// the ideal position of the vision target in the camera FOV, with 150 being
// centered.
//Designed to be run in a loop so other actions can occur simultaneously.


float KpX = .3;
float KdXSize = .1;
float prevErrorXSize = 0;
float KpXSize = 0.9;
float targetX = pos; // 150 is center
float targetSize = 90;
int stopThresh = 6;


Vision5.takeSnapshot(sig);


if (Vision5.objectCount > 0) {
// Determine the error
float errorX = targetX - Vision5.largestObject.centerX;
float errorXSize = targetSize - Vision5.largestObject.width;
//Set motor speeds
frontMotor.setVelocity(errorXSize * KpXSize - (errorX - prevErrorXSize) * KdXSize, percent);
```

```
leftMotor.setVelocity(-errorX * KpX, percent);
backMotor.setVelocity(-errorX * KpX, percent);
rightMotor.setVelocity(errorXSize * KpXSize - (errorX - prevErrorXSize) * KdXSize, percent);
// Run motors
frontMotor.spin(forward);
leftMotor.spin(forward);
backMotor.spin(forward);
rightMotor.spin(forward);
//Store size error for use in derivative control
prevErrorXSize = errorXSize;
// Check if robot is close enough to ideal position and return true/false to signal this
if (abs(int(-errorX * KpX)) < stopThresh && abs(int(errorXSize * KpXSize - (errorX -
prevErrorXSize) * KdXSize)) < stopThresh) {
return true;
} else {
return false;
}
} else {
return false;
}
}


float inchesToDegrees(float inches) {
// Convert inches of desired travel to motor degrees
const float diameter = 4.25;
const float ratio = 3; // Drivetrain gear ratio
float ans = (inches / (diameter * 3.14)) * 360 * ratio;
return ans;
}
```

```
void driveInchesDeg(float dist, float dir) {

// Drive a specified number of inches in a specified direction, given by a degree measure clockwise from forward = 0 degrees.

frontMotor.resetRotation();

leftMotor.resetRotation();

backMotor.resetRotation();

rightMotor.resetRotation();



// Do math for speed and degrees to turn for each motor.



float horiTrig = sin(6.283 * dir / 360);

float vertTrig = cos(6.283 * dir / 360);



frontMotor.setVelocity(velocity * vertTrig, percent);

rightMotor.setVelocity(velocity * vertTrig, percent);

leftMotor.setVelocity(velocity * horiTrig, percent);

backMotor.setVelocity(velocity * horiTrig, percent);



float deg = inchesToDegrees(dist);

float FR = deg * vertTrig;

float BL = deg * vertTrig;

float FL = deg * horiTrig;

float BR = deg * horiTrig;

// Determine which motion will take longer (x or y) and only wait for that one to finish. Allows for straight line motion.



if (abs(FR) > abs(FL)) {
```

```
rightMotor.spinFor(forward, BL, degrees, false);
leftMotor.spinFor(forward, BR, degrees, false);
backMotor.spinFor(forward, FL, degrees, false);
frontMotor.spinFor(forward, FR, degrees, true);
} else {
frontMotor.spinFor(forward, FR, degrees, false);
rightMotor.spinFor(forward, BL, degrees, false);
leftMotor.spinFor(forward, BR, degrees, false);
backMotor.spinFor(forward, FL, degrees, true);
}
frontMotor.setVelocity(velocity, percent);
leftMotor.setVelocity(velocity, percent);
backMotor.setVelocity(velocity, percent);
rightMotor.setVelocity(velocity, percent);
frontMotor.stop();
leftMotor.stop();
backMotor.stop();
rightMotor.stop();


}
void driveDirection(float dir) {


// Starts robot moving in a direction and allows it to travel until another sensor stops it.
// Used with vision sensor to travel until a target is found.
// Direction is degrees clockwise from forward = 0 degrees.


float horiTrig = sin(6.283 * dir / 360);
```

```
float vertTrig = cos(6.283 * dir / 360);


frontMotor.setVelocity(velocity * vertTrig, percent);
rightMotor.setVelocity(velocity * vertTrig, percent);
leftMotor.setVelocity(velocity * horiTrig, percent);
backMotor.setVelocity(velocity * horiTrig, percent);
frontMotor.spin(forward);
leftMotor.spin(forward);
backMotor.spin(forward);
rightMotor.spin(forward);
}
float turnToDegrees(float deg) {
// Converts pivot angle to motor degrees.
const float wheelBase = 15;
float dist = (wheelBase * 3.14) * (deg / 360) * 3;
return inchesToDegrees(dist);
}
void pivot(float deg) {
// Pivots the robot about its center a given number of degrees clockwise.
// Negative values can be passed for counterclockwise motion.
frontMotor.spinFor(forward, turnToDegrees(deg), degrees, false);
leftMotor.spinFor(forward, turnToDegrees(deg), degrees, false);
backMotor.spinFor(reverse, turnToDegrees(deg), degrees, false);
rightMotor.spinFor(reverse, turnToDegrees(deg), degrees, true);
}
void autoRamp() {
// Autonomous code for climbing the ramp. Uses ultrasonic sensor and P control to follow one
wall up, then detect the end of the ramp.
frontMotor.spin(reverse, 60, percent);
```

```
rightMotor.spin(reverse, 60, percent);
float Kp = 5;
float targetDist = 6;
float distTest = RangeFinder.distance(inches);
int breakFlag = 0;
while (true) {
distTest = RangeFinder.distance(inches);
if (distTest > 30) {
if (breakFlag > 10) {
break;
} else {
breakFlag++;
}
} else {
breakFlag = 0;
}
float error = targetDist - distTest;
leftMotor.spin(forward, error * Kp, percent);
}
frontMotor.stop();
rightMotor.stop();
frontMotor.setVelocity(velocity, percent);
rightMotor.setVelocity(velocity, percent);
}
void fullAuto() {
// Auto for CDR. Climbs ramp and delivers 1 pizza.
closed = true;
// Zero out the arm with the limit switch at the full up position to ensure accurate positioning.
armUp();
```

```
autoRamp();
velocity = 50;
driveInchesDeg(20, 180);
setArm(900);
velocity = 100;
pivot(-45);
driveInchesDeg(20, 135);
driveInchesDeg(20, 180);
velocity = 50;
driveDirection(225);


// Find Faraday for first pizza.


while (true) {
Vision5.takeSnapshot(Vision5__MARKER);
if (Vision5.objectCount != 0) {
break;
}
}
frontMotor.stop();
leftMotor.stop();
backMotor.stop();
rightMotor.stop();
pivot(-45);
driveDirection(90);


// Find pizzeria
```

```
while (true) {
Vision5.takeSnapshot(Vision5__MARKER);
if (Vision5.objectCount != 0) {
break;
}
}
frontMotor.stop();
leftMotor.stop();
backMotor.stop();
rightMotor.stop();
while (!targetFollow(150, Vision5__MARKER)) {
}
armUp();
driveInchesDeg(43 / 2.54, 0);
setArm(700);
clawOpen();
vexDelay(1000);
armUp();
driveInchesDeg(10, 180);
velocity = 100;
}
void fullAuto2() {


// Auto for CDR. Climbs ramp and delivers 2 pizzas: the preloaded one and one collected from the pizzeria.
```

```
armUp();
setArm(1390);
autoRamp();
velocity = 50;
driveInchesDeg(20, 180);
setArm(1000);
velocity = 100;
pivot(78);
setArm(1200);
driveInchesDeg(44, 155);
driveInchesDeg(25, 90);
driveDirection(90);


// Find Faraday for first delivery.


while (true) {
Vision5.takeSnapshot(Vision5__MARKER);
if (Vision5.objectCount != 0) {
break;
}
}
frontMotor.stop();
leftMotor.stop();
backMotor.stop();
rightMotor.stop();
while (!targetFollow(150, Vision5__MARKER)) {
}
armUp();
```

```
setArm(840);
vexDelay(1000);
driveInchesDeg(40 / 2.54, 0);
closed = true;
clawOpen();
driveInchesDeg(2, 90);
driveInchesDeg(10, 180);
pivot(-80);
setArm(1200);


// Find pizzeria.


driveDirection(0);
while (!targetFollow(150, Vision5__MARKER)) {
}
setArm(950);
vexDelay(1000);
driveInchesDeg(10, 0);


// Wait for pizza to be inserted into claw.


while (LineTrackerC.value(percent) > 59) {
}
closed = false;
clawClose();
vexDelay(2000);
```

```
driveInchesDeg(10, 180);
pivot(90);
setArm(1200);
driveInchesDeg(30, 90);


// Find Faraday again.


driveDirection(90);
while (true) {
Vision5.takeSnapshot(Vision5__MARKER);
if (Vision5.objectCount != 0) {
break;
}
}
frontMotor.stop();
leftMotor.stop();
backMotor.stop();
rightMotor.stop();
while (!targetFollow(150, Vision5__MARKER)) {
}
armUp();
setArm(1070);
vexDelay(1000);
driveInchesDeg(37 / 2.54, 0);
closed = true;
clawOpen();
driveInchesDeg(2, 90);
driveInchesDeg(26, 180);
```

```
}
void OEDauto(bool LR) {


// Auto for OED. Delivers preloaded pizza, returns to pizzaria and picks up second pizza, delivers second pizza.
// Pizzas go to 2nd and 3rd floor of Faraday for maximum points.
// Can be set for Blue side or Red side.


armUp();
setArm(1300);
driveInchesDeg(34, 0);
if (LR) {
pivot(-80);
driveDirection(90);
} else {
pivot(80);
driveDirection(-90);
}


// Find Faraday and deliver pizza.


while (true) {
Vision5.takeSnapshot(Vision5__MARKER);
if (Vision5.objectCount != 0) {
break;
}
}
```

```
frontMotor.stop();
leftMotor.stop();
backMotor.stop();
rightMotor.stop();
while (!targetFollow(150, Vision5__MARKER)) {
}
setArm(800);
vexDelay(1000);
driveInchesDeg(35 / 2.54, 0);
closed = true;
clawOpen();
driveInchesDeg(2, 90);
driveInchesDeg(10, 180);
setArm(1400);
if (LR) {
pivot(-75);
} else {
pivot(75);
}
setArm(1200);


// Find pizzeria and obtain second pizza.


driveDirection(0);
while (!targetFollow(170, Vision5__MARKER)) {
}
setArm(940);
vexDelay(1000);
```

```
driveInchesDeg(10, 0);


// Waits for pizza to be inserted in claw.


while (LineTrackerC.value(percent) > 59) {
}
closed = false;
clawClose();
vexDelay(2000);
driveInchesDeg(10, 180);
setArm(1200);
if (LR) {
pivot(70);
driveInchesDeg(30, 90);
driveDirection(90);
} else {
pivot(-70);
driveInchesDeg(30, -90);
driveDirection(-90);
}


// Find Faraday again and deliver second pizza.


while (true) {
Vision5.takeSnapshot(Vision5__MARKER);
if (Vision5.objectCount != 0) {
```

```
break;
}
}
frontMotor.stop();
leftMotor.stop();
backMotor.stop();
rightMotor.stop();
while (!targetFollow(150, Vision5__MARKER)) {
}
setArm(1000);
vexDelay(1000);
driveInchesDeg(37 / 2.54, 0);
closed = true;
clawOpen();
driveInchesDeg(2, 90);
driveInchesDeg(16, 180);
}


// Functions called when "B" button is pressed/released. Sets all motor speeds to half, so that any action can be performed slower to increase precision.


void slowModeOn() { slowMode = true; }
void slowModeOff() { slowMode = false; }


void toggleColor() {


// Toggles color shown on screen and the auto to run for CDR
```

```
if (colorToggle) {
Brain.Screen.clearScreen(red);
colorToggle = false;
} else {
Brain.Screen.clearScreen(blue);
colorToggle = true;
}
}

int main() {
// Initializing Robot Configuration. DO NOT REMOVE!
vexcodeInit();

// Set motor speeds and brake modes

frontMotor.setVelocity(velocity, percent);
leftMotor.setVelocity(velocity, percent);
backMotor.setVelocity(velocity, percent);
rightMotor.setVelocity(velocity, percent);

Controller1.ButtonB.pressed(slowModeOn);
Controller1.ButtonB.released(slowModeOff);

ArmMotors.setStopping(hold);
```

```
ArmMotors.setVelocity(ARMSPEED, percent);

clawMotor.setBrake(hold);

while (true) {

// Check button for color mode.

if (LimitSwitchA.pressing()) {
toggleColor();
while (LimitSwitchA.pressing()) {
vexDelay(100);
}
}

if (Controller1.ButtonA.pressing()) {

// Run auto.

if (colorToggle) {
fullAuto2(); // BLUE
} else {
fullAuto(); // RED
}
} else {
```

```
// Run teleop, which reads values from the controller and sets motors accordingly.


teleOp();
}
}
}
```