RBE1001: Introduction to Robotics Engineering

Professor Lewin

Group 4

Riley Harn, Alex Fenwick, Alex Knox, Emma Maertz

Fruit Harvesting Robot: D.S.[1]

Final Proposal

Saturday, March 2nd, 2024

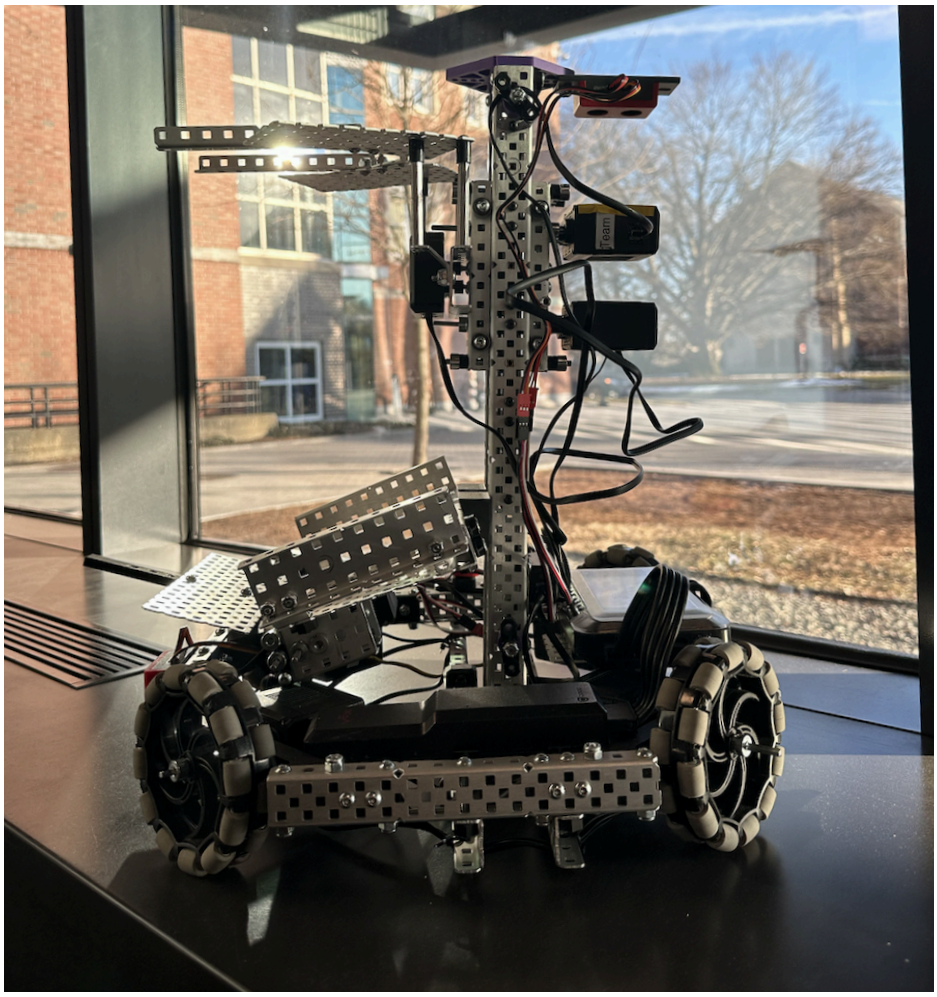


Figure 1. Image of D.S., the robot, as of Wednesday, February 21st.

[1] "D.S." is an acronym for the name of the robot.

Table of Contents

1.0 Introduction

Everything must consume. We live in a world where we are entirely dependent on our ability to acquire things to consume. From plants needing the sun to conduct photosynthesis, to animals needing grass to munch on, to humans needing food every day to survive, our entire world runs on the premise that we can acquire food to eat. The modern world—full of Doordash, where we can shop without lifting a finger and grocery stores having already harvested and cleaned food for us to eat, we can tend to overlook the processes by which we are able to acquire food. As the population increases, food harvesting and processing must also increase.

For this project, the group decided to delve into the world of fruits. Picking fruits from a tree can be a laborious task, however through the practice of engineering, we can create tools that can make picking fruit easier for humans, faster, and more efficient. Given an orchard filled with various fruits, we endeavor to create a robot that can traverse the field, detect specific fruits, collect fruits, and sort them into bins. To demonstrate this capability, we have scaled down this project, using a model orchard fashioned to a table with PVC pipe trees and various 3D printed fruits. We will develop a small, autonomous robot which can detect fruits of a variety of sizes/weights, collect them, and deposit them into a bin designated for that fruit.

1.1 Needs and Objectives

Assume all abilities would be conducted autonomously.

An ideal system consists of the ability to:

- ➢ Drive around the orchard.
- ➢ Traverse across various terrain.
- ➢ Pick/collect any flavor[2], size, or height of tree-hanging fruit.
- ➢ Pick/collect fruit in any row of the orchard.
- ➢ Deposit any flavor of fruit into any orchard deposit box.[3]
- ➢ Deposit any size and flavor of fruit into any deposit box.
- ➢ Deposit any size and flavor of fruit into its flavor-designated deposit box.

An ideal system is mechanically simple, but robust and reliable.

[2] "Flavor" refers to differentiating features of the limes, lemons, and oranges from one another. This can be visually seen in the colors of the fruits.

[3] "Deposit Box" is defined as the box in which fruits are deposited on the orchard.

To demonstrate expected abilities of robot:

- ➢ Drive around the orchard.
- ➢ Pick/collect any size limes, lemons, and oranges.
- ➢ Pick/collect fruit autonomously in predetermined rows of the orchard.
- ➢ Deposit any size of limes, lemons, and oranges into any deposit box or the flavor-designated deposit box.

Our system is mechanically simple, but robust and reliable. We have been constrained to a strict timeline in which manufacturing a surplus of custom parts is not realistic. Additionally, due to the fact that we are using VEX parts and sensors, a highly robust system is not attainable given that VEX parts are generally used for educational purposes and are not as durable or accurate as other sensors or parts.

Target Abilities & Why:

We aim to be able to drive around the orchard because you cannot collect fruit without having a chassis to drive to it. Picking limes, lemons, and oranges is the quintessential objective of this project. The robot must be able to pick fruits, however, due to the lack of grapefruits on the field, the robot does not need to be able to collect those fruits. Fruits come in all shapes and sizes, ergo our robot must be able to pick/collect and deposit fruits of every size variety available on the model orchard. Fruits can be located at varying heights on trees, therefore our robot must be able to pick/collect fruits at a variety of heights. For this project, our robot should be able to collect fruits at the three pre-determined branch heights of the trees. It will attempt to pick most of the fruits it passes in each row. Our robot will follow a prescribed path through the orchard to limit the programming complexity of our system and adhere to the restrictions placed on us regarding the number of sensors that can be used. Picking fruits is a laborious task, just as sorting fruits is. Due to the constraints placed upon this project, we have decided to prioritize picking and depositing fruits rather than sorting them. As important as sorting the fruits into correct bins are, we have decided to limit our scope to only include the collection and depositing of fruits. While this does not mean we will not attempt to sort the fruits into their designated bins, it does mean that we will not prioritize this ability over fruit collection and deposition.

1.2 Performance Goals

Goal/Objective	*Requirement*
Drive around the orchard.	The robot must be able to autonomously navigate the orchard[4] for at least 20 seconds.
Pick/collect limes, lemons, and oranges.	The robot must be able to pick/collect limes with an accuracy of 85% or higher.
Pick/collect limes, lemons, and oranges.	The robot must be able to pick/collect lemons with an accuracy of 85% or higher.
Pick/collect limes, lemons, and oranges.	The robot must be able to pick/collect oranges with an accuracy of 85% or higher.

See Appendix 1. for the entire list of requirements.

[4] "Navigation" – we assume that navigation means the avoidance of obstacles and movement that is done to reach a certain place on the field. To be more specific, we don't consider navigation to include the robot simply driving aimlessly; to be considered navigation, the robot must be traveling with an objective in mind, whether that be to reach a row, tree, fruit, or deposit box.

2.0 System Design

The robot consists of three subsystems: a drivebase, elevator, and box mechanism. Each of these subsystems will be detailed in this section.

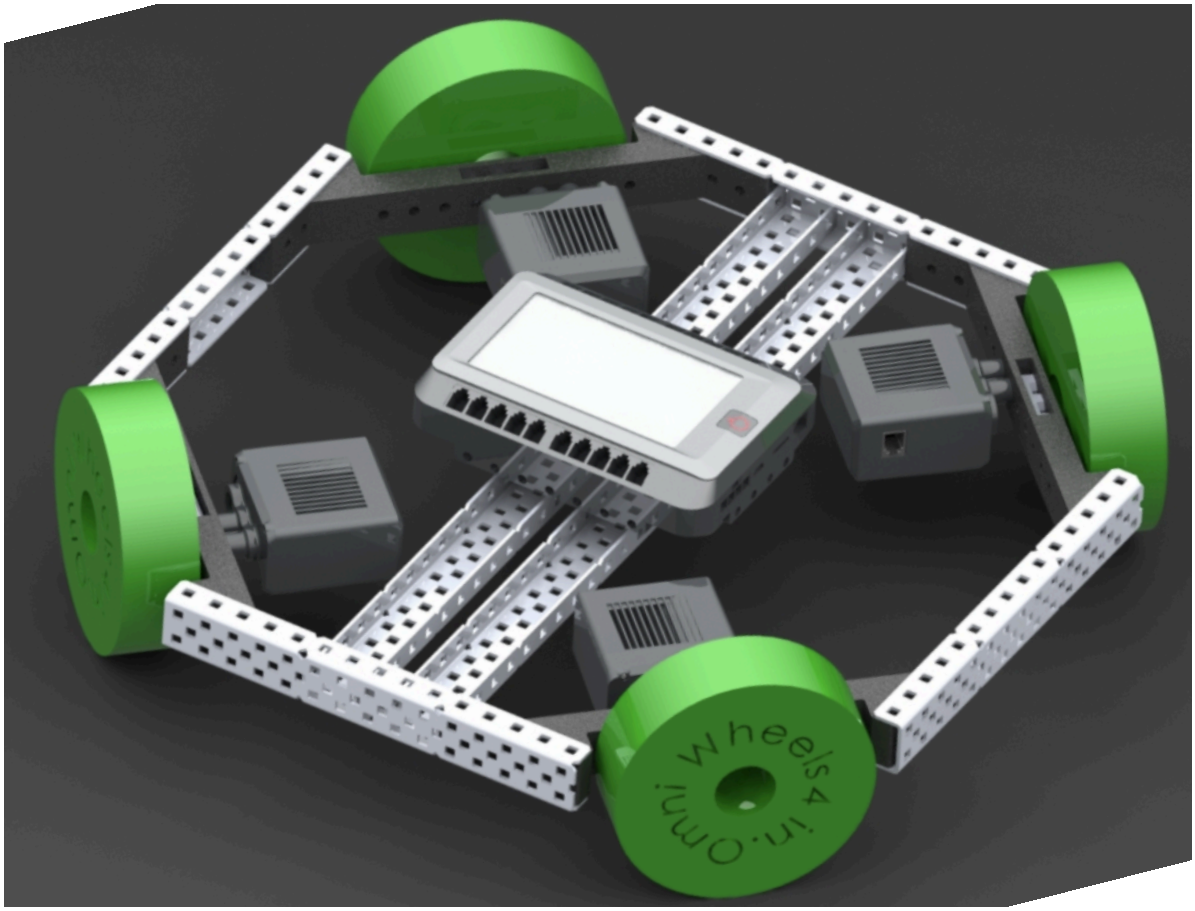


Figure 2. Computer Aided Design rendering of the drivebase.

2.0.1. Drivebase

The robot's drive base can be categorized as x-drive[5]. The robot has a holonomic drive and omni wheels to ensure maximization of maneuverability across the field. With the orchard consisting of rows, the robot must be able to turn around corners and move from row to row with ease.

[5] An "X-drive" drivebase is a holonomic base with four omni wheels positioned in an x.

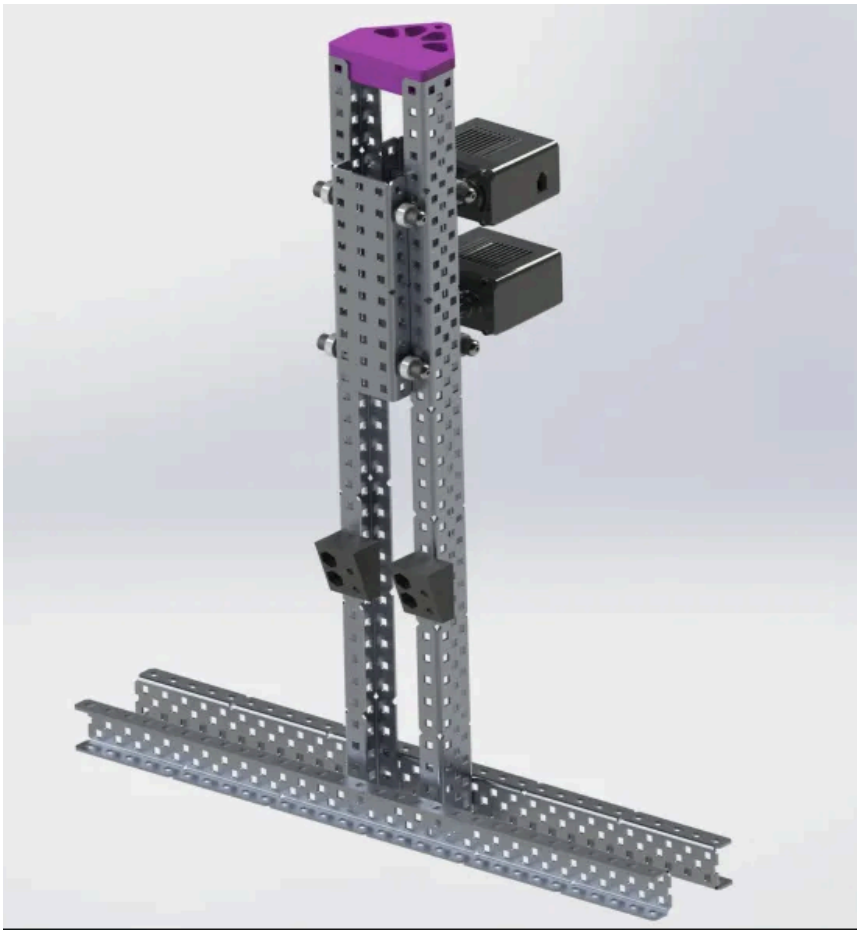


Figure 3. 3D rendering of elevator subsystem of D.S.

2.0.2. Elevator

An elevator is the mechanism that allows the robot to reach fruits at a variety of heights. The robot is able to raise and lower an integrated knocking geometry to collect fruit into the bucket, taking feedback from the vision sensor to make adjustments that way. To pick/collect fruit, the elevator is lifted or lowered and positioned such that the integrated knocking geometry straddles the fruit. The robot moves to knock the fruit and it falls into the bucket below the elevator.

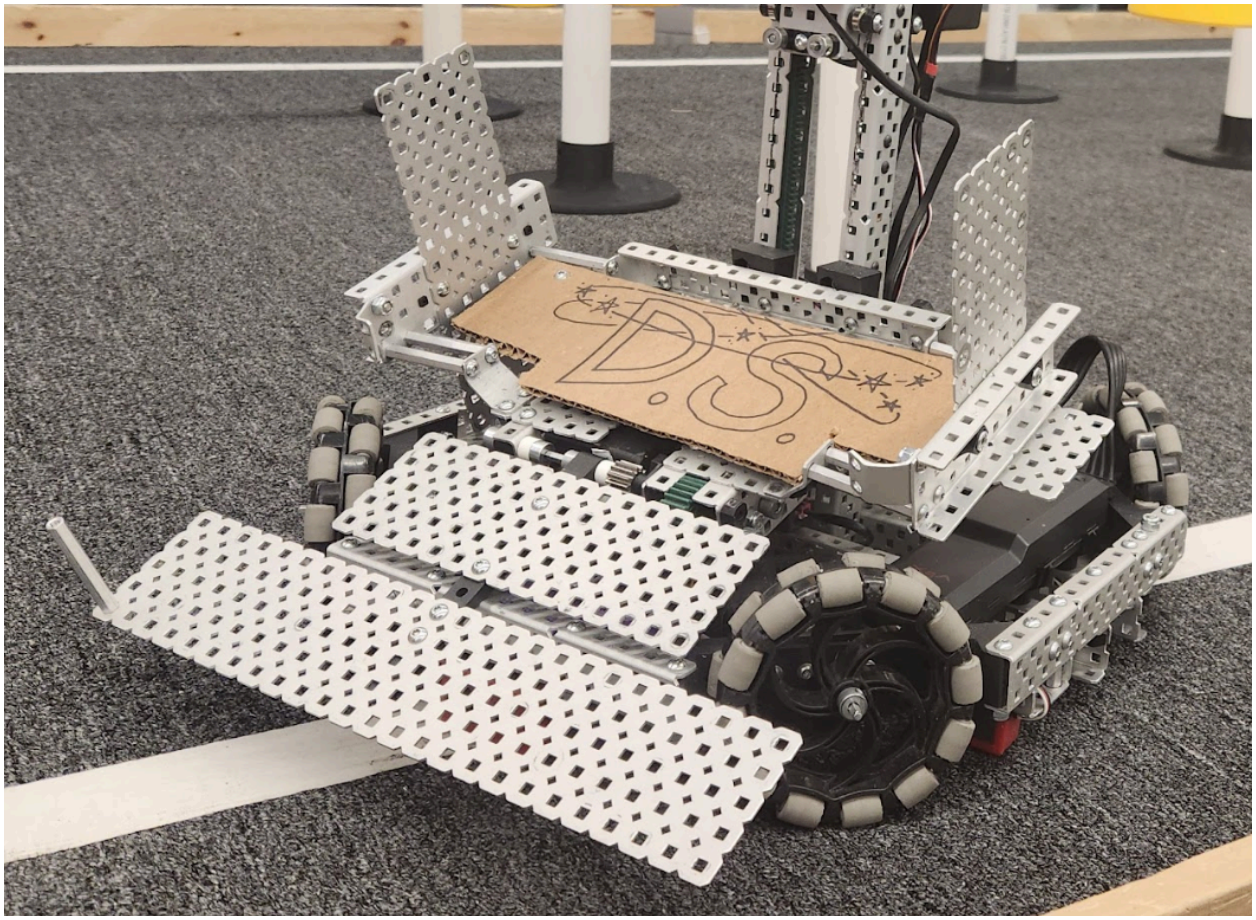


Figure 4. Image of the box mechanism of the robot with the door open.

2.0.3. Box Mechanism

The box mechanism acts as a holder to the fruits prior to depositing. After being knocked from the tree by the elevator, fruits are held within the box mechanism.

2.1 Subsystems

- *Describe the core functionality and related challenges and constraints in greater detail, focusing on this subsystem.*
- *Describe some of the options/possibilities.*
- *Describe experiments or analysis that you used to evaluate key options. Here, you should draw on previous lab exercises, where you tested different methods, or exercises for deciding among options (e.g., the harvesting subsystem). Your goal here is to justify your selection.*
- *Describe the final implementation in detail. Discuss specific design parameters (gear selection; processing methods; etc.).*
- *Describe unit testing that you did (note that you've already described testing in the selection process, so you might just refer to those tests).*

2.1.1 Mobility & Drivetrain

As mentioned in section 2.0.1., the x-drive enables the robot to have enhanced maneuverability without adding significant complexity. Additionally, the x-drive allows for greater control when turning and a tighter turn radius. Other drives were debated, especially the base bot drive, however, x-drive best balanced the desired maneuverability and low level of complexity. With the base bot drive, additional programming work would be needed to satisfy the high level of maneuverability required for the challenge. Based on testing completed in lab 1 (see appendix 2, controlling the base bot driving using the IMU and ultrasonic rangefinder was not as accurate as is necessary for this challenge. The base bot drive was changed in favor of x-drive and the accuracy of maneuverability was much increased.

For the drive motors, direct drive was used. The drive base did not require any gearing so direct drive removed the loss of efficiency due to gears.

2.1.2 Harvesting & Delivery

Initial ideas considered include the rake, roller claw, box collector, elevator with end effector, and big magnet. They are detailed below:

The Rake

The rake would have prongs that can reach up onto the branch and knock fruit off into some sort of collector underneath. One pro of this design is that it is mechanically simple; it would not take many designs to determine an effective geometry. It would also be reliable, as you can use the entire length of the branch to guide you to the fruit. However, this design also has a fairly narrow window for targeting the fruits.

Roller Claw

The roller claw is a claw with powered grippy wheels. The wheels on the ‘claw’ would spin and be squishy so that it could get both circumferences of fruit. Pros of this collector are: it would be fast at collecting individual fruits, it’s a tried and true method of collecting in FRC, which we’ve all used at some point, and would be a good candidate if we wanted to adopt a ‘touch it own it’ philosophy, where once we’ve made contact with the fruit, we have it. Cons are that this design is more complicated than other

ideas, is not necessarily as effective at a VEX scale as it is in FRC, and may need more than VEX or 3D printed materials.

Box Collector

The box collector is a collector that extends upwards towards the tree. It collects by surrounding the fruit while the robot is driving forward, so one of the walls of it knocks the fruit off the tree. The pros of this design are that it is very simple, requiring rudimentary building materials and construction of which we already have access to. Additionally, the wide range of the intake area would allow for more accuracy in collection. However, there are a couple cons. Lifting up the box requires adjustment to pick from different heights. The fruit could also swing instead of detaching from the branch and continue hanging if it is attached effectively enough.

Elevator with End Effector

An elevator or slides with an end effect would allow the robot to reach up to the desired height of fruit with variations in where they are located. There are two methods to accomplishing the strategy of using slides, either part directly beneath or at an angle to guide. The robot that uses non-angled (perpendicular to the floor) slides would use a collector aimed to drop it directly below into the robot's frame where the chassis is held. The angle would work to better intake the fruit aside from just disconnecting it. The main advantage to both is the speed at which slides would be able to operate especially

at a distance. The downside would potentially be the accuracy and feasibility of the design since we would need to design a lot of custom parts.

Big Magnet

The big magnet was one of our less serious ideas, but it utilizes a strong magnet to steal fruit from the tree. We could either put weaker ones on the end of whichever vertical system we use or we could use a comically large magnet to pull everything in with minimal lift systems. The pros would be very mechanically simple given our robot does not have its electronics fried or the structure collapsed in on itself. It could also have a big range, but would cause issues from such a large range.

Choosing the Box

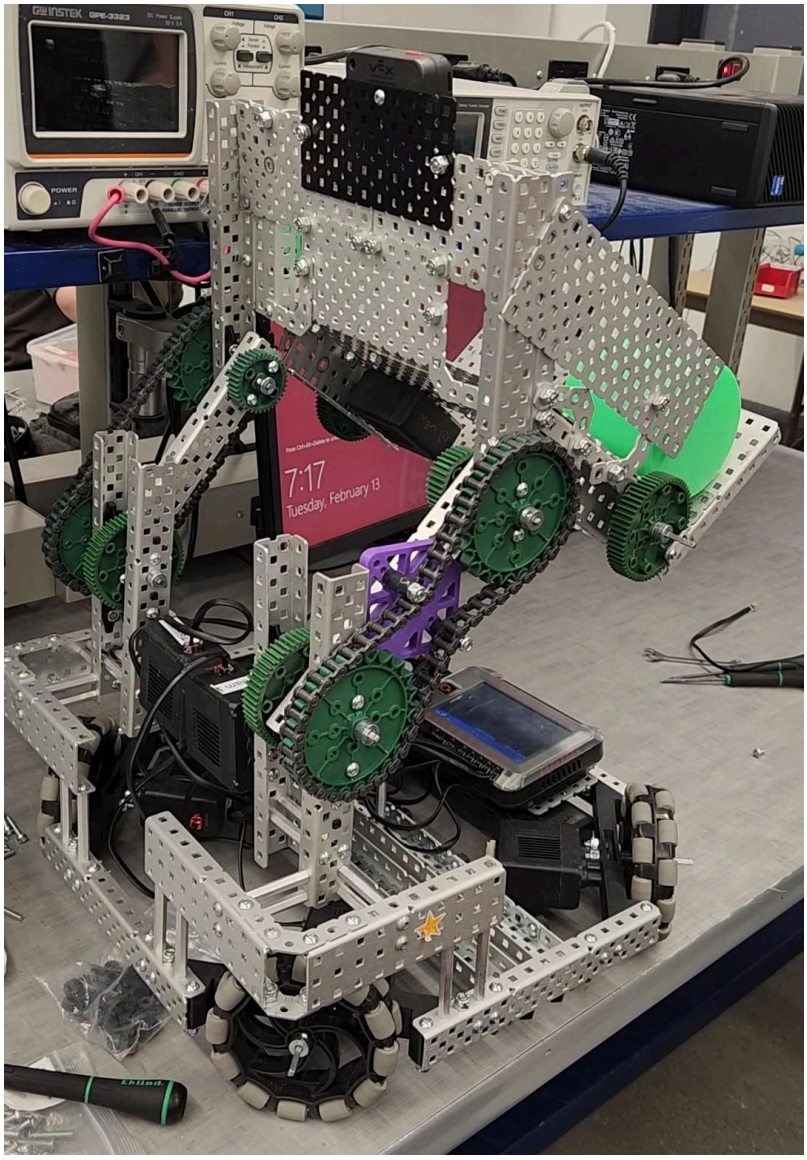


Figure 5: The initial box mechanism design

After determining criteria (see Appendix 3), a collection mechanism was chosen through discourse about how each collection mechanism best met the desired criteria (see Appendix 4). The box collector mechanism was chosen due to its simplicity and the fact that it was first in five out of the seven categories. Overall it excelled in every category. However, the categories that it did not rank first in it ranked second in. Other

ideas, such as The Bigger Magnet, had the potential to cause separate issues for other aspects of the robot, such as creating interference with our electrical systems. After testing, the Box's initial design (See Figure 5) was determined to not be suitable for autonomous control, and was changed to be stationary instead of the initial idea of mounting it on a virtual 4-bar, and a secondary collection mechanism was selected.

With the box subsystem, it is essential that fruits are able to be collected and then stored within the box. Being constrained to the robot perimeter and space taken up by the elevator, the box needed to maximize its size to fit as many fruits as possible. Additionally, the box mechanism was designed such that fruits could be released into their designated deposit boxes. Due to the geometry of the box mechanism, sometimes fruits would get stuck inside the box mechanism. Cardboard was added to reduce this problem, but an additional remedy, adding a "jiggle" function, that would cause the robot to jut back and forth quickly, dislodging the fruit from the box mechanism, was also needed. Additional work could be done to the geometry of the box mechanism so that there would be no need to jiggle fruits out.

Depositing fruit with the box mechanism was tested in lab 5, and it was determined that the box worked well, only needing minor improvements to its construction so that fruits couldn't get caught on the heads of screws or bolts (see Appendix 5). Since the Box's initial redesign, it was widened to fit the entire width of the robot and a gear ratio was added to increase consistency and precision. (See below)

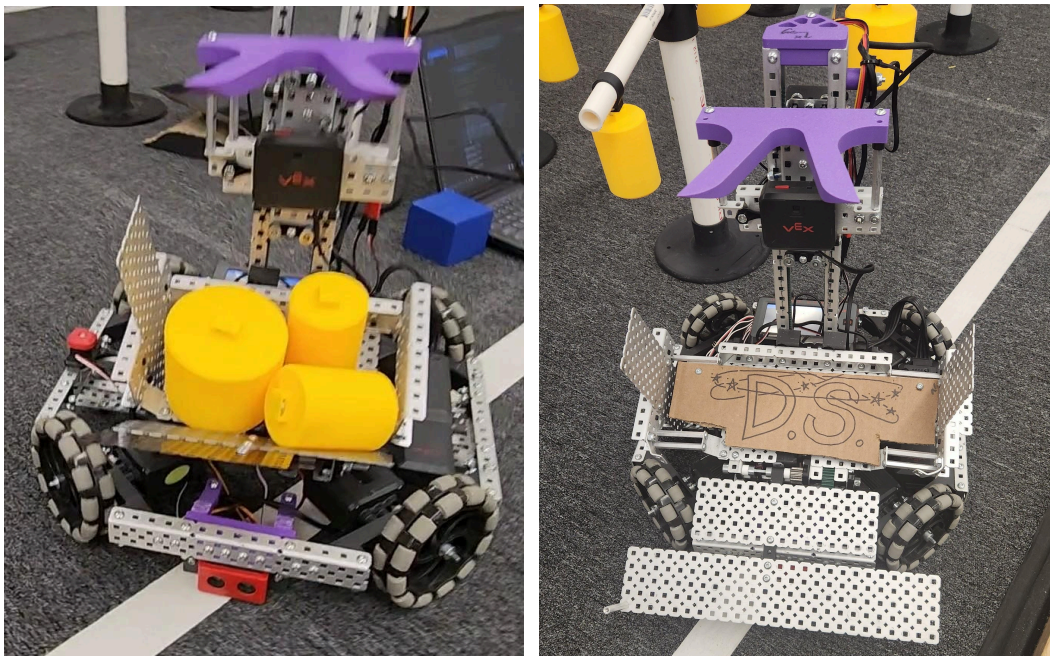


Figure 6a: Left, initial box redesign with narrower, stationary box.

Figure 6b: Right, final version with cardboard to prevent magnetism issues on the bottom and fruits getting stuck behind screw heads.

A 1:3 gear ratio was used on the opening of the box mechanism to allow for greater torque and it was controlled using a motor and two simple functions, openBox() and closeBox(), which rotated the motor to designated set points based on a reference 0 set during box calibration. During Box calibration, the box door was closed until the motor detected a spike in amperage, meaning it was trying to close further than the hard stop. Then the reference point for rotation was set to 0 at that point and the box could be rotated to either 0 (closed) or 270 (open).

The only testing done to the box mechanism was the testing done in lab 5, mentioned above (see Appendix 5).

Reaching the Fruits

Originally, a virtual four bar was developed to enable the collection of fruits at all heights. However, due to some mechanical issues and insufficient torque generated (see Appendix 7), the virtual four bar was removed from the design. An elevator was developed using two motors and some gears. The carriage has two driving gears and 4 stabilizing gears, The driving gears are connected to the motor and are what actually move the carriage up and down. The passive gears on the top and bottom help prevent the carriage from rotating and skipping on the racks. The elevator also uses an ultrasonic sensor to keep track of where the carriage is at all times so it never drives itself into the hardstops and damages the motors.

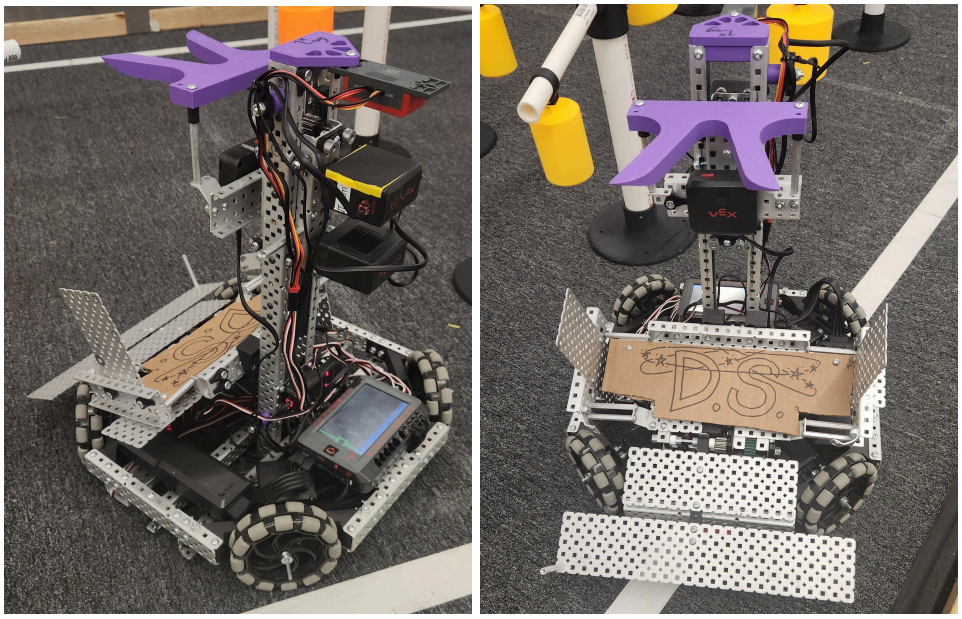


Figure 7a and b: Back and front view of elevator collector mechanism.

Harvesting & Delivery Overview

The robot is able to raise and lower an integrated knocking geometry to collect fruit into the bucket. To pick/collect fruit, the elevator is lifted or lowered and positioned such that the integrated knocking geometry straddles the fruit. The robot moves to knock the fruit and it falls into the bucket below the elevator. After being knocked from the tree by the elevator, fruits are held within the box mechanism. Upon approaching the deposit box, the box mechanism is opened, allowing the fruits to fall into the box.

2.1.3 Navigation & Sensing

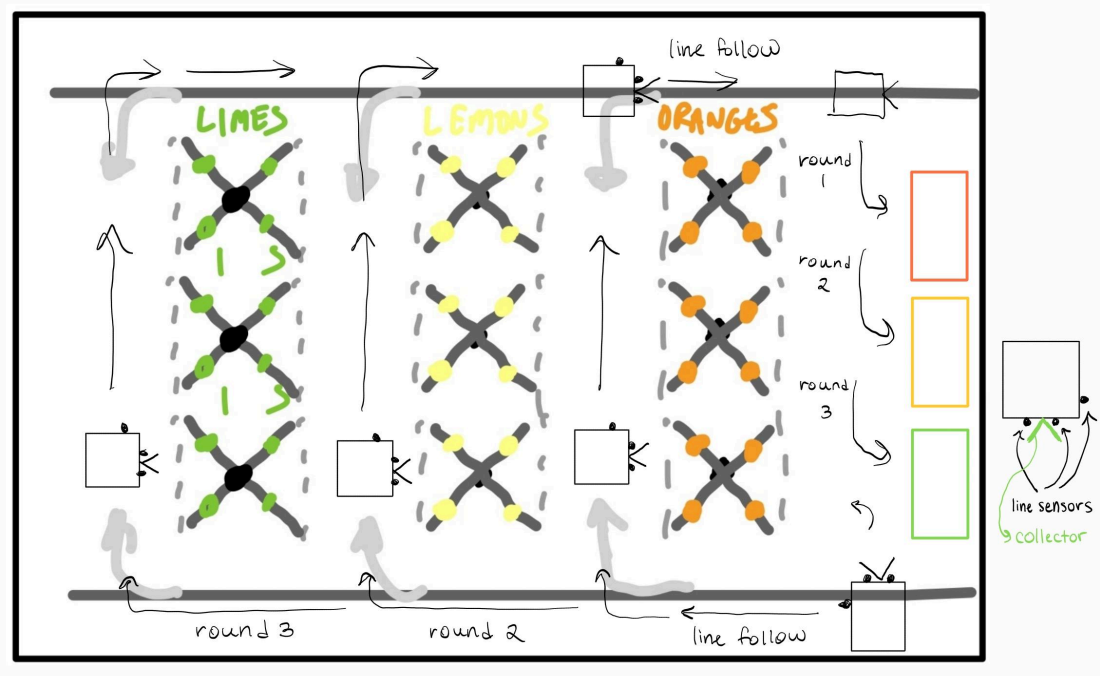


Figure 8: Collection mapping diagram of orchard. The 'V' shape on robot represents the collector, and the dots represent the line sensors.

Line Sensors

The robot features seven sensors that aid both autonomous and teleoperated control. Three line sensors are mounted on the bottom of the robot. Two are mounted about an inch and a half apart centered around the front center of the robot. The third is mounted off-center on the left side of the robot, closer to the front. The two front line sensors are used for forward facing line following such as when driving down the line to go deposit fruit. The side-mounted line sensor is used for sideways line following. Both methods of line following use proportional control to make adjustments to stay on the line that were proportional to how far the robot was off of the line. The two-line sensor

line following was tested and refined in Lab 5 (See Appendix 5), where it was very successful. No issues in testing during Lab 5 were attributed to line following. See Code Excerpt #1 in Appendix 8.

Ultrasonic Sensors

The robot also has two ultrasonic sensors. The first is mounted on the front of the robot between the two front line sensors. It is used for detecting the number of rows the robot passed passed, the wall when approaching the deposit box row, and to correctly distance the robot from the deposit box before depositing fruit (see Code Exerpt #2 in Appendix 8). The second ultrasonic is mounted at the top of the elevator, facing down toward the elevator motors. It measures the distance of the elevator motors and is used to keep the elevator from moving too far up or down and overdriving (see Code Exerpt #3 in Appendix 8).

Inertial Measurement Unit (IMU)

There is also an IMU mounted in the center of the robot. The IMU is used to do more precise proportional control turning and enables field-centric driving with X-drive. See Code Excerpt #4 in Appendix 8.

Camera

Last, a camera is mounted on the elevator just below the pronged collector. The camera was originally planned to be used to identify deposit boxes as well as fruits, but ended

up not being able to see the boxes when the box door was closed. As a result, the camera's primary function is identifying fruits and is used to hone in on fruits by adjusting the drive base and elevator until a fruit is in range for collecting. See Code Excerpts #5 and #6 in Appendix 8 for code using the camera to adjust the elevator and drivebase.

3.0 Integration & Performance

The robot starts in a predetermined starting position such that it can easily navigate down the line to the first row of the orchard. It line follows (see Appendix 8, Code Exerpt 1) until it detects it is at the first row of the orchard using an ultrasonic sensor (Appendix 8, Code Exerpt 2). It drives down one row, only collecting one flavor of fruit, using the camera to identify and hone in on fruits (Appendix 8, Code Exerpts 5 & 6). Then it makes its way to the deposit box row. Upon approaching the deposit box row, it dead reckons to the box that matches the flavor of fruit collected and deposit the fruits. This is repeated for all three flavors of fruit. In order to keep track of which fruit is currently being collected and which is next, the vision signatures were placed in a list and a variable, called currentTargetFruit, was created. The value of currentTargetFruit corresponded with the index of the current signature, starting at zero and going up until 2. Every time the robot completed a collect-deposit cycle, it added one to currentTargetFruit, moving to the next item in the list of fruit signatures. When the value of currentTargetFruit reached 2, the robot stopped and returned to the idle state. (See Appendix 8, Code Exerpt 7)

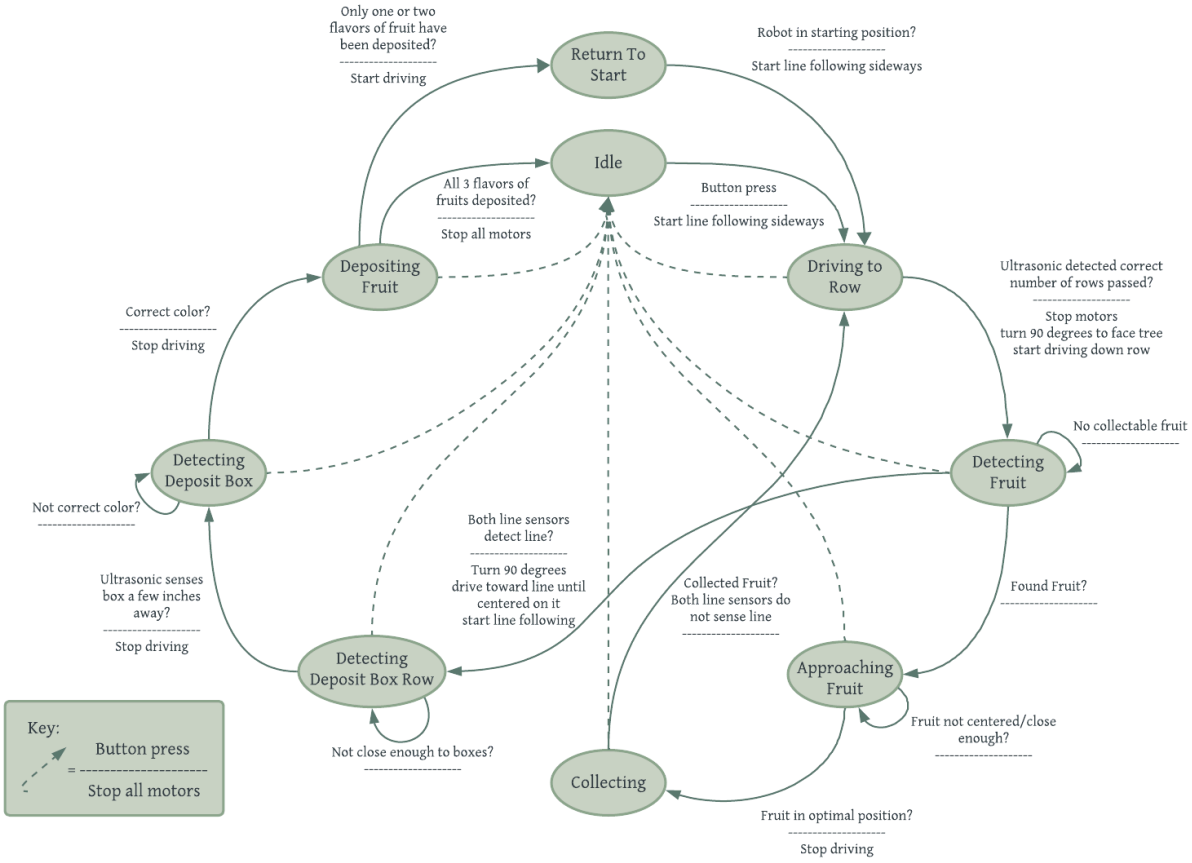


Figure 9: The complete state diagram for the robot's autonomous code

3.1 System Tests

Formal systems testing was not conducted due to time constraints. This is not to say that testing did not occur prior to the demo, however it was highly informal and improvements were made without substantial data analysis. However, some data was collected post-demo to help demonstrate the accuracy of the system. This data was collected prior to the final code-change, so it is only expected that this data would improve. See Appendix 6 to view the table of data collected.

4.0 Demonstration

The robot performed well. The client was in awe! Based on the goals we set with the client, almost all of the performance goals were met. There was some recurring difficulty relating to the ultrasonic sensor during the demo where it saw nonexistent objects, causing the robot to initially be unable to find the first fruit, however after some troubleshooting the problem was temporarily solved. The robot performed fairly well autonomously, with some issues that were previously known but were unable to be solved prior to the demo, such as occasionally becoming 'frozen' after locking onto a fruit or driving too far forward when collecting fruit. However, teleoperation worked well and was able to demonstrate the full range of ability of the robot that autonomous was not able to, such as collecting low-hanging fruit. Overall, through either teleoperation or autonomous control, the robot was able to meet objectives. Additionally, after the client demo, some minor changes were made to the code and the robot worked according to the performance goals more consistently prior to its demise. Please see section 5.0 for footage of the robot collecting and depositing fruits.

Due to the time constraints, we were unable to fully refine our autonomous code for the robot. Given more time, changes we would make include:

- Further tuning the Approaching and Collecting fruits states
- Add integral and derivative control to autonomous arm control and driving.

- Implement odometry so the robot can move more precisely down rows and when collecting.
- Remove any extraneous functions or code for easier readability
- Removing any nesting while loops in the code, which were initially used to save programming time, but in practice cause certain functions such as the ‘start/stop’ button to not work when one of these nested while loops is running.
    - This will most effectively be done by adding more transition states
- Update all state change ‘true/false’ checkers to event checkers by having them check previous values.
- Add another camera and implement vision to identify the deposit boxes by color instead of dead reckoning to them.

5.0 Cinematic Reveal

https://youtu.be/Ik-B6aeegKc?si=NUakhjSCVlF5JtgD

Enjoy!

Appendices

Appendix 1. Table of requirements for performance goals.

Goal/Objective	*Requirement*
Drive around the orchard.	The robot must be able to autonomously navigate the orchard[6] for at least 20 seconds.
Pick/collect limes, lemons, and oranges.	The robot must be able to pick/collect limes with an accuracy of 85% or higher.
Pick/collect limes, lemons, and oranges.	The robot must be able to pick/collect lemons with an accuracy of 85% or higher.
Pick/collect limes, lemons, and oranges.	The robot must be able to pick/collect oranges with an accuracy of 85% or higher.
Pick/collect any size of tree-hanging fruit; Deposit any	The robot must be able to pick/collect the small fruits with

[6] "Navigation" – we assume that navigation means the avoidance of obstacles and movement that is done to reach a certain place on the field. To be more specific, we don't consider navigation to include the robot simply driving aimlessly; to be considered navigation the robot must be traveling with an objective in mind, whether that be to reach a row, tree, fruit, or deposit box.

size of fruit into any deposit box.	an accuracy of 85% or higher.
Pick/collect any size of tree-hanging fruit; Deposit any size of fruit into any deposit box.	The robot must be able to pick/collect the medium fruits with an accuracy of 85% or higher.
Pick/collect any size of tree-hanging fruit; Deposit any size of fruit into any deposit box.	The robot must be able to pick/collect the large fruits with an accuracy of 85% or higher.
Pick/collect any size of tree-hanging fruit; Deposit any size of fruit into any deposit box.	The robot must be able to deposit the small fruits with an accuracy of 85% or higher.
Pick/collect any size of tree-hanging fruit; Deposit any size of fruit into any deposit box.	The robot must be able to deposit the medium fruits with an accuracy of 85% or higher.
Pick/collect any size of tree-hanging fruit; Deposit any size of fruit into any deposit box.	The robot must be able to deposit the large fruits with an accuracy of 85% or higher.
Pick/collect fruit at any height.	The robot must be able to pick/collect fruit from the low branch of the tree with an

	accuracy of 85% or higher.
Pick/collect fruit at any height.	The robot must be able to pick/collect fruit from the middle branch of the tree with an accuracy of 85% or higher.
Pick/collect fruit at any height.	The robot must be able to pick/collect fruit from the high branch of the tree with an accuracy of 85% or higher.
Pick/collect fruit autonomously in predetermined rows of the orchard.	The robot must be able to pick/collect fruit from the first predetermined row.
Pick/collect fruit autonomously in predetermined rows of the orchard.	The robot must be able to pick/collect fruit from the second predetermined row.
Pick/collect fruit autonomously in predetermined rows of the orchard.	The robot must be able to pick/collect fruit from the third predetermined row.
Deposit limes, lemons, and	The robot must be able to deposit

oranges into any deposit box; Deposit limes, lemons, and oranges into its flavor-designated deposit box; Deposit any size of fruit into its flavor-designated deposit box.	limes into any deposit box with an accuracy of 85% or higher.
Deposit limes, lemons, and oranges into any deposit box; Deposit limes, lemons, and oranges into its flavor-designated deposit box; Deposit any size of fruit into its flavor-designated deposit box.	The robot must be able to deposit lemons into any deposit box with an accuracy of 85% or higher.
Deposit limes, lemons, and oranges into any deposit box; Deposit limes, lemons, and oranges into its flavor-designated deposit box; Deposit any size of fruit into its flavor-designated deposit box.	The robot must be able to deposit oranges into any deposit box with an accuracy of 85% or higher.

Deposit limes, lemons, and oranges into any deposit box; Deposit limes, lemons, and oranges into its flavor-designated deposit box; Deposit any size of fruit into its flavor-designated deposit box.	The robot must be able to deposit limes into its flavor-designated deposit box with an accuracy of 50% or higher.
Deposit limes, lemons, and oranges into any deposit box; Deposit limes, lemons, and oranges into its flavor-designated deposit box; Deposit any size of fruit into its flavor-designated deposit box.	The robot must be able to deposit lemons into its flavor-designated deposit box with an accuracy of 50% or higher.
Deposit limes, lemons, and oranges into any deposit box; Deposit limes, lemons, and oranges into its flavor-designated deposit box; Deposit any size of fruit into its flavor-designated	The robot must be able to deposit oranges into its flavor-designated deposit box with an accuracy of 50% or higher.

deposit box.	
Deposit limes, lemons, and oranges into any deposit box; Deposit limes, lemons, and oranges into its flavor-designated deposit box; Deposit any size of fruit into its flavor-designated deposit box.	The robot must be able to deposit small fruits into any deposit box with an accuracy of 90% or higher.
Deposit limes, lemons, and oranges into any deposit box; Deposit limes, lemons, and oranges into its flavor-designated deposit box; Deposit any size of fruit into its flavor-designated deposit box.	The robot must be able to deposit medium fruits into any deposit box with an accuracy of 90% or higher.
Deposit limes, lemons, and oranges into any deposit box; Deposit limes, lemons, and oranges into its flavor-designated deposit box; Deposit any size of	The robot must be able to deposit large fruits into any deposit box with an accuracy of 90% or higher.

fruit into its flavor-designated deposit box.	
Deposit limes, lemons, and oranges into any deposit box; Deposit limes, lemons, and oranges into its flavor-designated deposit box; Deposit any size of fruit into its flavor-designated deposit box.	The robot must be able to deposit small fruits into its flavor-designated deposit box with an accuracy of 50% or higher.
Deposit limes, lemons, and oranges into any deposit box; Deposit limes, lemons, and oranges into its flavor-designated deposit box; Deposit any size of fruit into its flavor-designated deposit box.	The robot must be able to deposit medium fruits into its flavor-designated deposit box with an accuracy of 50% or higher.
Deposit limes, lemons, and oranges into any deposit box; Deposit limes, lemons, and oranges into its flavor-designated	The robot must be able to deposit large fruits into its flavor-designated deposit box with an accuracy of 50% or higher.

deposit box; Deposit any size of fruit into its flavor-designated deposit box.	

Appendix 2. Data Collected in Lab 1 Relating to Performance of Base Bot Navigation using the Ultrasonic Rangefinder and IMU

Trial	RPM	INCHES			DEGREES
	Speed	Start Distance	End Distance	Delta (in)	End Angle
1	125	0	9.25	9.25	3
2		0	9.25	9.25	6
3		0	9	9	4
4		0	9.125	9.125	5
5		0	9.125	9.125	3
6	150	0	8.75	8.75	6
7		0	8.75	8.75	2
8		0	8.75	8.75	1
9		0	8.625	8.625	4
10		0	8.625	8.625	3
11	175	0	8.25	8.25	2
12		0	8.25	8.25	3
13		0	8.25	8.25	0
14		0	8.325	8.325	4
15		0	8	8	3
16	200	0	7.625	7.625	3
17		0	7.875	7.875	2
18		0	7.625	7.625	2
19		0	7.625	7.625	3
20		0	7.5	7.5	3

Table of 20 trials of the robot driving at 4 different speeds, navigating the orchard autonomously.

Appendix 3. Criteria

Criteria	Metrics	Desired Performance	NOTE:
Accuracy of collection	Success rate of acquiring fruit	>= 90%	
	number different types of fruit we can collect	can collect 3 types of fruit	
Speed of collection	Time to collect one fruit	Faster is better	
	Time to collect multiple fruit	Faster is better	
	Time to extend to fruit	Faster is better	
Collector Holding Strength	number of times "detached" fruit is lost during collection	Less is better	
	number of times fruit is dropped while moving/turning/collisions	less is better	
Durability	number of repairs needed per 30 minutes of testing	Less is better	
	number of degrees gears can wiggle	<= 5 degrees, no shifting for stationary items	Not definite depending upon materials given
	number of parts needed to be replaced per 5 trials	0 parts	Replaced within initial testing
Simplicity	number of 3D printed parts	Less is better	
	number of parts	Less is better	
	lines of code	Less is better	
Speed of Field Navigation	time to do only one lap around field	<50 seconds	

	time to go up and down all isles of trees (only driving)	<2 min	
	time to stop in front of tree (DRIVING -> IDLE)	<1 second	
Accuracy of Field Navigation	percent of trials when the robot finishes match within an inch of the prescribed finish position	>90% of the time	
	percent of trials where the robot drives with a >1 inch variation from the path	>90% of the time	
	number of degrees off from desired position at all times	<10 degrees	
Stable	location of center of gravity	lower is better	
	weight of intake/manipulator	<25% of body weight (includes brain/drive motors)	

Appendix 4. Design Selection of Collection Mechanisms

IDEA	Criteria							
	Accuracy of collection	**Speed of collection**	**Collector Holding Strength**	**Durability**	**Simplicity**	**Speed of Field Navigation**	**Accuracy of Field Navigation**	**Stable**
		(assuming lined up already)				(how long it takes to line up to fruit)		
The Rake	2	1	2	2	3	1	n/a	1
The Roller	2	2	1	4	4	2	n/a	3
The Box	1	1	2	1	1	1	n/a	2
The Bigger Magnet	4	3	4	1	1	3	n/a	3
	AUTHORS NOTE: WE ARE ONLY USING NUMBERS TO RANK AND AID IN AUTOMATED COLOR CODING. THANK YOU.							

Appendix 5. Data Collected in Lab 5 when depositing fruits into the deposit boxes

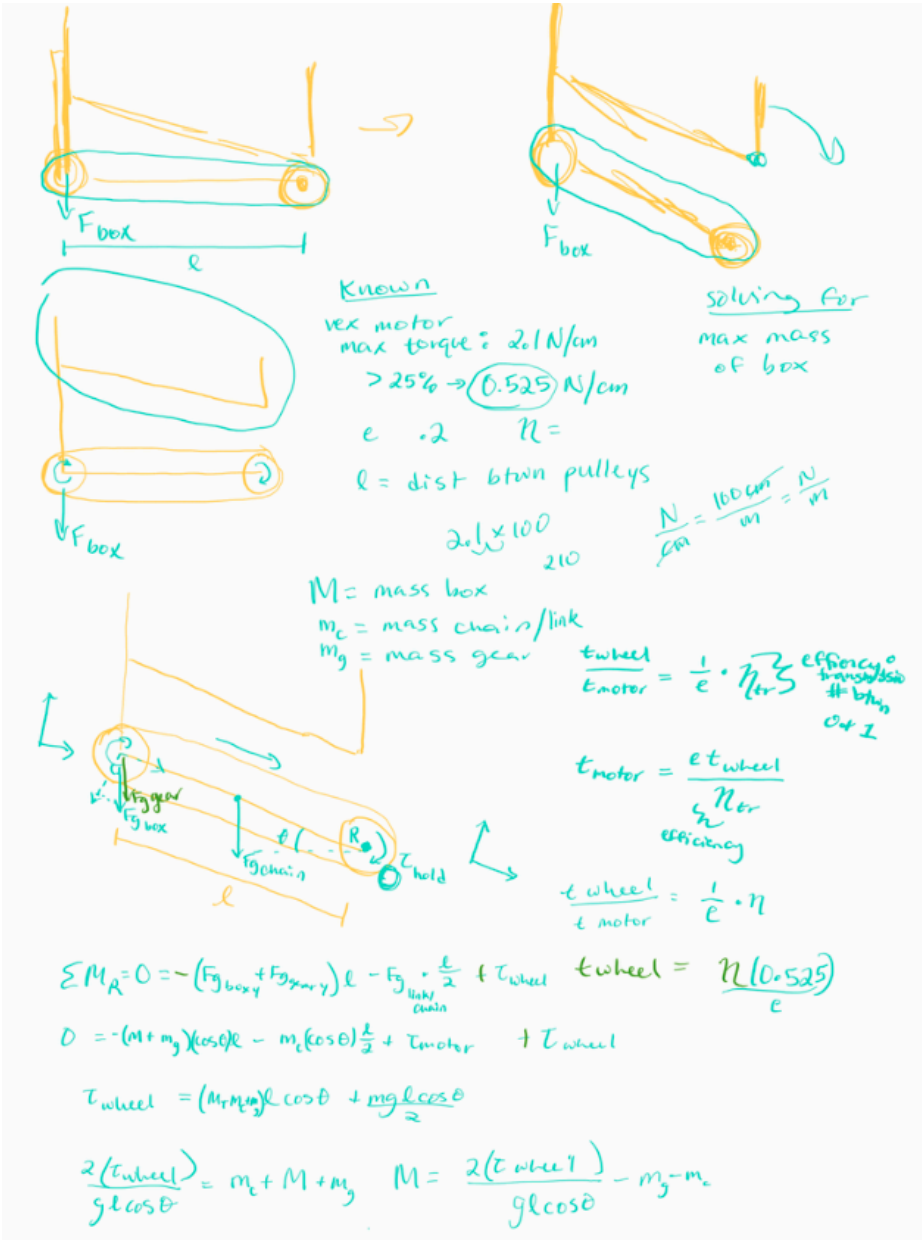
Starting position	Fruit Orientation	# Fruits in bin	Sizes	# Fruits successfully deposited	Accuracy (percent)	Notes (see key for what numbers mean)
End of lemon row	upright	1	big	1	100.0%	
				1	100.0%	
		2	big big	0	0.0%	1st attempt, did not stop @ line, 2nd time, were wedged in too tight together
				2	100.0%	
		3	big small small	2	66.7%	2
				3	100.0%	
		4	all small	4	100.0%	
				0	0.0%	3
		5	all small	5	100.0%	2
				3	60.0%	box did not open all the way; 2
End of lemon row	side	1	big	1	100.0%	
				1	100.0%	
		2	big big	2	100.0%	
				2	100.0%	
		3	big small small	3	100.0%	
				3	100.0%	
		4	all small	3	75.0%	1
				4	100.0%	
		5	all small	5	100.0%	
				5	100.0%	
end of lime row	upright	1	big	1	100.0%	
				1	100.0%	
		2	big big	0	0.0%	
				0	0.0%	4
		3	big small small	2	66.7%	2
				2	66.7%	2.1
		4	all small	2	50.0%	4
				4	100.0%	
		5	all small	4	80.0%	2
				5	100.0%	
end of	side	1	big	1	100.0%	

lime row				1	100.0%	
		2	big big	2	100.0%	
				0	0.0%	3
		3	big small small	3	100.0%	
				3	100.0%	
		4	all small	4	100.0%	
				3	75.0%	1
		5	all small	5	100.0%	
				4	80.0%	
End of orange row	upright	1	big	1	100.0%	4
				1	100.0%	4
		2	big big	2	100.0%	here, updated code to back up a bit so 4 didn't happen
				2	100.0%	
		3	big small small	0	0.0%	3
				0	0.0%	bumped tree and got too angled, then was too far back and all fruit fell out but missed box
		4	all small	3	75.0%	2
				4	100.0%	
		5	all small	5	100.0%	
				4	80.0%	2
End of orange row	side	1	big	1	100.0%	
				1	100.0%	
		2	big big	0	0.0%	moved into depositing fruit state too soon (ultrasonic saw something?)
				2	100.0%	
		3	big small small	3	100.0%	
				3	100.0%	
		4	all small	4	100.0%	
				4	100.0%	
		5	all small	5	100.0%	
				5	100.0%	
				Average:	81.3%	
				Average Upright:	71.5%	
				Average Side:	91.0%	

Appendix 6. Data Collected for Post-Demo Systems Test

	Requirement		**Requirement**	**Requirement**
Trial #	*The robot must be able to autonomously navigate the orchard for at least 20 seconds.*		*The robot must be able to pick/collect at least 1 fruit with an accuracy of 85% or higher.*	The robot must be able to deposit large fruits into any deposit box with an accuracy of 90% or higher.
	Pass	**Fail**	**# of Fruits Picked**	**# Fruits Deposited in the Correct Box**
1	x		0	0
2	x		2	2
3	x		2	2
4	x		2	2
5	x		3	3
6	x		1	1
7	x		2	1
8	x		3	2
9	x		2	2

Appendix 7. Hand-written derivation of the static physics for each variable we could change.



Appendix 8. Code Exerpts

Code Excerpt # 1: lineFollow() and lineFollowSideways() functions used to navigate to and from rows.

```
def lineFollow(speed): # tested, calibrated, working, could still use tuning, but okay for now
    """
    follows the line using front two line sensors. does not adjust heading at all

    :speed: rate at which robot traverses line. positive speed is forward, negative is backward
    """
    strafeKp = 0.004
    # if positive, needs to go left. if negative, needs to go right
    error = lineSensorLeft.value() - lineSensorRight.value()
    strafeEffort = error * strafeKp
    drive([speed, strafeEffort], 0)


def lineFollowSideways(speed): # tested, calibrated, working
    """
    follows along the back side of the line (when robot is backed up to the wall)

    :speed: rate at which robot traverses line. positive speed is left, negative is right
    """
    strafeKp = 0.004
    # if positive, needs to go right. if negative, needs to go left
    error = lineSensorRightAlternate.value() - 1900
    # print(error)
    strafeEffort = error * strafeKp
    drive([strafeEffort, -speed], 0)
```

Code Excerpt #2: two functions, wallDistanceMet() and rowDetected(), used during state changes that use the ‘driving’ or low front-mounted ultrasonic.

```
def wallDistanceMet(dist): # tested, calibrated, working
```

```
    '''
    returns true if the inputted distance from the wall is met (not very
accurate due to ultrasonic difficulties)

    :dist: the desired distance from the wall, in inches
    '''
    distanceToWall = ultrasonicDriving.distance(INCHES)
    if (distanceToWall < dist):
        print("wall detected")
        return True
    else:
        return False

def rowDetected():
    '''returns true if something was detected less than 15 inches away and is
no longer detected, like a tree'''
    global distPrevDetected
    distanceToTree = ultrasonicDriving.distance(INCHES)
    if (distanceToTree > 15) and ((distPrevDetected < 15) and (distPrevDetected
> 4)):
        distPrevDetected = distanceToTree

        print("row detected")
        return True
    else:
        distPrevDetected = distanceToTree
        return False
```

Code Excerpt #3: Another two functions using the ‘lift’ ultrasonic mounted at the top of the elevator shaft to prevent the elevator from going too high or low and driving off of the track.

```
def elevatorMoveTo(position):
    '''
    Moves the elevator to the designated distance at a preset speed.

    :position: The distance from the ultrasonic, in inches, the elevator will
move to. Should not be greater than 5 or less than 1.25'''
    while True:
        distError = position - ultrasonicLift.distance(INCHES)
        distEffort = 40 * distError
        elevator1.spin(REVERSE, distEffort)
        elevator2.spin(REVERSE, distEffort)
```

```
        # print(ultrasonicLift.distance(INCHES))
        if abs(distError) < .1:
            elevator1.stop()
            elevator2.stop()
            break
        sleep(20)

# Elevator Control
def elevatorControl(speed): #Direction = TRUE -> Move up, Direction = FALSE -> Move Down
    '''
    controls elevator speed but doesn't move further down than 5 inches from the ultrasonic or
    further up than 1.25 inches from the ultrasonic

    :speed: rate at which the elevator should move
    '''
    MAXDIST = 5
    MINDIST = 1.25
    #print("Elevator", ultrasonicLift.distance(INCHES))
    #print("Larger than min? ", ultrasonic.distance(INCHES) > MINDIST)
    #print("Smaller than max? ", ultrasonic.distance(INCHES) < MAXDIST)

    if ultrasonicLift.distance(INCHES) > MINDIST and speed > 0:
        elevator1.spin(FORWARD, speed)
        elevator2.spin(FORWARD, speed)
    elif ultrasonicLift.distance(INCHES) < MAXDIST and speed < 0:
        elevator1.spin(FORWARD, speed)
        elevator2.spin(FORWARD, speed)
    elif ultrasonicLift.distance(INCHES) > MAXDIST:
        # print("EXCEEDED MAX HEIGHT")
        elevatorMoveTo(MAXDIST)
    elif ultrasonicLift.distance(INCHES) < MINDIST:
        # print("EXCEEDED MIN DIST")
        elevatorMoveTo(MINDIST)
    else:
        elevator1.stop()
        elevator2.stop()
```

Code Excerpt #4: Three functions used when navigating the field that use the IMU and robot heading to help the robot navigate. The first, drive(), is the basic X-drive function. The second

two are used to spin to a designated heading without driving, and then spinning a number of degrees from the current heading, respectively.

```
def drive(rawDirection, rotationSpeed):
    '''
    X-drive function.

    :rawDirection: the x and y speeds, as a list formatted [xSpeed, ySpeed]
    :rotationSpeed: the speed at which the robot should rotate
    '''
    angle = math.radians(360 - IMU.heading())
    direction = [rawDirection[0] * math.cos(angle) - rawDirection[1] *
math.sin(angle),
                 rawDirection[0] * math.sin(angle) + rawDirection[1] *
math.cos(angle)]
    # print("Translated Direction: ", direction)
    frontRight.spin(FORWARD, direction[1] - direction[0] + rotationSpeed)
    backRight.spin(FORWARD, direction[1] + direction[0] - rotationSpeed)
    backLeft.spin(FORWARD, direction[1] - direction[0] - rotationSpeed)
    frontLeft.spin(FORWARD, direction[1] + direction[0] + rotationSpeed)


def spinToHeadingNoDrive(targetHeading): # tested & calibrated
    '''
    Spins to a heading using the IMU to track angle. uses proportional control

    :targetHeading: the heading the robot will spin to

    currently REDUNDANT
    '''
    Kp = .5
    error = targetHeading - IMU.heading()

    while (error > 3 or error < -3):
        if (error > 180): error -= 360
        elif (error < -180): error += 360

        effort = error * Kp

        drive([0, 0], effort)
        sleep(20)
        error = targetHeading - IMU.heading()
```

```
    drive([0,0], 0)


# positive degrees is right, negative is left (however, will always take
shortest path, so -270 and 90 will
     # go same direction if it starts in diff positions)
def turnDegrees(degrees): # tested
    '''
    turns the designated number of degrees from the current heading. It will
always take the shortest path,
    so passing in -270 and 90 will result in a turn of the same direction

    :degrees: the number of degrees from the current heading the robot should
turn
    '''
    desiredHeading = (IMU.heading() + degrees + 360) % 360
    spinToHeadingNoDrive(desiredHeading)
```

Code Excerpt #5: Excerpt from the Detecting Fruit state demonstrating use of vision to hone in on fruits.

```
objects = camera.take_snapshot(FRUITS[currentTargetFruit])
        # print(camera.take_snapshot(FRUITS[currentTargetFruit]))
        if objects:
            brain.screen.draw_rectangle(0, 0, 480, 600, Color.GREEN)
            if camera.largest_object().height > 80:
                # print("Object Found!")
                horizontalError =  157.5 - camera.largest_object().centerX
                horizontalEffort = .35 * horizontalError
                drive([0, -1 * horizontalEffort], 0)

                verticalError = (211 / 2) - camera.largest_object().centerY
                verticalEffort = 2 * verticalError

                elevatorControl(verticalEffort)

                if abs(horizontalError) < 7 and abs(verticalError) < 10:
                    stopAllMotors()
                    wait(500)
                    print("DETECTING_FRUIT -> APPROACHING_FRUIT")
                    state = APPROACHING_FRUIT
            else:
```

```
                # print("Object Lost!")
                elevatorMoveTo(3)
                drive([0, -50], 0)
```

Code Excerpt #6: Excerpt from the Approaching Fruit state demonstrating using vision to hone in on fruits.

```
objects = camera.take_snapshot(FRUITS[currentTargetFruit])
        if objects:
            if camera.largest_object().height > 80:
                verticalError = (211 / 2) - camera.largest_object().centerY
                verticalEffort = 2 * verticalError

                horizontalError =  157.5 - camera.largest_object().centerX
                horizontalEffort = .35 * horizontalError

                if camera.largest_object().width < 70:
                    # print('narrow fruit')
                    distanceError = 285*.75 - camera.largest_object().width
                    distanceEffort = .15 * distanceError

                else:
                    # print('wide fruit')
                    distanceError = 285 - camera.largest_object().width
                    distanceEffort = .15 * distanceError

                print('distance Error: ', distanceError)

                elevatorControl(verticalEffort)
                drive([distanceEffort, -1 * horizontalEffort], 0)

                # print("Vertical Error: ", verticalError)
                # print("Horizontal Error: ", horizontalError)
                # print("Distance Error: ", distanceError)

                if (abs(verticalError) < 20 and abs(distanceError) < 10 and
abs(horizontalError) < 15):
                    stopAllMotors()
                    print("APPROACHING_FRUIT -> COLLECTING")
                    state = COLLECTING
            else:
                # reverseFromFruit(-30)
```

```
                elevatorControl(0)
                print("APPROACHING_FRUIT -> DETECTING_FRUIT")
                state = DETECTING_FRUIT

        else:
            # reverseFromFruit(-30)
            elevatorControl(0)
            print("APPROACHING_FRUIT -> DETECTING_FRUIT")
            state = DETECTING_FRUIT
```

Code Exerpt #7: Code from the state machine and variable definitions that controlled the currentTargetFruit variable, either ending the run if it was over the max index of the list of vision signatures for the fruits, or incrementing it by 1 to move to the next signature.

```
# FRUIT WE START SEARCHING FOR
FRUITS = [LEMON, ORANGE_FRUIT, LIME]
currentTargetFruit = 0
.
.
if (currentTargetFruit > 2):
        print("PROGRAM COMPLETE\n", state, " -> IDLE")
        state = IDLE
        Break
.
.
.
elif (state == RETURN_TO_START):
            # drive toward wall
            driveStraight([0, 70], IMU.heading())
            sleep(100)

            if (backLeft.current() > 2) or (backRight.current() > 2):
                stopAllMotors()
                fruitNotDeposited = True

                # while True:
                #     drive([0, -20], 0)
                #     if (lineOnAlternateRight()):
                #         stopAllMotors()
```

```
                #         break
                #
                # wait(2000)
                print("RETURN_TO_START -> DRIVING_TO_ROW")
                state = DRIVING_TO_ROW
                currentTargetFruit += 1
                brain.screen.draw_rectangle(0, 0, 480, 600, Color.PURPLE)
                IMU.reset_heading()
                sleep(1000)
                print('new fruit: ', FRUITS[currentTargetFruit])
```