Final Design Report - MAE Capstone

Prof. Curtis Wilson
TA: Han Wang, Omar Curiel

Team 5

Naveed Torabzadeh
An Le
Cory Chilton
Jonathan Huang
Mathis Wouters

MAE-162E: Mechanical Product Design-I
Spring 2023

Mechanical & Aerospace Engineering Department
University of California, Los Angeles

6/16/2023

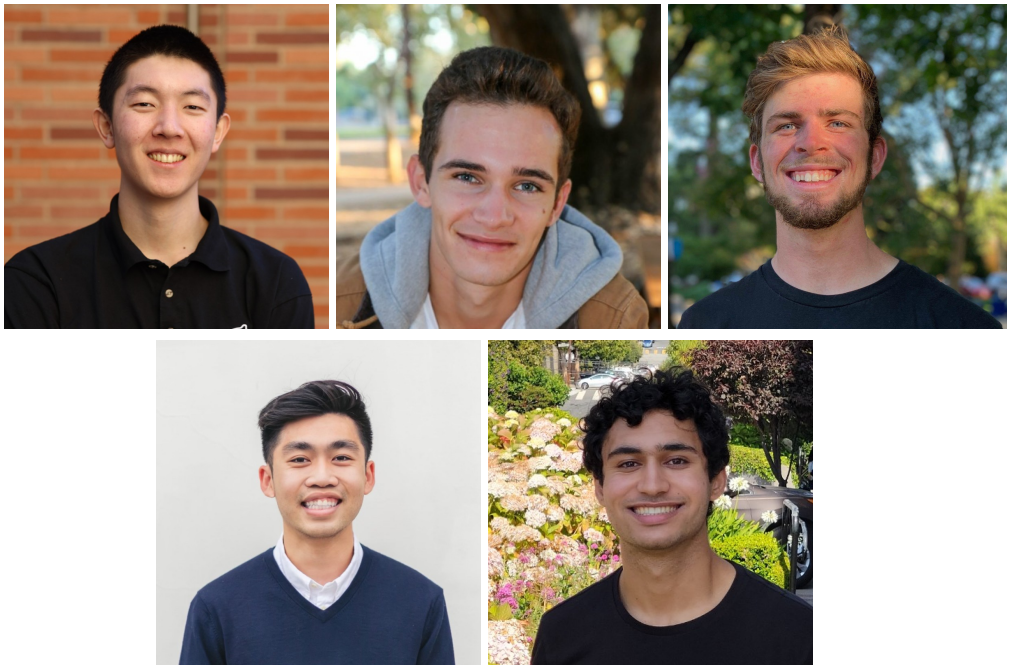


Figure 1: Team 5; team members pictured from left to right, Top row: Jonathan Huang, Mathis Wouters, Cory Chilton; Bottom row: An Le, Naveed Torabzadeh

Abstract

This report covers the concept development, design, building, and testing processes of an autonomous robot that is required to pick up and transport a juice box through a Venue of various obstacles and deliver it to a final destination. Background research was performed on similar robots that are present in industry environments. As part of the design process, High Level Design Requirements and Lower Level Design Requirements were developed. From these design requirements, four potential design concepts are presented based on team brainstorming. These concepts were then judged on an Objectives Tree, whose design factor weights were calculated using a Pairwise Comparison Chart. From this design analysis and preliminary testing, a four-bar linkage lift with an attached motorized gripper claw design on a four-wheel-drive chassis was selected as our team's final design for this project. A fully-featured CAD of the design was created and calculations were performed that suggested this was a viable robot design to move forward with. The robot was built using a combination of off-the-shelf, 3D-printed, and laser-cut parts, and its autonomous functions and capabilities were programmed using Arduino IDE. After thorough refinement of the code from testing the robot on the Venue, the robot is able to successfully satisfy all the Design Requirements and meet all the objectives of the project.

Table of Contents

List of Figures

List of Tables

List of Symbols

Symbol	Meaning
L	Wheelbase
L_c	Distance RW to CoM
h_c	Height of CoM
θ	Angle of incline/bumps
μ	Static coefficient of friction
γ_r	Drive system factor for RW
γ_f	Drive system factor for FW
$F_{NR} = N_r$	Normal force RW + ground
$F_{NF} = N_f$	Normal force FW + ground
$F_{mg} = F_m$	Force of gravity
F_{TR}	Tractive force for RW
F_{TF}	Tractive force for FW
$F_{//}$	Force of gravity along slope
$F_{\perp}$	Force of gravity normal to slope
N	Normal force
f_s	Force due to static friction
g	Acceleration due to gravity
μ_s	Coefficient of static friction
m_j	Mass of the juice box
m_B	Mass of one bar in the four bar mechanism
m_p	Mass of the plate and attached axles in the four bar mechanism
m_g	Mass of the gripper sub assembly

r_j	Horizontal distance from the geared four bar axle to the CG of the juice box, when the four bar mechanism is in a horizontal position
r_B	Horizontal distance from the geared four bar axle to the CG of one of the bars, when the four bar mechanism is in a horizontal position
r_p	Horizontal distance from the geared four bar axle to the CG of the plate and attached axles, when the four bar mechanism is in a horizontal position
r_g	Horizontal distance from the geared four bar axle to the CG of the gripper assembly, when the four bar mechanism is in a horizontal position
T	Minimum torque necessary to raise four bar mechanism

1. Introduction

1.1 Problem Statement

The robot being proposed is tasked with picking up an item from “the warehouse”, navigating a course and series of obstacles with the item, and delivering the item safely to a specified location. Specifically for this project, the robot must remove a juice box with pre-inserted straw from a shelf, navigate the course and obstacles with the juice box, and finally deliver the juice box to a drop-off zone on the course, all while not having any juice spill from the open juice box. This must all be done autonomously by the robot. The course and obstacles that the robot must navigate include: two 90-degree turns, static obstacles including some height constraints, a dynamic (moving) obstacle, and a rough “washboard” surface on the course. There will be multiple possible options for where the juice box can be picked up from and where it can be delivered, and the robot must be capable of completing any combination of these locations.

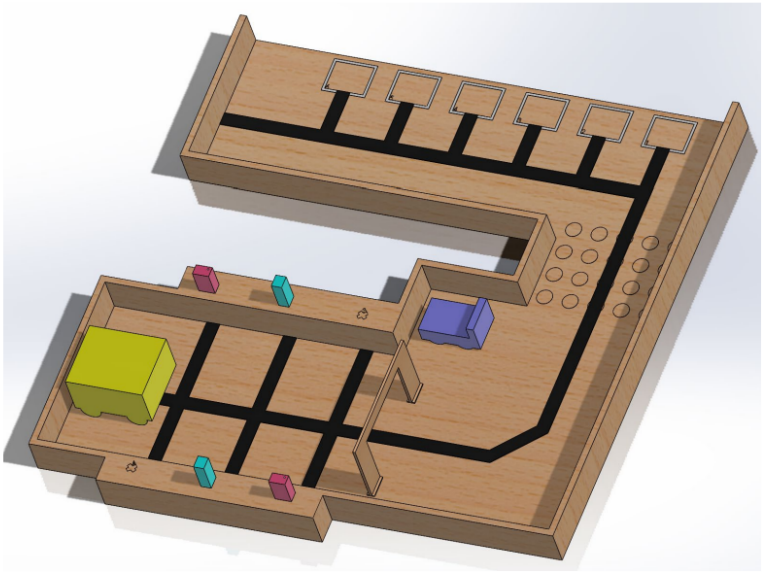


Figure 2: The Venue (course) that the proposed robot (yellow) must navigate. It must pick up the item (red or blue), pass through the doorway, complete the turn, navigate safely past the moving obstacle (purple), drop off the item in the correct drop-off location, and end safely at the back wall of the course. The black line on the course is 2 inches thick and is used to aid in the robot’s autonomous line following.

The robot solution for this project will take some inspiration from similar-function industry robots, and ideally represent a solution that is both simple to create and manufacture and robust for achieving all of the project tasks. The robot will be made from scratch using the given budget to buy equipment and hardware, and making use of 3D printing and fabrication resources available in our lab. It is desired by our team for the robot to perform the tasks at a reasonable pace, and ideally complete all objectives and navigate the course within 2 minutes. The speed of the robot will be mainly determined by hardware selected and can be adjusted to a certain degree using software.

1.2 Literature Review

The robot being designed for this project is very similar in function to warehouse robot devices used by companies like Amazon. The robots used by Amazon either carry individual packages or carry heavier crates and cages filled with many packages [1] [2]. Similar function delivery robots have also been popularized by companies like Starship in places like college campuses to deliver food or even groceries [3]. A photo of a Starship delivery robot is below. Although Starship robots require human loading and unloading, unlike this project's robot which will load and deliver the item autonomously, there are still areas that can be learned from like the drivetrain and environment sensing of the robot.

Figure 3: Starship delivery robot on UCLA campus

Starship robots and Amazon's "Scout" robots use cameras and ultrasonic sensors to take in the environment and then use software to take in this sensor input and plot a path for the robot to take. Additional sensing tools such as radar are also commonly used in the industry [4]. Our robot will use similar environment sensing, but the course and environment will be less complicated and more predictable than the real-world. Compared to industry robots performing similar functions, our robot will be smaller, lighter, and faster, but this is of course because it is not expected to handle a heavy load.

For this project specifically, the robot will follow a black line that is laid out on the course. This line following will allow the robot to autonomously navigate the course and reach its desired locations. Line following robots can be pretty simple and achieved using an Arduino microcontroller and IR transmitter and receiver sensors [5]. For the case of a black line on the course, the robot uses several IR transmitters (photodiodes) to send out light and depending on whether the light is reflected back and received by the IR receivers (indicating a non-black reflective surface) or if the light is absorbed by the black colored line on the floor, the robot will have a sense of where it is oriented on the course. Based on which sensors receive light and which ones do not, the robot is able to turn accordingly to orient itself with the black line and proceed with its path.

1.3 High-Level and Low-Level Design Requirements

1.3.1 High-Level Design Requirements

The High Level Design Requirements are the project design requirements given in class that are the overarching requirements governing the robot design.

The High Level Design Requirements are as follows:

Table 1: High-Level Design Requirements

HLDR	Description	Comment
1	Device must complete the full task set	Go to shelf location, remove item, lower item through doorway, navigate two 90-degree turns, avoid all obstacles, deliver item to destination in the upright orientation, stop after delivery
2	Device must operate autonomously	Can tell the device results of the die rolls before start of the run and press start
3	The device must complete the task set within 10 minutes	Any and all trials must be finished in this time limit
4	Starting dimensions must be limited to 20x25x30 cm (HxWxL)	Dimensions can change afterwards
5	Electric batteries are the only source of power for the device	Choose between disposable and rechargeable batteries (no mixing) Disposable: Ten 1.5V plus two 9V batteries (12 total) Rechargeable: Three 3.7V, three 9V, and six 1.2V batteries (12 total)
6	All sensors must be mounted on the device	Should not be a problem
7	Device may not leave behind any parts or components	Should not be a problem
8	Actuators must be designed and fabricated by the team	Only motors are excluded from this requirement
9	Purchasing budget is limited to $500 per team	Microcontrollers, routers, cables provided do not count towards this limit

1.3.2 Low Level Design Requirements

The Low Level Design Requirements are set by the team and are more tangible goals that will help the robot achieve the High Level Design Requirements.

Table 2: Low-Level Design Requirements

LLDR	Description	Comment
1	Device must have locomotion, lifting capabilities, and grabbing capabilities. Device needs DC motors to create motion/actuation.	Challenge requires the object to be moved and manipulated, meaning the robot needs to be able to have motion and control over the object.
2	Device must be able to be programmed for autonomous motion, and use sensory feedback to update its decision in real time. Needs 1 ultrasonic sensor, 3 IR sensors, and encoders on each wheel. An IMU would be nice to have.	There are moving obstacles on the course, which require sensors to update the robot's path.
3	Device must have high speed motors to traverse the course quickly.	Robot size is small relative to the size of the field.
4	Lift must start in smallest configuration, but can expand outwards after the start.	No major protrusions at the start of the challenge.
5	Electric DC batteries in series or parallel to meet required voltage/current/capacity requirements of the robot.	Electric batteries must fit within the starting robot size boundary.
6	Device needs sensors to detect objects in front of it, position on the track, and lines on the track.	Robot needs to know if it's off track so it can self-correct.
7	Device needs bolts, fasteners, adhesives, zip ties, etc. to securely mount all components. Important components should have secondary retention.	Components like wheels, lifts, etc. should have two forms of retention in case one fails. Auxiliary components like battery mounting don't need to be as robust.
8	The team must design a lifting mechanism powered by rotary electric motors.	The lift will convert rotational motion of the motor into linear motion of the juice box.
9	The team must make use of cheaper components, instead of more expensive components that could theoretically make the design challenge easier.	$500 is a relatively low budget for this project compared to industry robots, forcing us to find creative and cheap solutions.
10	Robot is robust and able to accommodate variations in starting position, juice box size, and bumpiness variation in the course.	It is desired for the robot to be able to work with imperfect course conditions and starting conditions. This will give the greatest chance of success.

Team personal target goals:

1. Complete the tasks and the course in less than 3 minutes (team goal). This will allow enough time to re-do a trial if there is a failure during operation.
2. Have the smoothest driving robot out of all the other robots. Avoid lots of jittering on the line-following for autonomous driving. This will involve having a good control system set up working with the IR light sensors used for the line-tracking.

3. A final goal is to have the robot capable of operating with any combination of pickup and delivery locations without having to edit the robot's code after the sites are chosen randomly but before the trial begins. This may require a camera and more complex vision system. Work on this will only be attempted after the robot is capable of meeting the basic requirements.

2. Design Description

2.1 Design Concept Development

2.1.1 Design Concept 1

Design concept 1 focuses on a simple forklift style configuration. A single stage rack and pinion lift system mounted at the front allows vertical motion of a gripper claw assembly. The lift system is supported by two support arms mounted to the chassis. Two gearbox and brushless motor units power the forward wheels, and a free spinning 360° caster wheel provides a free range of motion of the back end of the robot. The forward wheels are intentionally large, and the gearbox is geared down to allow for fine tuned position control. A forward mounted light sensor allows for ground line tracking and navigation through the venue. The battery housing is mounted towards the rear as a counterweight to the load, and the microcontroller housing is mounted centrally for streamlined wiring and vehicle weight distribution. A single large chassis plate provides all mounting points. This concept was conceived because it is relatively simple in design but able to complete all the required operational tasks.

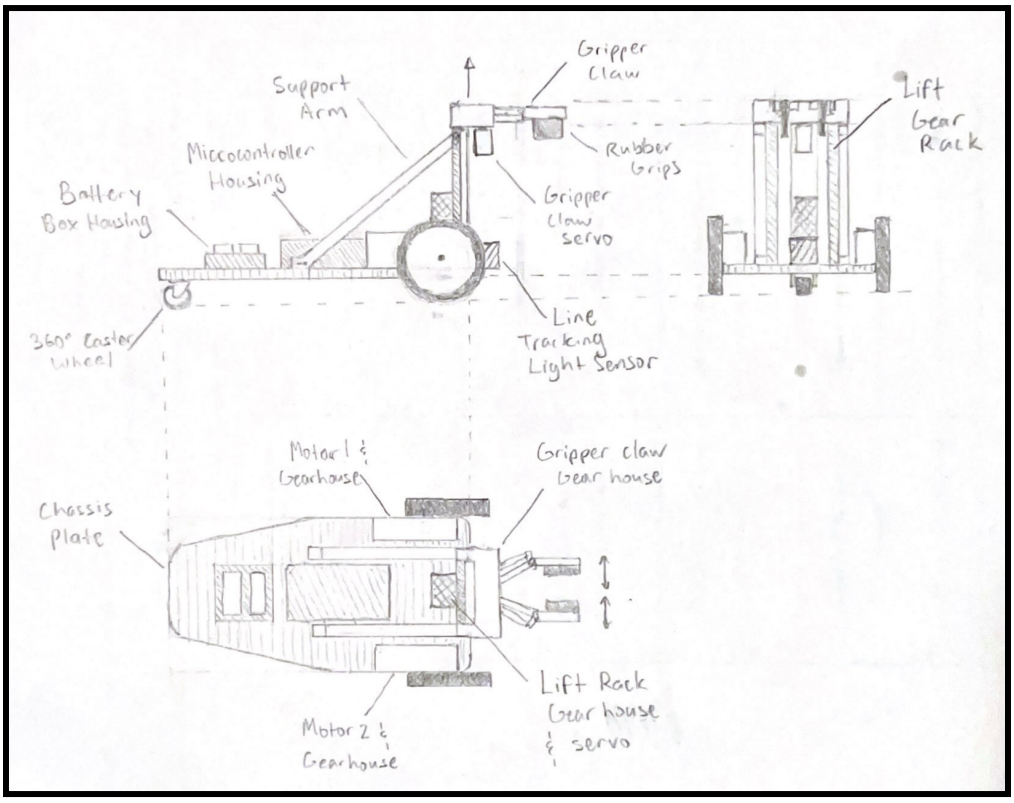


Figure 4: Concept Sketch 1 – forklift style. 2 front wheels with motors and 1 rear passive caster wheel.

Concept 1 is predicted to achieve all the HLDRs and most of the LLDRs. It is predicted that the hardest LLDR to achieve will be getting the forklift design to reach the height necessary to grab the juicebox from the elevated surface but have the mechanism also fit in the required starting dimensions. It is expected that Concept 1 will give the fastest course navigation because of the added maneuverability that this particular wheel setup provides. If speed is a priority, this will be a good choice. However, one concern is that the rear passive wheel will not navigate the bumpy part of the course well and also inhibit some of the robot's turning.

2.1.2 Design Concept 2

Concept 2 consists of a 4 wheel drive base, four-bar linkage lift, and standard claw. The drivebase contains 4 wheels and 1 motor directly powering each wheel. This would give us the maximum control over each wheel, and provide the correct amount of torque necessary for any scenario. The four-bar linkage can start in a low configuration, and then expand to reach the necessary heights. It also keeps the juice box level, making sure there are no spills. Since the juice boxes vary in size, a simple claw design allows the robot to grab objects of various sizes. The claw will have grippy material on the inside surface of the claw, and the amount of force it applies to the object can be controlled by how much the motor closes the claw using some sort of force sensor on one of the claw faces. The battery and microcontroller housings are placed in the center and middle of the robot to keep the CG low and point of rotation centered.

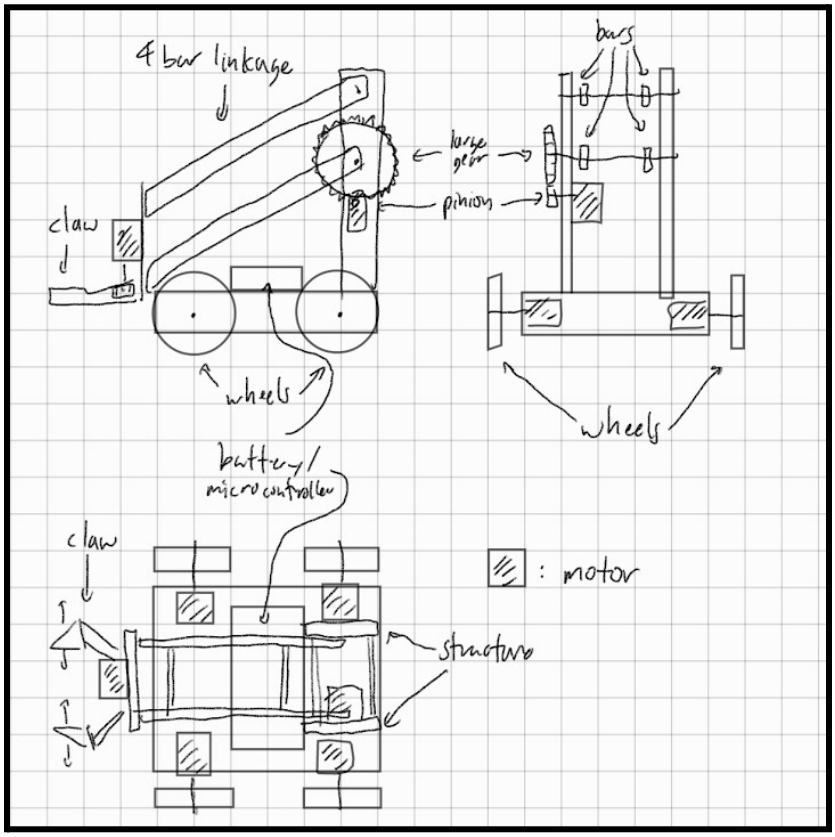


Figure 5: Concept Sketch 2 – 4-bar style configuration. 4x4 drive (4 wheels, 1 direct driven motor per

wheel).

Concept 2 is predicted to achieve all the HLDRs and most of the LLDRs. It is predicted that this design will have slightly less maneuverability than Concept 1, because of the limitations of a rigid 4-wheel design. This could be alleviated by using omni-directional wheels. Concept 2 seems to be a robust yet simple option with a proven 4x4 drivetrain design and a reliable claw grabbing mechanism. The biggest challenge with this concept will be designing and constructing a reliable 4-bar lifting mechanism and making sure the motors attached to the lift provide enough torque.

2.1.3 Design Concept 3

This design concept differs from the other design concepts because it uses treads instead of wheels, a roller intake instead of claws, and it has a stationary platform for the juice box to sit on throughout the course. The tread concept was derived from the idea that it would be effective for traveling across the bumpy section of the track, and it is an interesting alternative to consider instead of wheels. The roller intake was conceptualized because it is a good way to slowly move an object without applying too much squeezing force, preventing any spillage. Finally, the platform is an obvious place for the juice box to sit so that it does not fall whilst traveling through the course. This is a potential issue with the claw design: if the claw holds the juice box too loosely it may fall while the robot jerks or goes over bumps, but if it holds it too tightly it may squeeze juice out of the box. This concept is predicted to perform well with respect to most of the high level design requirements, the main concerns being the relative difficulty of implementing treads instead of wheels, and the risk of the juice box falling over when the roller intake tries to draw it in and/or deliver it at the end of the course.

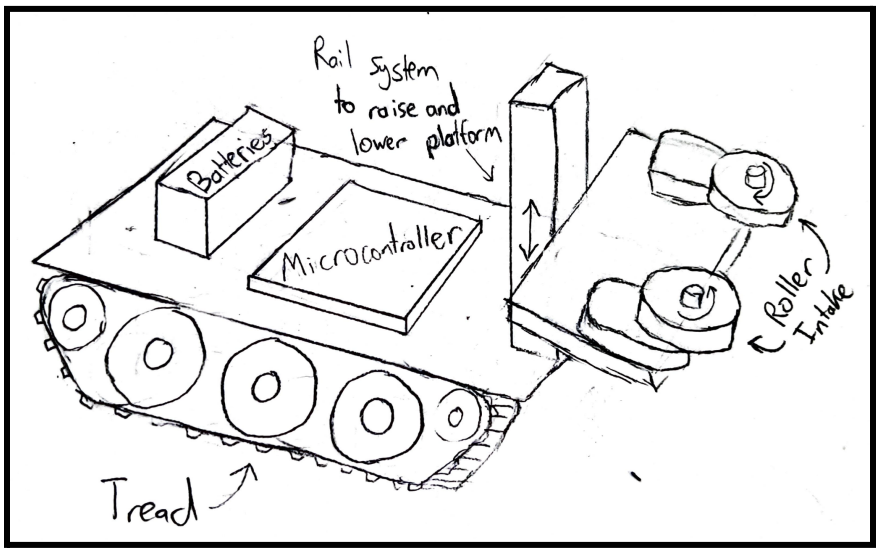


Figure 6: Concept Sketch 3, with major changes including the use of treads and a roller intake.

Concept 3 is predicted to achieve all the HLDRs and most of the LLDRs. This design is predicted to be much less maneuverable and heavier than Concept 1, but only slightly less maneuverable and slightly heavier than Concept 2. It is predicted to be able to handle bumpy terrain better than the other concepts. The effectiveness of the roller intake instead of using a claw is the biggest uncertainty with this design from a preliminary perspective.

2.1.4 Design Concept 4

Concept 4 was developed through iterative design of the drivetrain and lift arm. While designing the caster wheel, it was realized that like all free spinning caster wheels, they would attempt to reorient themselves 180 degrees when the robot backs up. This is a very unpredictable maneuver that could throw off the robot's autonomous trajectory. Thus, a system was developed where the axle of the caster wheel clevis is attached to two stationary pins on either side of the clevis using rubber bands. This will allow the robot to turn normally while preventing it from reorienting its wheels 180 degrees while backing up. It will also limit the amount of travel caused by driving over rough terrain, decreasing autonomous trajectory error. When redesigning the lift arm, it was discovered that if one was to shorten the drivebase, there would be enough room to fit the gripper claw within the dimensional envelope in order for a single four bar linkage to be used. This simple system was extremely desirable. However, due to the increased height of the center of gravity, it was decided to use two caster wheels in the back of the robot instead of one central unit, in order to increase the overall stability of the robot.

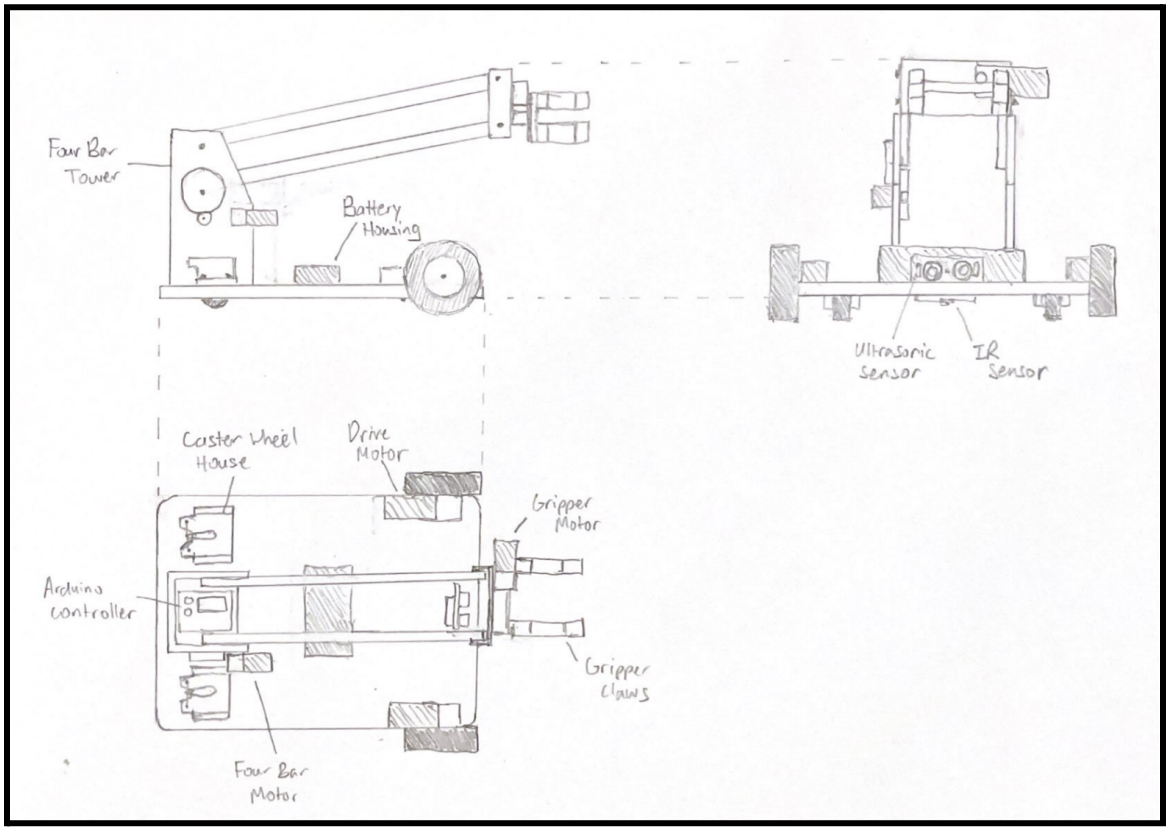


Figure 7: Design Concept 4 sketch with single four-bar lift and double limited rotation caster wheels.

2.1.5 Pairwise Comparison Chart and Objectives-Tree

Table 3: Piecewise Comparison Chart with our selected design factors detailed and compared relative to each other. The weights in the blue column are calculated and used in the Objectives Tree below to compare our three design concepts. A larger weight indicates higher importance.

PAIR-WISE COMPARISON CHART

# of Designers / Scale	
5	max rating

max rating is above; row factor is compared to column factor; a rating of half the max rating means the 2 factors are of equal importance; more than half means row factor is more important, etc

	Is this Factor Used? (1 = YES, 0 = NO)	Factors (aka subfunctions)	Compared To: Starting dimension	Operate autonomo	Purchasing budget is	Able to grab and	Electric batteries	Aesthetics	Durability	Weight	Speed of performan	Maneuver ability	Able to properly	Total	Weight
Asset	1	Starting dimensions of robot must be 30x25x20cm (L*W*H)	5	2.5	2.5	2.5	2.5	5	4	5	5	4	2.5	40.5	0.12
	1	Operate autonomously	2.5	5	2.5	2.5	2.5	5	4	5	5	4	2.5	40.5	0.12
	1	Purchasing budget is $500	2.5	2.5	5	2.5	2.5	5	4	5	5	4	2.5	40.5	0.12
	1	Able to grab and hold the juicebox and move without spilling	2.5	2.5	2.5	5	2.5	5	4	5	5	4	2.5	40.5	0.12
	1	Electric batteries are only source of power	2.5	2.5	2.5	2.5	5	5	4	5	5	4	2.5	40.5	0.12
	1	Aesthetics	0	0	0	0	0	5	0	0	0	0	0	5	0.02
	1	Durability	1	1	1	1	1	5	5	4	4	2.5	1	26.5	0.08
	1	Weight	0	0	0	0	0	5	1	5	2.5	1	0	14.5	0.04
	1	Speed of performance	0	0	0	0	0	5	1	2.5	5	1	0	14.5	0.04
	1	Maneuverability	1	1	1	1	1	5	2.5	4	4	5	1	26.5	0.08
	1	Able to properly release and deliver the juicebox	2.5	2.5	2.5	2.5	2.5	5	4	5	5	4	5	40.5	0.12
													Total Sum:	330	1.00

Table 4: Objectives Tree that is used to assess and identify the best design concept to proceed with. The weights for each design factor were calculated using the Piecewise Comparison Chart. Concept 4 has the highest total score.

Pairwise Comparison Chart
Group 5 MAE 162D Winter 2023
Professor Wilson

OBJECTIVES TREE									
	max rating	10							
		Design Scores				Weighted Design Scores			
Design Objective Factors	PCC Weight	Design 1	Design 2	Design 3	Design 4	Design 1	Design 2	Design 3	Design 4
Starting dimensions of robot must be 30x25x20cm (L*W*H)	0.12	10	10	10	10	1.23	1.23	1.23	1.23
Operate autonomously	0.12	10	10	10	10	1.23	1.23	1.23	1.23
Purchasing budget is $500	0.12	10	10	10	10	1.23	1.23	1.23	1.23
Able to grab and hold the juicebox and move without spilling	0.12	10	10	10	10	1.23	1.23	1.23	1.23
Electric batteries are only source of power	0.12	10	10	10	10	1.23	1.23	1.23	1.23
Aesthetics	0.02	3	5	7	5	0.05	0.08	0.11	0.08
Durability	0.08	5	5	5	5	0.40	0.40	0.40	0.40
Weight	0.04	7	5	3	5	0.31	0.22	0.13	0.22
Speed of performance	0.04	5	5	5	5	0.22	0.22	0.22	0.22
Maneuverability	0.08	8	5	3	8	0.64	0.40	0.24	0.64
Able to properly release and deliver the juicebox	0.12	5	10	8	10	0.61	1.23	0.98	1.23
	1.00			Total Score:		8.37	8.68	8.22	8.92
(1) Evaluation marks 0 - unaccepatable, 1-3 still acceptable, 4-6 fair, 8-9 good, 10 very good (optimum solution)									
(2) Total sum of weighing factors is unity,									
(3) Order of Merit: 1 highest; 10 least									

2.1.6 Design Selection

Four initial design concepts (hand sketches labeled Design Concepts 1, 2, 3 and 4) for the robot were developed. These four designs were analyzed and compared using a Piecewise Comparison Chart and an Objectives Tree (see above). These tools helped identify important design factors, weigh the importance of each factor relative to the other factors, score each design concept based on its ability to achieve the design factors we came up with, and ultimately select a final design concept to move forward with for this project. Key robot requirements such as maximum size and key functions such as picking up the item, carrying it safely, and delivering it accurately, were weighted the most amongst the design factors. Design factors such as aesthetic quality and speed of the robot were weighted much less to reflect that achieving good results in these factors is not mandatory and required for the robot to succeed in its given tasks related to the item delivery and meet the mission requirements.

Based on the Pairwise Comparison Chart, the design objective factors with equally the greatest weight are the starting dimensions of the robot, being able to operate autonomously, the purchasing budget, being able to grab and hold the juicebox and move without spilling, using electric batteries as the only source of power, and being able to properly release and deliver the juicebox. These are built into the design requirements and thus hold the greatest weight. The durability and maneuverability of the robot have the next greatest weight, followed by the weight and speed of performance of the robot, and finally the aesthetics, the last of which is a design objective factor that holds almost no weight given the purpose of the robot.

The four design concepts were designated scores in the Objective Tree based on the Pairwise Comparison Chart weights. Design Concept 4 was ultimately chosen because it scored the highest of the four design concepts with a total score of 8.92, compared to 8.37, 8.68, and 8.22 for Design Concepts 1, 2, and 3, respectively. Design Concept 4 scored equal or better compared to the other designs in all design objective factors, except for the aesthetics and weight factors, which had lower weight than some of the more important factors. While all four designs scored essentially the same for durability, speed of

performance, and ability to accomplish the critical tasks related to the juicebox (except Design Concept 1), Design Concept 4 came out on top primarily due to its overall better maneuverability.

After more brainstorming and reviewing the Venue (the course that the robot needs to navigate through and grab/deliver the juice box in), it was decided that the single stage "forklift" rack and pinion lift mechanism of Concept 1 would not be sufficient in achieving the height necessary to pick up the juicebox from the shelf. Thus, the four-bar linkage lift mechanism was decided on to move forward with.

Although the Objectives Tree and initial design analysis states that Design Concept 4 is the one to move forward with, it was found in our initial testing of its drivetrain design (two front wheels with motors and two passive caster wheels in the rear), that the robot could not turn consistently and line-follow well using this drivetrain setup. **Thus, it was decided that the drivetrain of Design Concept 2 (traditional 4x4; two front wheels and two rear wheels, with a motor directly attached to each) would be the one to move forward with for our robot. The rest of the design elements of Design Concept 4, specifically the four-bar linkage lift mechanism and the gripper claw, were retained and kept in the final design that we went with for the robot.** This trial and error experience showed that pre-build design analysis is not perfect and cannot always be 100% relied upon to give the design solutions. Once the drivetrain concept of Design Concept 4 was built and tested, it was realized that it would not work as well as a traditional 4x4 wheel design, and so we had to be flexible with our design and adjust accordingly. That being said, the pre-build analysis was definitely very useful in helping brainstorm and decide upon other key elements of our design, including the lift and the item gripper.

2.2 Design Overview

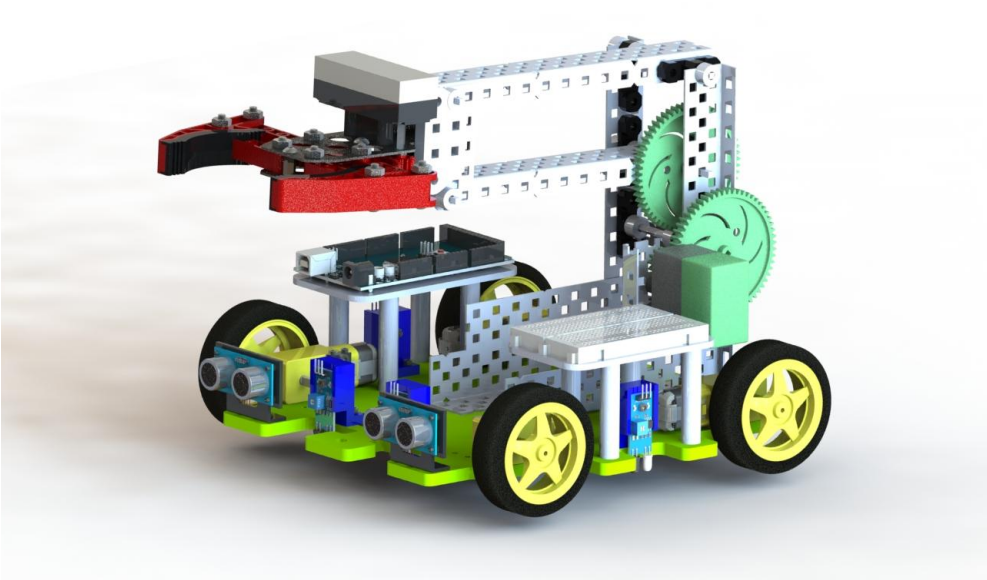


Figure 8: CAD model of the final designed robot

The final robot design features a four-wheel-drive (4x4) drivetrain, a four-bar linkage lift, and a gripper claw. There are four drive motors, one lift motor, and one gripper claw motor. For achieving autonomous driving, the robot has two forward-facing ultrasonic sensors and four IR reflective photoelectric line-following sensors. The robot uses a 7.4V battery pack and an Arduino Mega 2560

microcontroller board. All these components are mounted onto a ¼" thick acrylic chassis plate (two stacked ⅛" thick acrylic layers) that has been custom laser-cut with the appropriate mounting holes. The IR sensors, microcontroller board, and breadboard each have their own custom designed and 3D-printed mounts that are attached to the chassis to hold the components in their respective locations.

The four-bar linkage lift mechanism of our robot is built using aluminum bars, C-channels, and various hardware/fasteners from a VEX Robotics kit. These linkage pieces have been modified in the machine shop to have their lengths shortened to fit the size requirements and design of our robot. VEX spur gears, shafts, and a 2-Wire Motor 393 [8] are used on this mechanism. The gripper claw assembly (including gears and mounting aluminum pieces) is also from the VEX Robotics kit we purchased. However, the gripper claw pieces that grab the item have been modified and swapped out with our own designed and 3D-printed pieces (see detailed gripper subsystem description in the next report section). This helps in specifically grabbing the juicebox that we have been tasked with.

The drivetrain will use four of the "ELEGOO Smart Robot Car Kit V4.0" drivetrain motors that were provided to our group. These are ADAFRUIT brushless DC motors (full datasheet link). We did not use encoders on our drivetrain motors, but if needed we planned to purchase and attach encoders to the kit motors we already have or possibly purchase a motor with an integrated Hall effect encoder such as the BEMONOC 25GA370 DC Encoder Motor (found here) that was analyzed and tested in our lab.

The final design uses four HiLetGo IR "Infrared Obstacle Avoidance Sensor" line-tracking modules [9] and two HC-SR04 ultrasonic sensors [10] to perform the autonomous course navigation. The ultrasonic sensors will be mounted on the front of the robot with a clear line of sight to the course in front of the robot. These sensors will be used for distance perception at the front of the robot, to mark different stages of the course, and obstacle detection. The IR line tracking modules will be mounted on the bottom of the chassis plate at the front (two in front) and sides (one on each side) of the robot, pointed at the ground. These sensors will be used to keep the robot following the black line that will be on the floor of the Venue course, as well as aid in branch detection and selection for the different pickup and dropoff locations.

2.3 System Specification

Table 5: High-Level System Specifications

	Specification	Reasoning
Weight (kg)	1.99 (4.39lb)	Minimize weight to reduce load on motors
LxWxH (cm)	30 x 23.7 x 19.5 (11.8in x 9.33in x 7.68in) *Most compact configuration*	This fits in the max dimensions requirement
Cost (USD)	$275 (see BOM section)	Fit within the given $500 budget; the allocated budget is "use it or lose it"

Chassis	¼ in. thick acrylic baseplate (2 layers, each layer is ⅛ in. thick)	Rigid, easy to manufacture/iterate, can laser-cut and include component mounting features
Drive	4x4 (4WD)	Precise turning and control over rough surfaces
Lift	1-motor 4-bar	Simple, adequate range, keeps object level, OTS parts
Grabber	1-motor claw attached to 4-bar lift	Simple, works with varied objects
Electronics	Arduino Mega 2560; motor shield	Increased I/O pins, shield directly connects to drive motors

2.4 Mechanical Systems

Our robot moves around using a 4WD drivetrain and uses its two ultrasonic sensors and four IR object detection sensors to autonomously navigate the Venue and line-follow the black line on the ground. The four-bar linkage lift is powered with one VEX motor (see References) and begins in its lowest configuration to give the robot the smallest dimensions for its starting configuration. The robot then drives up to the correct item, detecting and selecting the correct pickup branch using the side IR sensors. The lift comes up to the proper height to pick up the item off the shelf, the gripper claw opens up to its widest position, and then the robot drives up to the shelf to pick up the item. The gripper claw closes on the object, applying a delicate pressure to provide a good grip but not squeeze the item. The robot then returns to its line following on the course and lowers its lift mechanism so that the entire robot can fit underneath the doorway obstacle. Refer to the figures below for a detailed look at the drivetrain, lift, gripper, and electronics/sensor setup.

After passing through the doorway obstacle, the robot uses its autonomous driving sensor setup to continue line-following until it detects the moving obstacle in front of its path. At which point the robot stops until the object leaves its path. Then, the robot continues line following and brings the lift assembly and claw holding down so that it is more secure when traversing the bumpy terrain on the Venue. After getting through the bumpy terrain, the robot uses its side IR sensors to detect which branch it needs to drop off the item at and then slowly lowers the lift and opens the gripper to release the item in the designated drop-off section. After the item drop off, the robot continues to the end of the track and stops at the end wall.

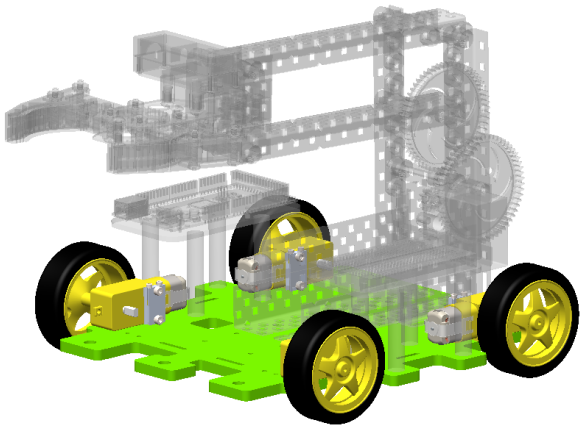


Figure 9: Drivetrain and chassis of the robot highlighted in a CAD render of the robot. It is a 4WD drivetrain. In gray is the rest of the robot, which is all mounted to the chassis and drivetrain.

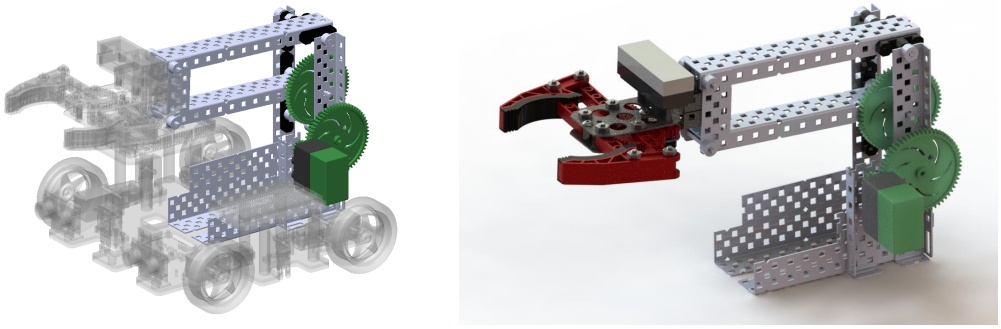


Figure 10: The four-bar linkage lift mechanism by itself (left) and with the gripper claw assembly attached to it (right).

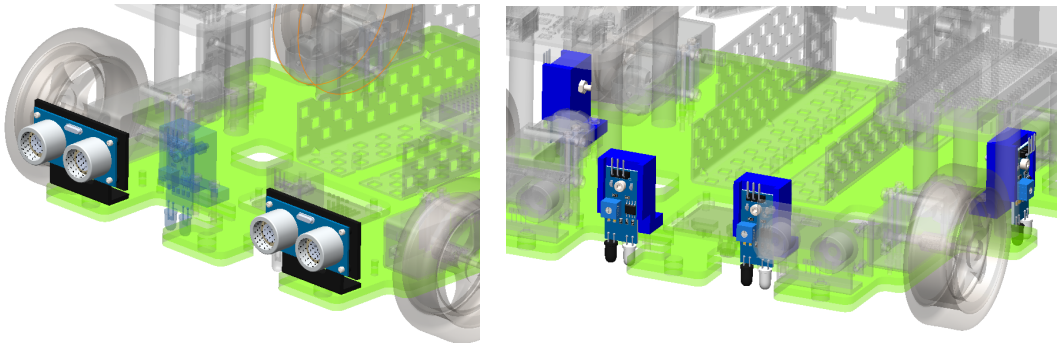


Figure 11: The front ultrasonic sensors (Left) and 4 IR line-following sensors (right) mounted on the chassis of the robot.

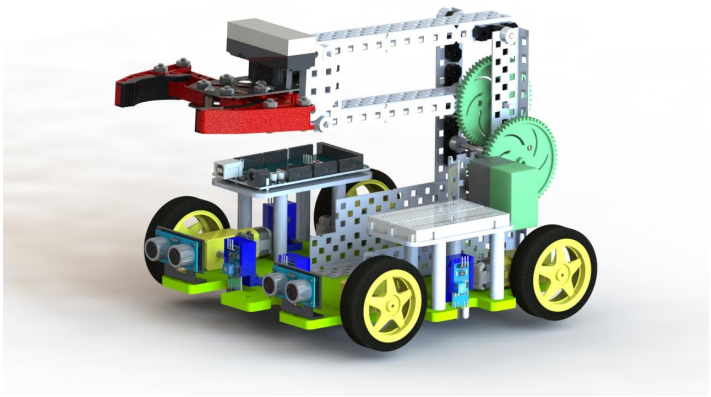


Figure 12: The full robot assembly showing all the subsystems, electronics, etc. assembled together.

2.5 Control Systems

The autonomous capabilities of our robot were all programmed in Arduino IDE. Code was written to direct the robot to complete all the tasks outlined in the Design Requirements. The robot code has eleven stages in total, and the entire Code Structure and logic is detailed in Section 5 of this report. Essentially, the robot uses its IR sensor to line follow and turn on the correct branches to grab and drop off the item. The ultrasonic sensors are used to position the robot correctly at the shelf to pick up the item and for obstacle detection, in particular the moving obstacle. The ultrasonic sensors are also used in the control system and code of the robot to stop the robot at the back wall and terminate the program.

The robot's program and autonomous movement is initiated by user input into an IR remote, with two digits of input given to signify the pickup and dropoff locations that it will need to visit. Then the robot begins stage 1. The code then goes through the code stages in a series programming structure. The code for each stage loops until it is completed and then it turns itself off and starts the next stage.

3. Subsystem Design Description

3.1 Chassis

3.1.1 Subsystem Description

The goal of the chassis design was to achieve a simple and rigid way to mount all other subsystems to the robot. Thus, we chose to use a 2D, lasercut profile to allow us to rapidly prototype different chassis base plate designs based on the iterations of the lift, electronics packaging, and drivetrain. The profile is constructed using 2 acrylic plates, as the acrylic is easily accessible from the Makerspace, and the 2 plates allow for recessed areas in the chassis for mounting various objects. We also found that only 1 sheet of ⅛ in thick acrylic was not rigid enough for our requirements. Doubling the thickness of the chassis increases our bending rigidity by a factor of 8.

3.1.2 Design Requirements

Table 6: Chassis Subsystem Design Requirements

	Requirement	Reasoning
Weight (kg)	0.32	Minimize weight to reduce load on motors
LxWxH (mm)	235 x 235 x 6.35	Square drive base allows for smooth point-turns
Material	Acrylic	Rigid, easily obtainable
Construction	2X stacked 3mm pieces	Increased stiffness, allows for internal pockets

3.1.3 Subsystem CAD Models and Engineering Drawings

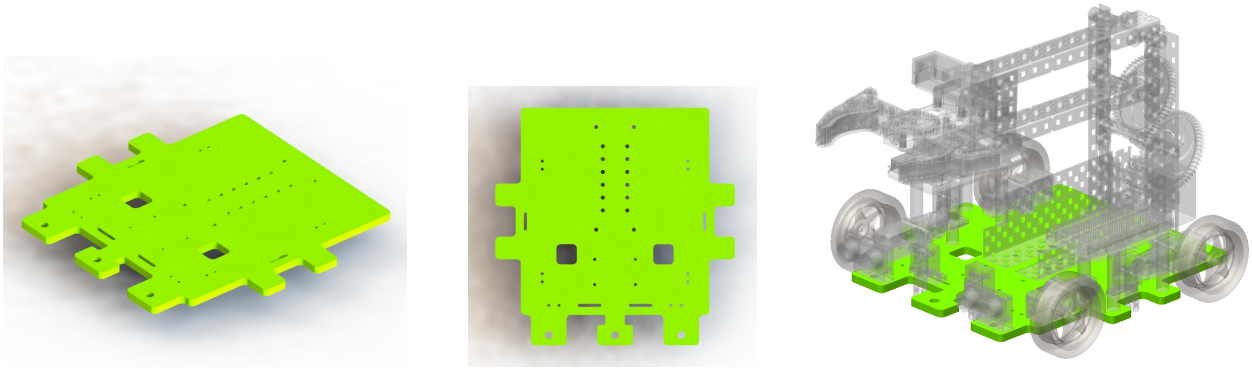


Figure 13: In order of appearance: isometric, top, and integrated views of the chassis assembly.

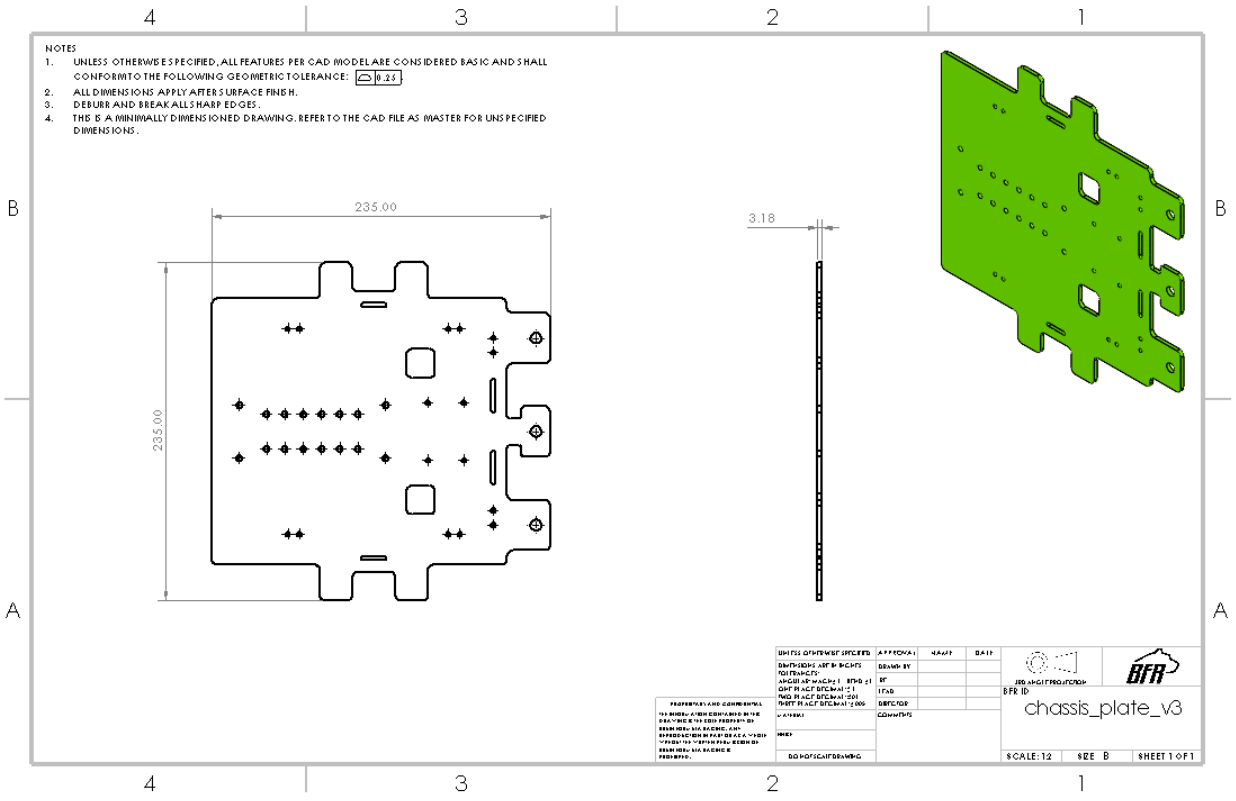


Figure 14: Engineering drawing of one chassis plate. Two plates are used in the final robot and stacked on top of each other to increase the overall chassis plate thickness and strength

3.2 Drivetrain

3.2.1 Subsystem Description

The drivetrain, similar to the chassis, is as simple as possible to minimize points of failure and make it easy to program and maneuver. Thus, given our design requirements, we chose to do a 4x4 (4 wheels direct driven by 4 motors) drive. The motor/gearbox/wheel assembly is directly from the Elegoo kit bot. Given our experience with programming the kit bot, we knew directly swapping the drivetrain onto our robot would be a smooth transition. The direct-driven wheels also allow for increased grip over bumpy sections and turning about the center-point of the chassis for smooth point-turns.

3.2.2 Design Requirements

Table 7: Drivetrain Subsystem Design Requirements

	Requirement	Reasoning
Wheels	66mm OD traction wheels	Available from kit bot, increased grip over bumpy section
Motors	4X DC motors	Available from kit bot, pivot about center of chassis
Gear Train	Integrated planetary gearbox mounted to each motor	Available from kit bot, reduces free speed of motors, increases torque
Construction	Gearboxes bolted directly to chassis	Simple, rigid, easy to assemble

3.2.3 Subsystem CAD Models and Engineering Drawings

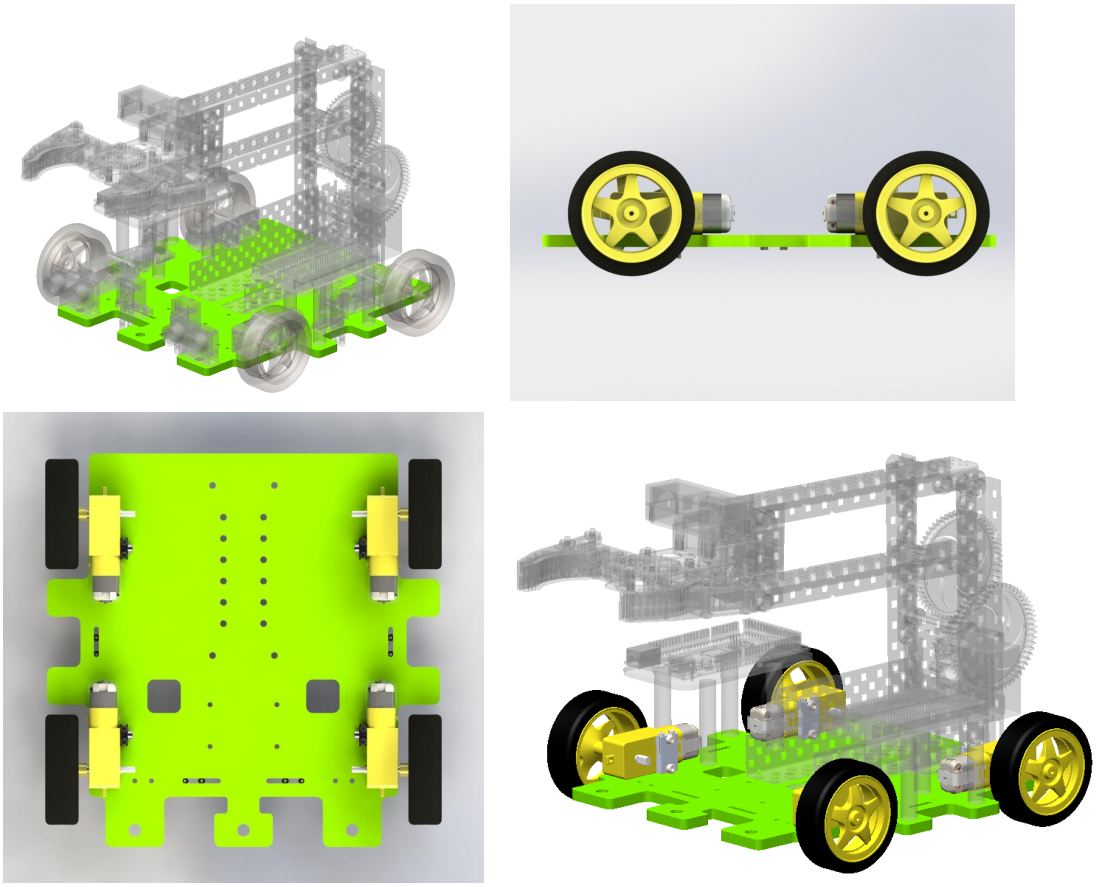


Figure 15: In order of appearance: isometric, top, side, and integrated views of the drivetrain assembly.

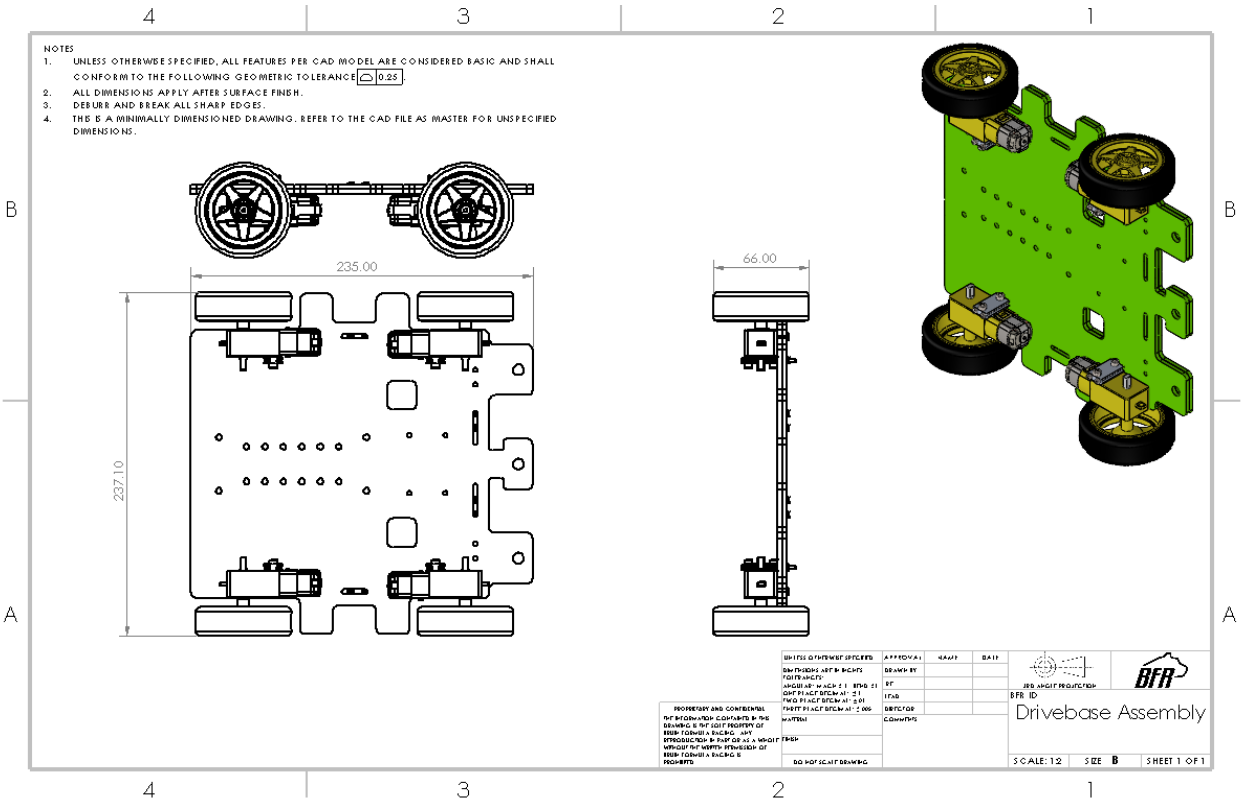


Figure 16: Engineering drawing of the drivetrain assembly.

3.3 Lift

3.3.1 Subsystem Description

The lift consists of a 4-bar linkage powered by 1x VEX robotics motor, geared down 25:1. We decided to use VEX steel construction channels as it allows for rapid iteration and ease of assembly. The ½ in grid system means we can integrate it easily into our chassis and design our own sensor mounts, gripper mounts, etc. It is also relatively lightweight and very rigid. Through our design concept development, we chose the 4-bar due to its simplicity and capability of reaching the design required height. Given the relatively small size of the robot, we needed something that could fit compactly within the bounding box and extend to reach the highest juicebox. It also extends longitudinally when it's horizontal, allowing the robot to grab the juicebox without needing to drive forwards as much.

3.3.2 Design Requirements

Table 8: Lift Subsystem Design Requirements

	Requirement	Reasoning
Linkage	4-bar	Simple, keeps object level, extends longitudinally to reach juicebox
Gear Train	2-stage 5x1 (25x1 total)	High gear ratio required to gripper assembly + full juicebox
Motors	1X DC VEX motor	Sufficient speed/torque requirements, integrates with VEX construction system
Construction	VEX steel channels/hardware	Rigid, adjustable

3.3.3 Subsystem CAD Models and Engineering Drawings

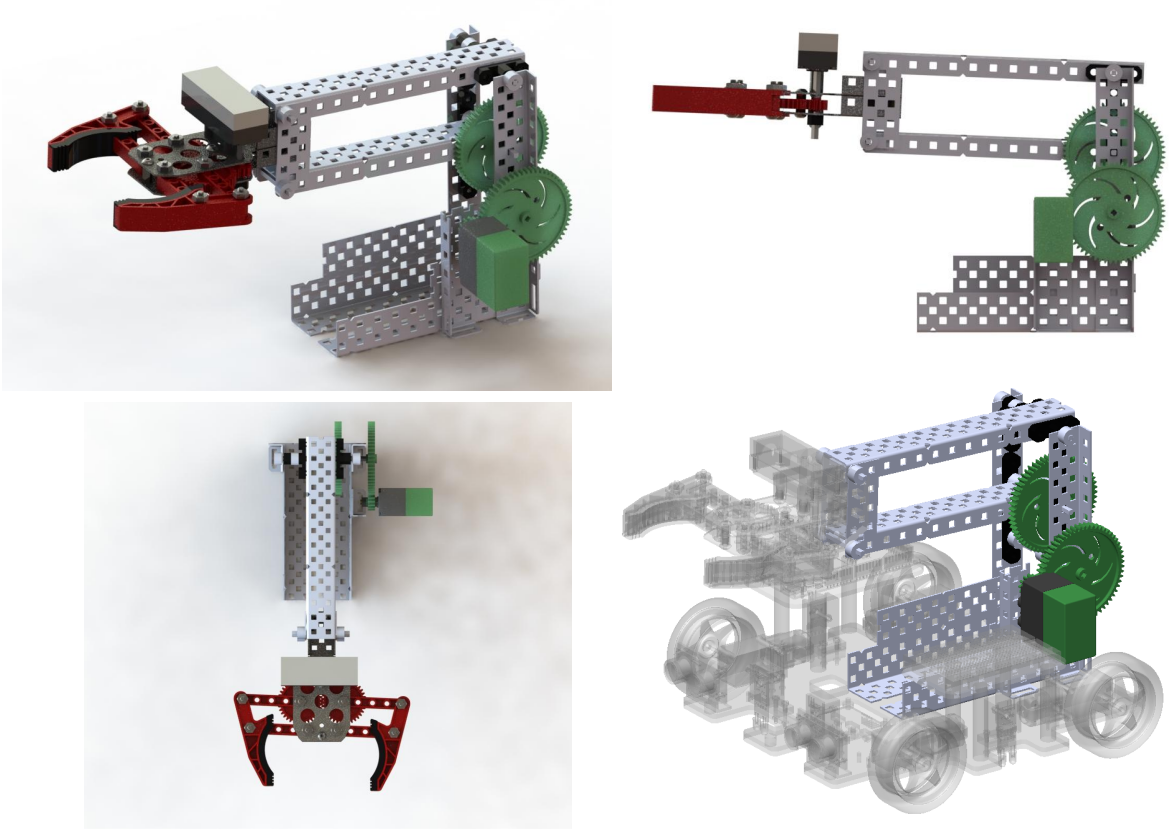


Figure 17: In order of appearance: isometric, top, side, and integrated views of the lift assembly.

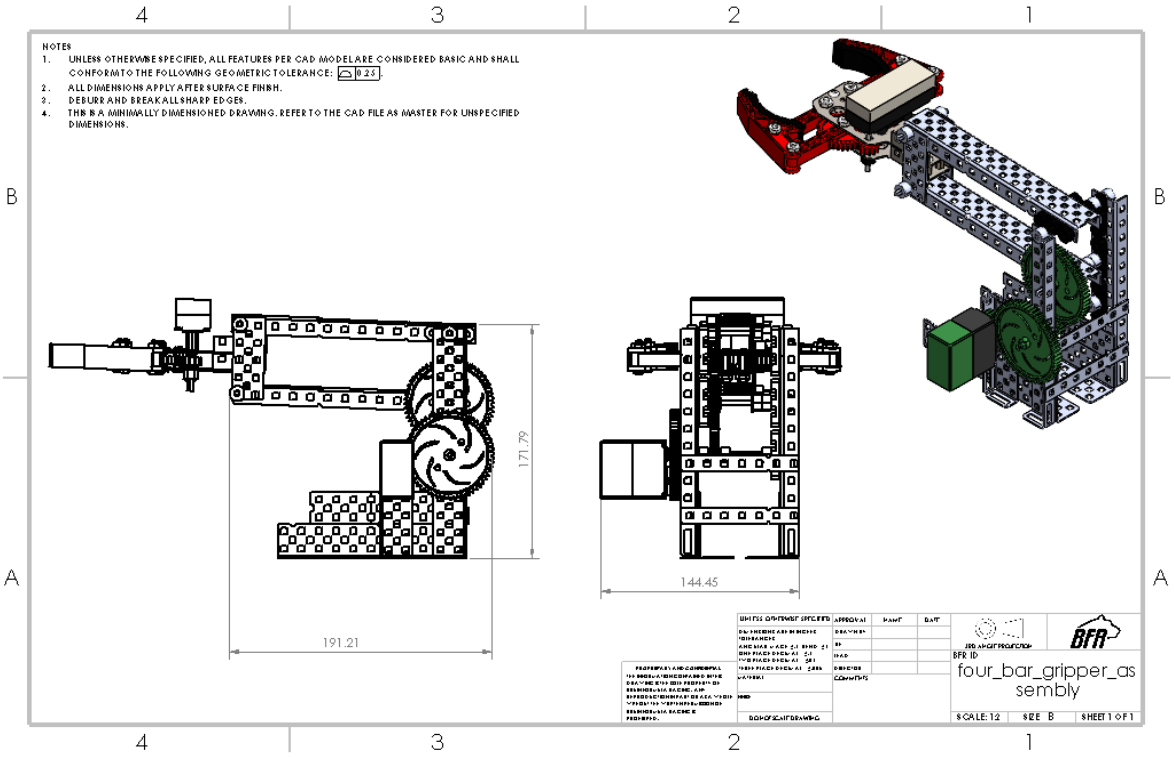


Figure 18: Engineering drawing of the lift assembly.

3.4 Grabber

3.4.1 Subsystem Description

The grabber is built off the VEX robotics clawbot kit. Since we are already using the kit for other systems on the robot, the claw is easily accessible and meets most of our requirements. To customize the claw to fit our needs, we swapped the “gripping” portions with custom-design grippers. The default grippers had a concave portion which didn’t conform to the juicebox well. Our custom gripper is completely flat, which fits well with the rectangular juicebox. In addition, we added adhesive grip tape to the claw so ensure the juicebox doesn’t slip out of its grip.

3.4.2 Design Requirements

Table 9: Grabber Subsystem Design Requirements

	Requirement	**Reasoning**
Linkage	3:1 geared 4-bar linkage	Interlinks both arms, keeps gripper faces parallel, extends/retracts
Motor	1X DC VEX motor	Sufficient speed/torque requirements, integrates

		with VEX construction system
Traction	Adhesive-backed rubber	Increases friction between gripper and juicebox

3.4.3 Subsystem CAD Models and Engineering Drawings

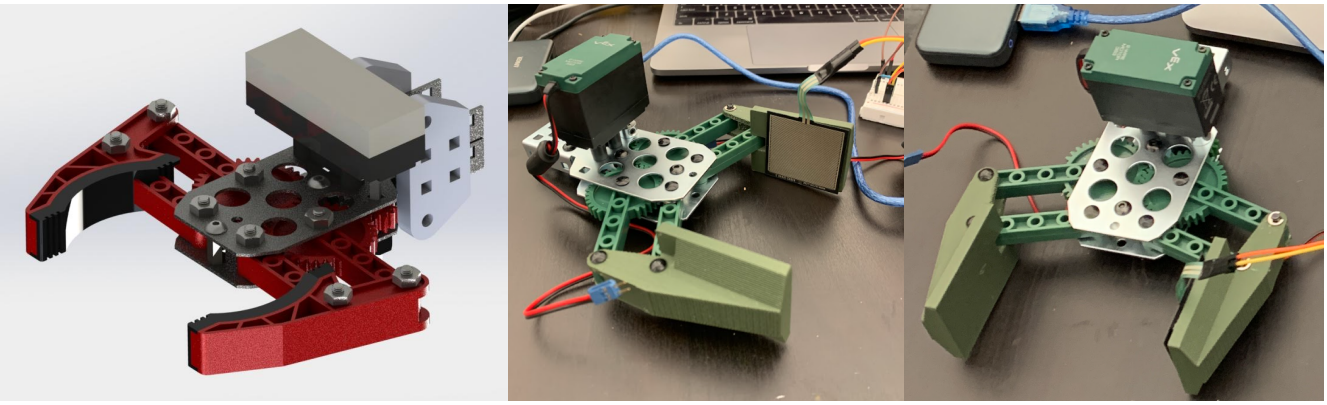


Figure 19: In order of appearance: isometric and real-life views of the grabber assembly.

3.5 Electronics

3.5.1 Subsystem Description

Our electronics system is built around the Arduino Mega. We're using the Mega (as opposed to a mini or uno) due to its increased number of digital/analog/PWM pins. We have the physical space in our robot and need the added I/O to integrate all our sensors. There's also the Arduino shield from the Elegoo kit bot attached, as it already comes with the correct connectors to control the 4X drive motors. For line tracking, we're using 4x individual IR sensor modules. 2x keep the robot on the straight lines and 2x detect junctions for pickup and dropoff locations. There are 2x frontward facing ultrasonic sensors to detect walls and the moving obstacle. We also experimented with a force-sensing resistor on the gripper faces and an encoder to output lift position. To improve wire management, we use a standard breadboard with standard 22 AWG wires and pins.

3.5.2 Design Requirements

Table 10: Electronics Subsystem Design Requirements

	Parameter	Reasoning
Computer	Arduino Mega w/ Elegoo smart shield	Increased I/O pins, shield directly connects to drive motors
Line-tracking	4X IR sensors (2X front, 2X side)	Keeps alignment on straight lines, detects junctions
Positioning	2X ultrasonic sensors (forwards facing)	Detects moving obstacle
Gripper feedback	Force-sensing resistor	Closed-loop feedback control for grip strength
Wiring	Breadboard, 22 AWG wires/pins	Expands number of pinouts from Arduino
Lift position	HC-020K Double Speed Encoder	Precise height positioning of the lift

3.5.3 Subsystem CAD Models and/or pictures

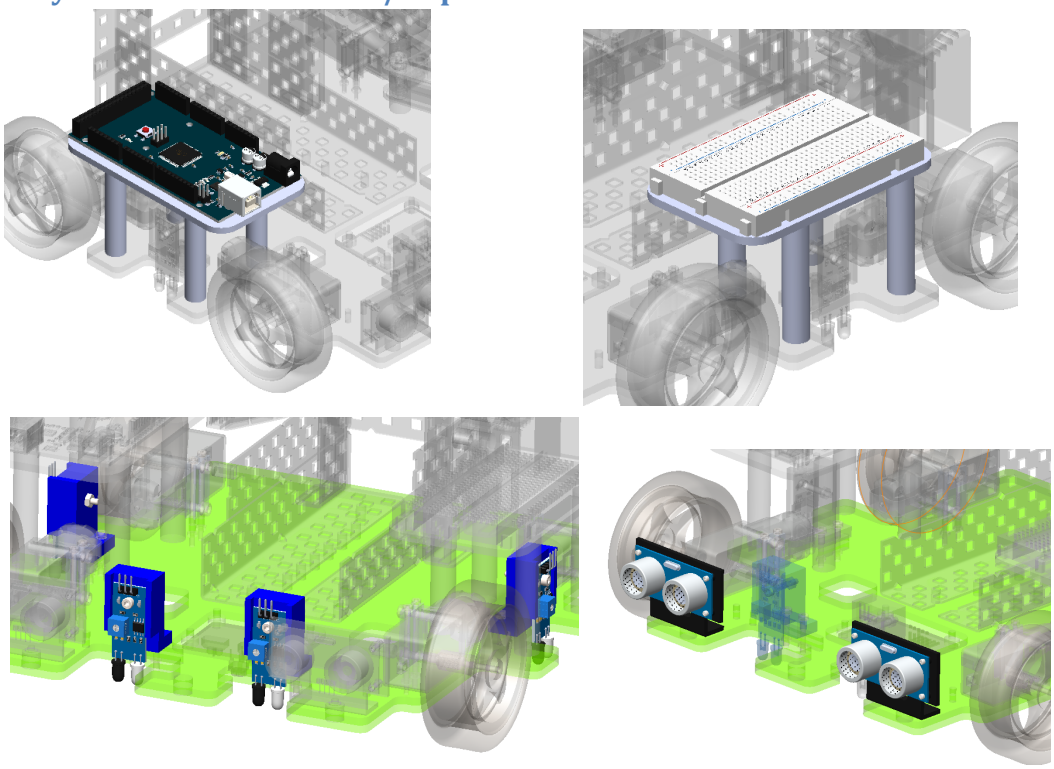


Figure 20: In order of appearance: Arduino Mega, breadboard, ultrasonic, and IR sensor integration.

4. Design Analysis

4.1 Analysis and Calculations

4.1.1 Move Profile

When planning out the path of our robot, we constructed a theoretical "move profile" for the challenge, consisting of each segment in the robot's trajectory. We also had a target of ~1 min for the overall challenge completion time. This drove our requirements for velocity and acceleration of the robot's drive system.

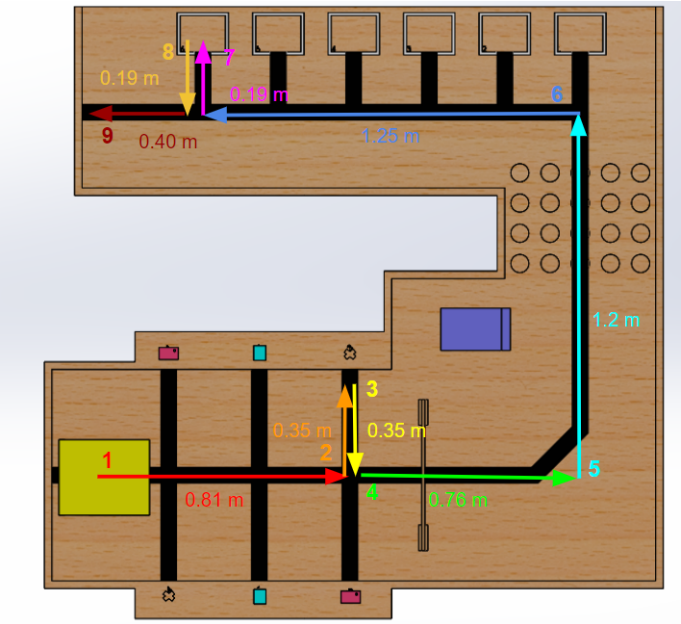


Figure 21: Planned move profile of robot.

Based on the move profile and our target complete time, we calculated target ramp-up, peak, and ramp-down velocities for each segment. Below is the analysis of each segment, plotted to show the velocity vs. time of the robot throughout the course.

V_{max} and a_{max} for each segment are calculated using the below estimates:

$$V_{max} = \frac{L_{seg}}{t_{tot}} \tag{1}$$

$$a_{max} = \frac{V_{max}}{t_{acc}} \tag{2}$$

Path Segment	L_seg (m)	t_tot (s)	V_max (m/s)	a_max (m/s^2)
1	0.81	6	0.135	0.135
2	0.35	3	0.1166666667	0.07777777778
3	0.35	3	0.1166666667	0.07777777778
4	0.76	5	0.152	0.152
5	1.2	7	0.1714285714	0.1714285714
6	1.25	7	0.1785714286	0.1785714286
7	0.19	2	0.095	0.095
8	0.19	3	0.095	0.06333333333
9	0.4	3	0.1127307558	0.08888888889

Figure 22: Maximum velocity and acceleration for each path segment.

Path Segment	Move Profile Type	Accel time t_acc (s)	Const. Vel. Time t_c (s)	Decel. Time t_dec (s)	Time Along Segment t_tot (s)
1	Trapezoidal	1	4	1	6
2	Triangular	1.5	0	1.5	3
3	Triangular	1.5	0	1.5	3
4	Trapezoidal	1	3	1	5
5	Trapezoidal	1	5	1	7
6	Trapezoidal	1	5	1	7
7	Triangular	1	0	1	2
8	Triangular	1.5	0	1.5	3
9	Triangular	1.5	0	1.5	3

Figure 23: Velocity and acceleration ramp for each path segment.

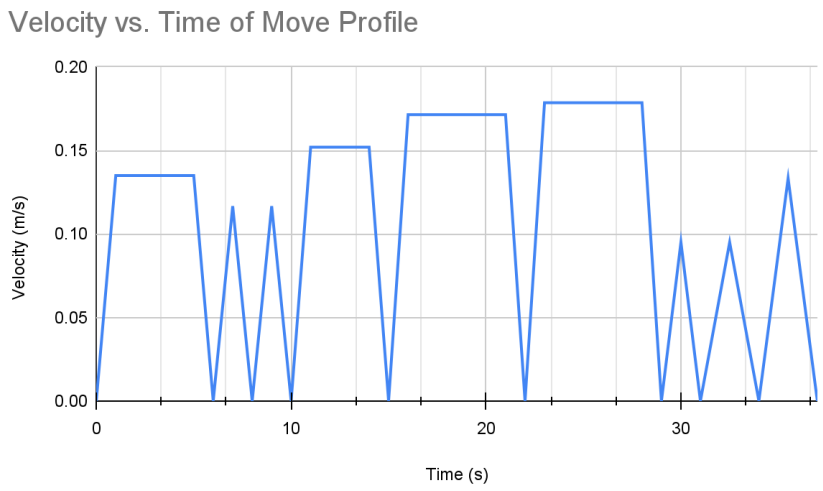


Figure 24: Velocity vs. time plot for each path segment.

4.1.2 Drivetrain Power Requirements

From the above kinematic requirements, we were able to derive force/power/torque requirements for the drivetrain system. This took into account the diameter of our wheels and the expected efficiency (friction, heat loss, rolling resistance, etc.) of the entire system. Below are our theoretical propulsive forces, power, and torque for each path segment.

To calculate the appropriate propulsion forces of the robot in each segment, we go back to Newton's Third Law:

$$F = ma. \tag{3}$$

More specifically, we need to sum all the forces the robot must overcome to propel itself:

$$F_{propulsion} = F_i + F_W + F_f + F_{rol} \tag{4}$$

where

$$F_i = m_{robot} a_{max} \tag{5}$$

$$F_W = m_{robot} g = 0 \text{ N} \tag{6}$$

$$F_f = \mu N_f \tag{7}$$

$$F_{rol} = 0 \text{ N}. \tag{8}$$

Note that $F_W = 0$ as there is no incline in the venue, and $F_{rol} = 0$ as we are using relatively stiff, non-pneumatic tires. This means our contact patch is negligible, and thus so if the force generated from rolling resistance.

Path Segment	F_i (N)	F_W (N)	F_f (N)	F_rol (N)	F_propulsion (N)
1	0.15795	0	0.9993984	0	1.1573484
2	0.091	0	0.9993984	0	1.0903984
3	0.091	0	0.9993984	0	1.0903984
4	0.17784	0	0.9993984	0	1.1772384
5	0.2005714286	0	0.9993984	0	1.199969829
6	0.2089285714	0	0.9993984	0	1.208326971
7	0.11115	0	0.9993984	0	1.1105484
8	0.0741	0	0.9993984	0	1.0734984
9	0.104	0	0.9993984	0	1.1033984

Figure 25: Summation of forces for each path segment.

To calculated the required motor power, we use

$$P_{prop} = F_{prop} \times V_{max}. \tag{9}$$

Path Segment	P_prop (W)
1	0.156242034
2	0.1272131467
3	0.1272131467
4	0.1789402368
5	0.2057091135
6	0.2157726735
7	0.105502098
8	0.101982348
9	0.1243869355

Figure 26: Theoretical required propulsive power for each path segment.

Lastly, to calculate the required propulsion torque, we can use the equation

$$T_{prop} = \left(F_{prop} \times \frac{\text{wheel diam.}}{2} \right) \frac{1}{\eta_{\text{drive-system efficiency}}}. \tag{10}$$

Path Segment	P_prop (W)	T_prop (N*m)
1	0.156242034	0.05456071029
2	0.1272131467	0.051404496
3	0.1272131467	0.051404496
4	0.1789402368	0.05549838171
5	0.2057091135	0.0565700062
6	0.2157726735	0.0569639858
7	0.105502098	0.05235442457
8	0.101982348	0.05060778171
9	0.1243869355	0.05201735314
Wheel Diam (m)	**Efficiency**	
0.066	0.7	

Figure 27: Theoretical required propulsive power and torque for each path segment.

4.1.3 Friction and Tractive Force Calculations

The material that our wheels will be made out of is rubber, and the coefficient of friction of 'hard rubber' on wood was found to be around 0.7 [6]. Calculating the required minimum coefficients of

friction using Equation () for the various drive mechanism types resulted in 0.1267 for FWD, 0.2825 for RWD, and 0.0875 AWD. Comparing the friction coefficients, AWD has the lowest coefficient of friction, whereas RWD has the highest, which suggests that RWD experiences the most friction.

$$\mu = \frac{L\sin\theta}{(\gamma_r - \gamma_f)(h\sin\theta - L_c\cos\theta) + \gamma_r L\cos\theta} \tag{11}$$

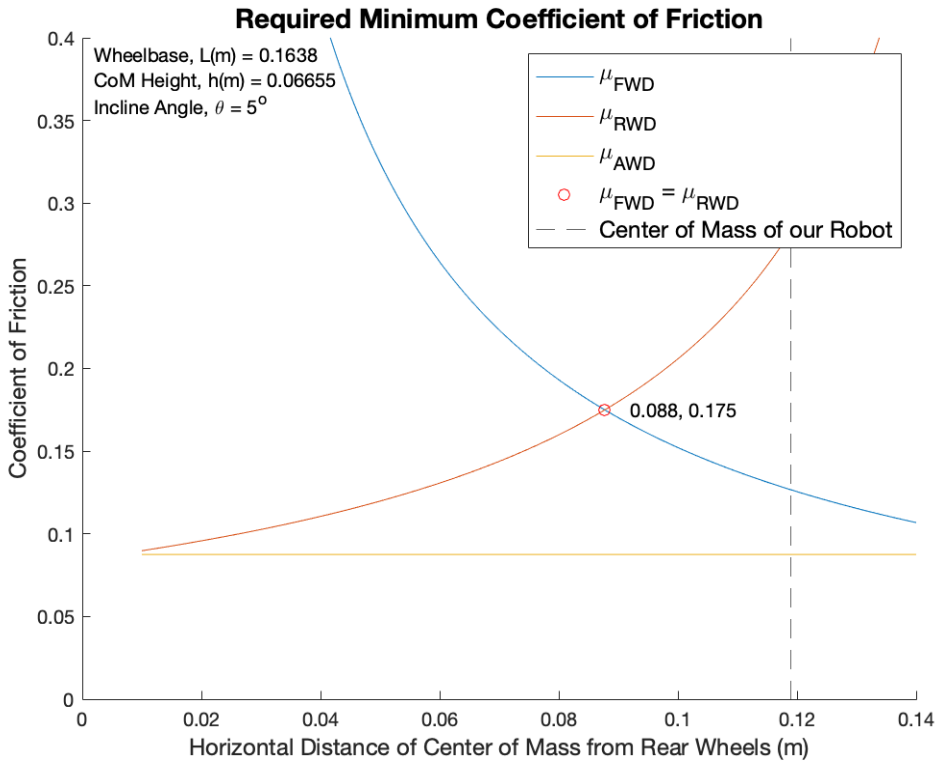


Figure 28: Coefficient of friction vs horizontal distance of center of mass from rear wheels for FWD, RWD, and AWD

In this figure we can see that AWD is better than RWD and FWD across the whole domain of center of masses shown. We can also see that at the center of mass for our robot FWD requires a lower coefficient of friction than RWD, so therefore FWD is better for our intents, which is what our robot uses.

The normal force between the rear wheels and the ground was calculated to be 4.625 N and the normal force between the front wheels and the ground was 10.312 N using the following equations:

$$F_{NR} = N_r = \frac{F_{mg}(L - L_c)\cos\theta + F_{mg}h\sin\theta}{L} \tag{12}$$

$$F_{NF} = N_f = \frac{F_{mg}L_c\cos\theta - F_{mg}h\sin\theta}{L} \tag{13}$$

Using the calculated normal forces and the coefficients of friction for the different drive types, the total tractive force was calculated for each one using Equation 4. The total tractive forces were

determined to be 1.8925 N for FWD, 4.2197 N for RWD, and 1.307 N for AWD. Since the only term changing between each is the coefficient of friction, it makes sense that the trend here aligns with that of the coefficient of friction. RWD has the greatest tractive force, whereas AWD has the least.

$$F_{TF} = \mu N_f \, , \, F_{TR} = \mu N_r \rightarrow F = F_{TF} + F_{TR} \tag{14}$$

The force of gravity acting along a slope of a maximum of 5 degrees was calculated to be 1.3068 N, according to Equation 5. Since all three drive mechanism types have tractive forces that are greater than the force of gravity, all of them would be able to drive the transporter up the slope. However, it is important to note that the tractive force for AWD is essentially the same as the force of gravity along the slope, which is barely enough to make it up the slope.

$$F_{//} = mg \sin\theta \tag{15}$$

Using Equations 6 and 7, the weight distribution between the front and rear wheels was determined to be 0.6904 and 0.3096, respectively. This suggests that just over two-thirds of the total weight distribution is at the front wheels, and just under one-third is at the rear wheels.

$$\beta_f = \frac{N_f}{F_m \cos\theta} \tag{16}$$

$$\beta_r = \frac{N_r}{F_m \cos\theta} \tag{17}$$

4.1.4 Motor Torque Requirements

From the tractive force and wheel diameter, the minimum required wheel motor torque for the three different drive systems we considered were calculated. We found that the front wheel drive torque was the least, followed by the all wheel drive torque, and lastly the rear wheel drive torque. The minimum RWD motor torque required was more than twice that of the FWD motor torque. Even though it was not the least, we ultimately chose AWD for our drive system, and the minimum motor torque required was 0.0863 N-m. The exact results from our motor torque calculations are shown below.

Minimum Required Wheel Torque Based on Tractive Force				
Front Wheel Diameter	**D_fw**	**Diam_fw**	**0.066**	m
Rear Wheel Diameter	**D_rw**	**Diam_rw**	**0.066**	m
Front Wheel Torque	**T_Fw**	**T_Fw**	**0.0625**	N-m
Rear Wheel Torque	**T_Rw**	**T_Rw**	**0.1393**	N-m
All Wheel Torque	**T_aw**	**T_Aw**	**0.0863**	N-m

Figure 29: Theoretical minimum required motor torque calculations for the three different drive systems.

4.1.4 Juice Box Retrieval Calculations

In order for the robot to pick up the juice box, it needs to have a high enough clamping force so that the juice box does not slip out of the gripper, and it needs to have enough torque to raise and lower the four bar mechanism carefully. Please refer to the appendix for the entire hand calculation. For the first part, the required clamping force was calculated by first making the following free body diagram.

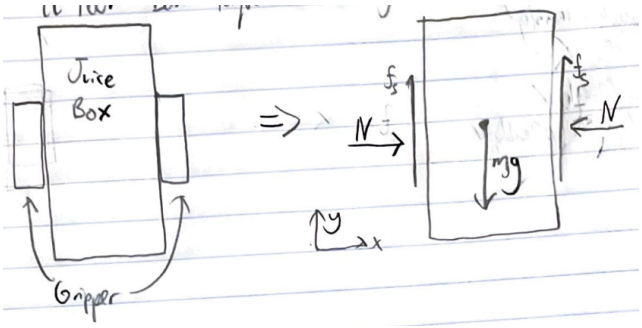


Figure 30: Free body diagram to calculate required gripper clamping force.

From here, the force balance in the y-direction was calculated and then solved for the normal force.

$$\sum F_y : 2f_s - m_j g = 0, \quad f_s = \mu_s N \tag{18}$$

$$N = \frac{m_j g}{2\mu_s} = \frac{(0.3kg)(9.8m/s^2)}{(2)(.2)} = 7.35N \tag{19}$$

In these equations μ_s is the coefficient of static friction between the juice box and the gripper. It was conservatively chosen to be 0.2 because the exterior of the juice box is polyethylene (PE), and the coefficient of static friction for PE on PE is 0.2. As this was a static analysis, the implicit assumption was made that the four bar lift will accelerate slowly enough that it does not play a significant role in the forces exerted on the juice box. We are planning to have the four bar mechanism move slowly and carefully, so this assumption is justified.

For the second part the torque required was calculated by first making the following free body diagram.

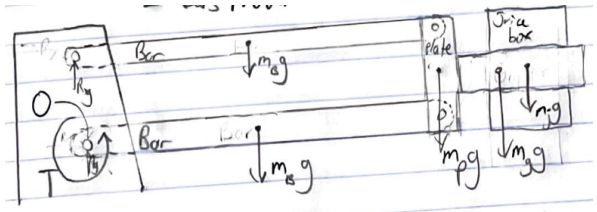


Figure 31: Free body diagram to calculate required four bar torque.

As seen in the free body diagram, this calculation was chosen to be completed while the four bar mechanism was at its most horizontal position. This is because this is the position where maximal torque is required, because the distance from the point where the torque is applied (O) to the line

of action of each of the gravitational forces is at its greatest in this position. From here the torque balance around the point of rotation (labeled O) was calculated, and then solved for the torque T.

$$\sum M_O : T - 4m_B r_B g - m_p r_p g - m_g r_g g - m_j r_j g = 0 \quad (20)$$

$$T = (4m_B r_B + m_p r_p + m_g r_g + m_j r_j)g \quad (21)$$

$$T = (4(0.0038kg)(0.09m)+(0.002kg)(0.19m)+(0.11kg)(0.22m)+(0.3kg)(0.25m))(9.8m/s^2) \quad (22)$$

$$T = 0.989N - m \quad (23)$$

Since it is a static analysis, this is the torque required to hold the juice box in this position, but a greater torque is required to lift the juice box. In these equations m_j, the mass of the juice box, was taken as 0.3 kg because that was given as the maximum possible weight of the juice box. The rest of the masses and center of gravity distances were calculated from the CAD model of the robot. The static analysis assumption was once again made in this calculation, for the same reasons stated in the previous calculation. There is also the assumption that there is no friction between the axles and the mount. We will attempt to minimize the effects of friction at these locations, and if the friction is too great, we will use a bearing to minimize it further. Therefore, this is a reasonable assumption to make in this analysis, but it is worth noting that the torque required is slightly greater than the one calculated.

Since the robot will be holding the juice box in the gripper for the duration of the course, it does not require appreciable force to unload the juice box. The gripper simply needs to be opened enough for the weight of the juice box to overcome the friction of the gripper and land in the unloading zone.

The gripper's range of motion is estimated to be from a maximum opening size of 100 mm to being completely closed with a minimum opening of 0 mm, which was determined from the CAD model of the gripper. This is under the assumption that the gripper uses a motor that is capable of continuous rotation. If a motor is used that only allows for 180° of rotation then the difference between maximum and minimum opening would be $\pi D/2$, where D is the diameter of the gear attached to the motor. The four bar mechanism is estimated to be able to go from an angle of 70° above horizontal, corresponding to a gripper height of 300 mm off the ground, down to an angle of 30° below horizontal, corresponding to a gripper height of 40 mm off the ground due to interference with other parts of the robot. However the upper limit is realistically constrained by the fact that the gripper cannot grab a juicebox sitting on top of a wall if the gripper is behind the front of the robot that is up against the wall. With this constraint the max height of the gripper is about 180mm corresponding to a four bar angle of 15° above horizontal.

5. Control System Design

5.1 Drive Motor Selection and Motor Specifications

5.1.1 Motor Selection

We chose to use the included Elegoo kit bot drive motor modules for our robot. These provide sufficient torque/speed for our requirements and are simple to implement with the rest of our electronics. The specifications of these motors is below:

Table 11: Drive Motor Specifications

Drive Motor Specifications (quantity = 4)	
Vendor	Adafruit
Weight (g)	30.6
Dimensions (mm)	70 x 22 x 18
Type	Brushed DC
Rated voltage (V)	3~6
Stall torque @ 6V (kg*cm)	0.8
Free speed @ 6V (RPM)	200+/- 10%
Free current (mA)	150 +/- 10%
Gear Ratio	1:48

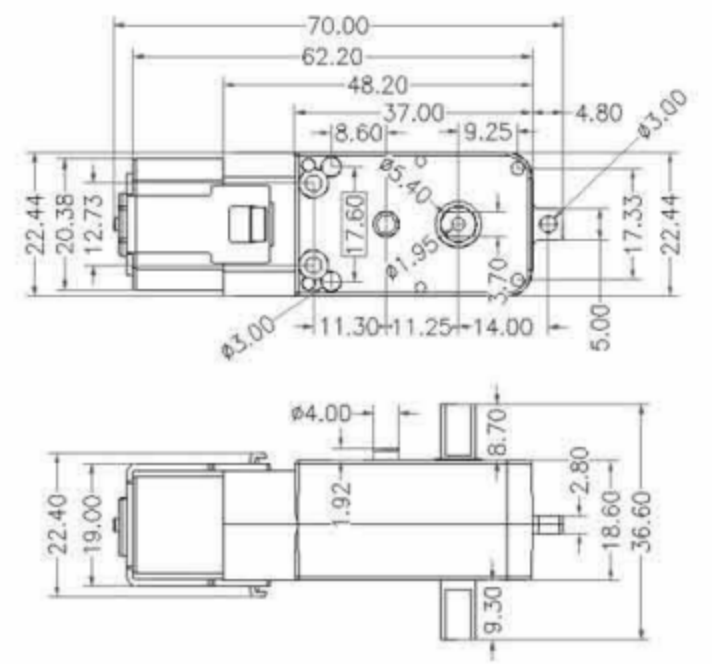


Figure 32: Schematic of Adafruit drive motor module

5.1.2 Circuit Wiring Diagram

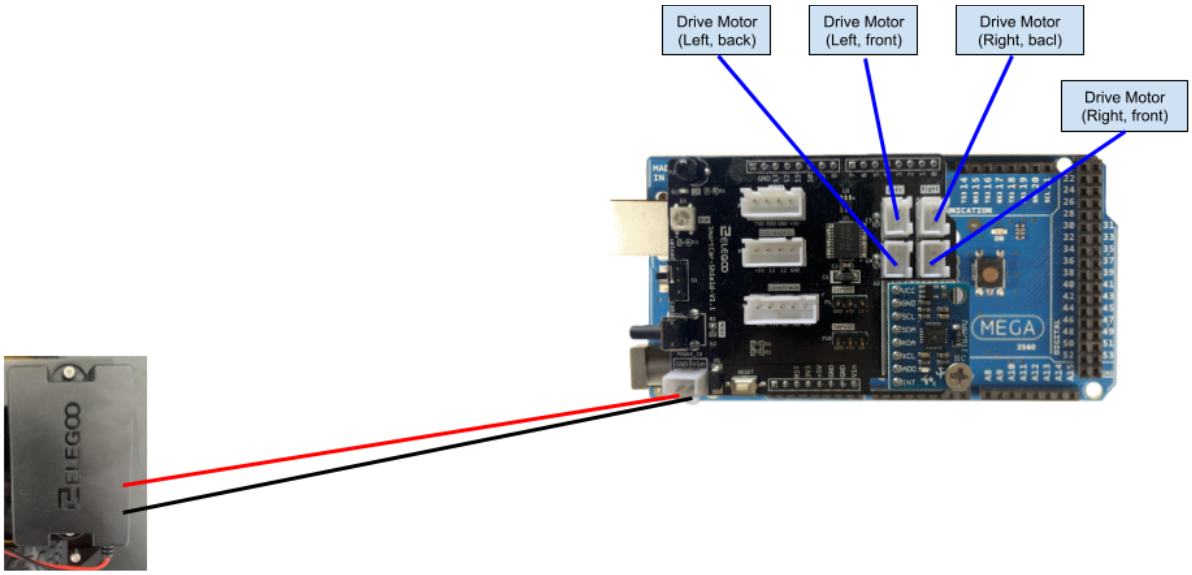


Figure 33: Drive Motor Circuit diagram

5.2 Juice Box Retrieval/Delivery Mechanism

5.2.1 Motor Selection

For the lift and grabber, we use the VEX robotics DC motors as they integrate directly with the rest of the VEX pre-made claw and steel structural channels. One motor is used for the lift and another motor is used for the gripper claw. The motors provide sufficient torque/speed for our requirements. To control the motors, we use the VEX robotics 2-pin motor controllers, which convert the 3-wire PWM signal from the Arduino to a 2-phase DC signal that the motor can understand.

Table 12: Lift and Grabber Subsystem Design Requirements

Lift and Grabber Motor Specifications	
Vendor	VEX Robotics
Type	Brushed DC
Rated voltage (V)	7.2
Stall torque (N*m)	1.67
Free speed (RPM)	100
Stall current (A)	4.8
Free current (A)	0.37

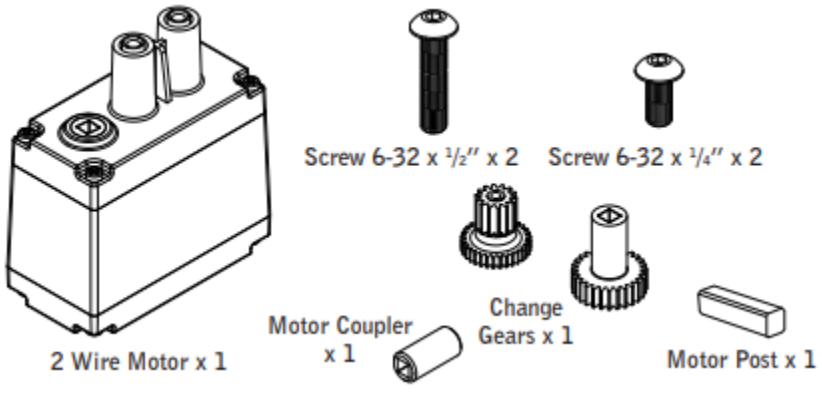


Figure 34: Diagram of lift and grabber VEX Robotics motor

5.2.3 Circuit Diagram

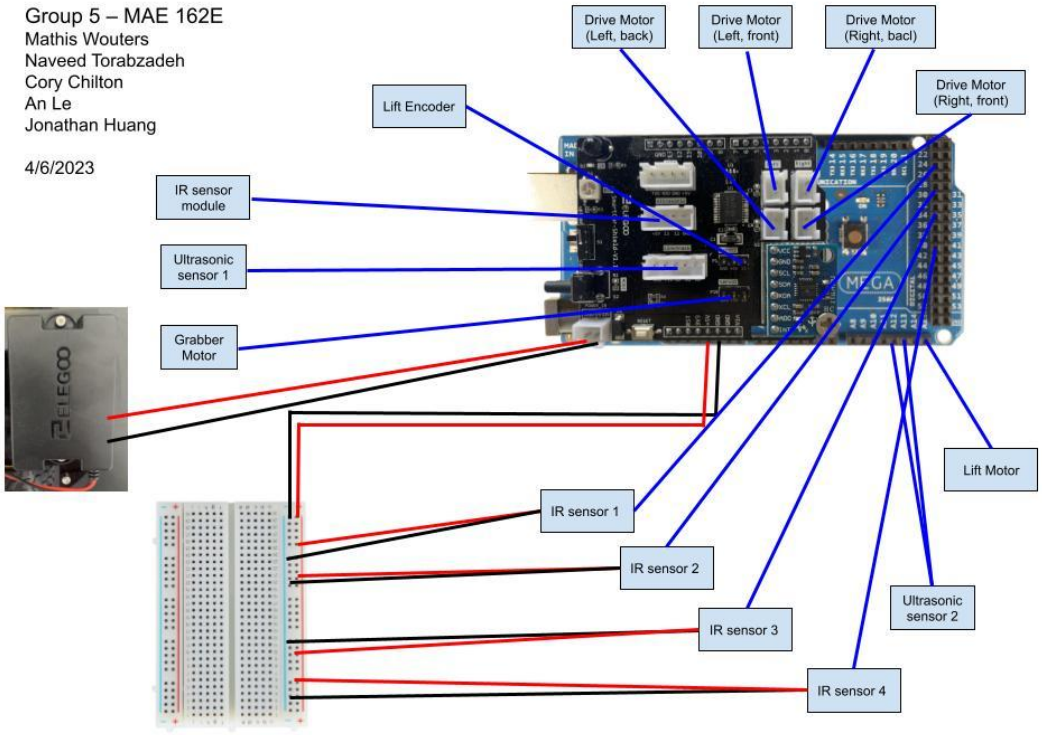


Figure 35: Circuit diagram

5.3 Sensors and Theory of Operation

5.3.1 Ultrasonic Sensors

The robot makes use of two front mounted ultrasonic sensors in order to detect walls and obstacles. The choice was made to use two of them in order to verify that the robot is perpendicular to the wall. This is done by continually comparing the distance detected by each sensor.

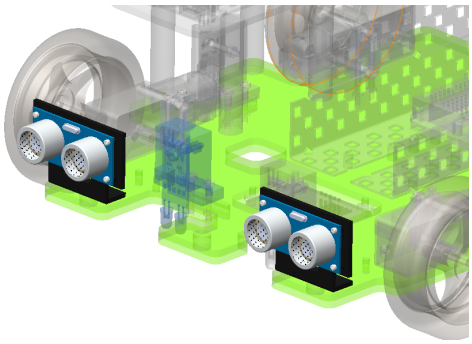


Figure 36: Front mounted ultrasonic sensors

Each ultrasonic sensor has its own distance detecting function (detectDistanceR and detectDistanceL). These functions are essentially the same except that they read in values from the corresponding sensor.

5.3.2 Line-following IR sensors

For our design we chose to use 4 IR sensors in order to accomplish line following and branch detection.

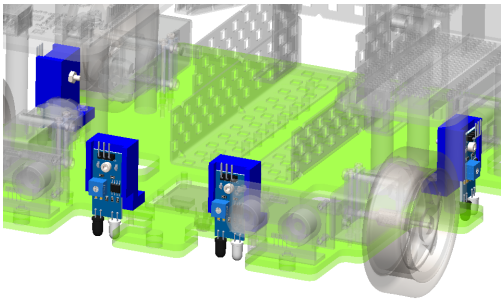


Figure 37: IR sensors used for line following and branch detection

The two front IR sensors are used primarily for line following and are physically calibrated and positioned to sit just along the edge of the two-inch-wide black line that has been laid on the floor of the Venue. When it is detected that one of the sensors has gone off the black line, the robot, per its code described in this section, makes a correction to keep the robot on the line.

The two side IR sensors are used for branch detection and branch selection. Using the read-in from these sensors, the robot is able to tell how many branches it has driven past so that it can pick up and drop off the item at the correct locations.

5.3.3 Limit Switches

In our design we do not use limit switches, however we use sensors that achieve the same function as a limit switch, which are used to control the gripper arm.

The lift uses an encoder, which provides precise movement data for the vertical distance traveled by the arm. The encoder disk is mounted to the gear train for the gripper arm, and encodes the rotation of the gear axle, converting it into vertical lift distance. In this way, we can determine when to stop the vertical travel of the lift.

The gripper uses a force resistive sensor, which measures the force that the gripper is inputting on the sides of the item. The sensor itself was calibrated using provided manufacturer specifications, and the system program was calibrated using a juice box to see the amount of force necessary to lift and carry the juice box without squeezing water through the straw. In this way, we know when to stop the motor when gripping the claw, and we can tell if we need to re-grip the item if it gets loose.

5.3.4 Circuit Diagram of all Sensors

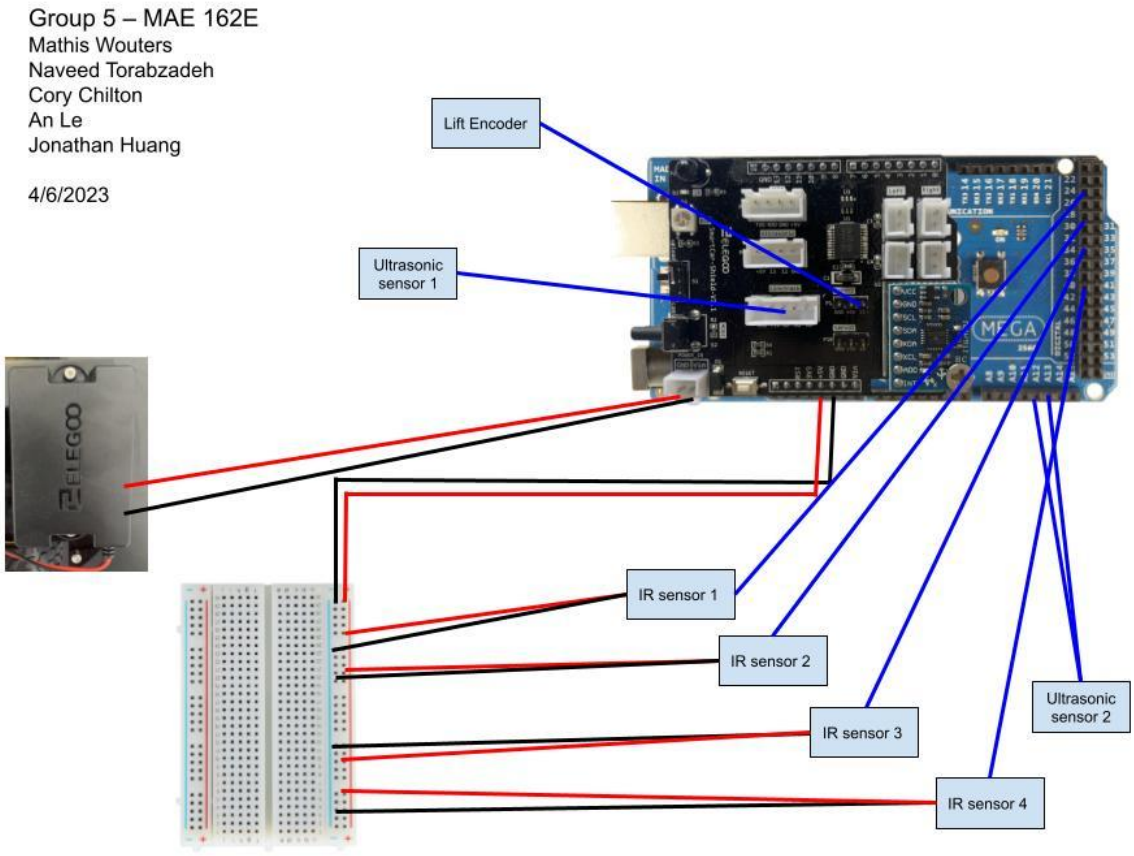


Figure 38: Circuit Diagram of all Sensors

5.4 State Diagram

5.4.1 Stateflow Chart

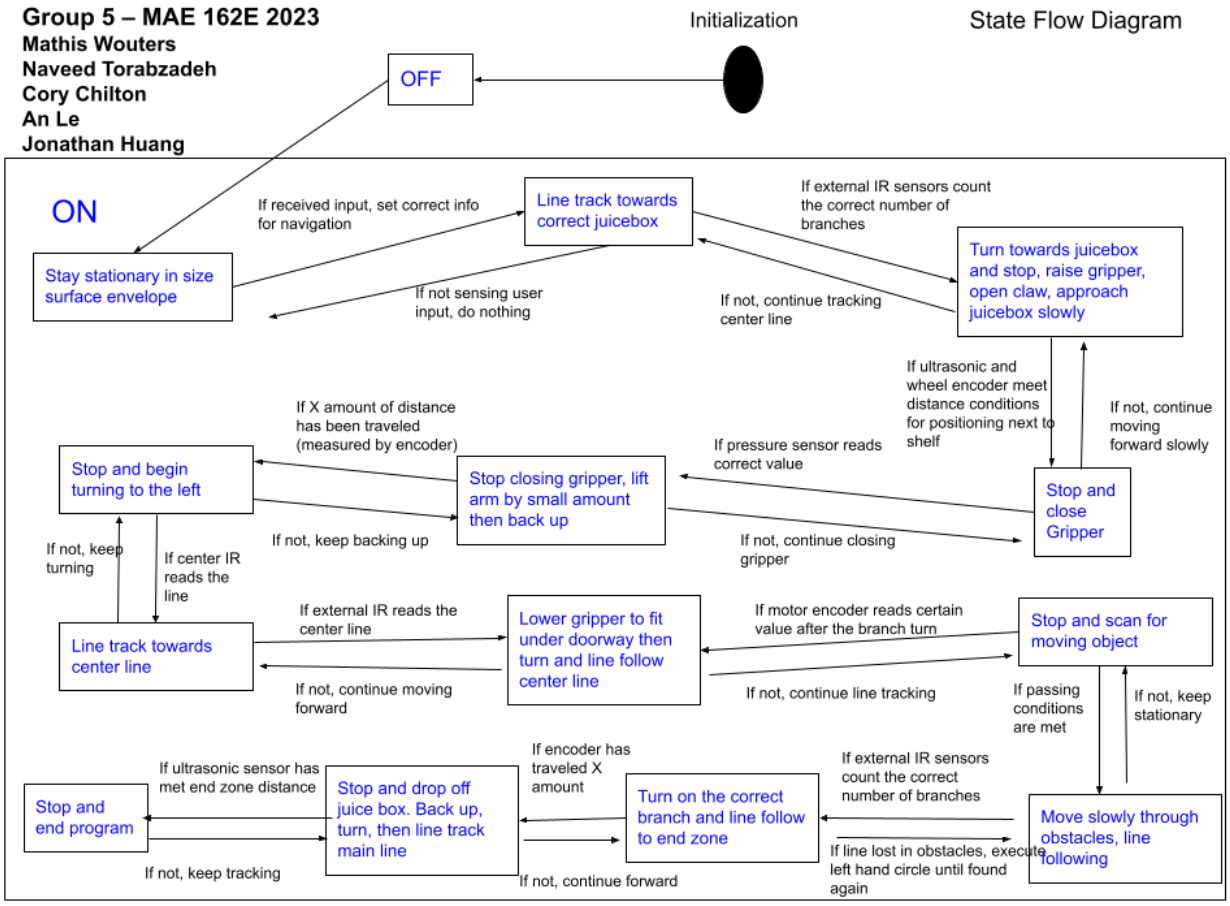


Figure 39: Stateflow Chart for autonomous control of the robot

5.4.2 Code

5.4.2.1 Overall Code Structure and Stage Flow

The full arduino code is appended. Here we will discuss the code structure and stage flow. The main code loops through three sections of code. It first runs through a program shutoff detection function, which acts as a fault mechanism. It detects input from the IR remote and stops all robot functions in case of emergency. It then loops through the item location information function. This detects input from the IR remote for the pickup and dropoff of the item. This only occurs once at the beginning of the program. Once this is confirmed, the program will begin stage 1. The code then loops through 11 discrete stages in a series programming structure. Each stage loops until it is completed, at which point it starts the next stage and turns itself off. The program terminates at the end of the last stage.

Here we will describe each of the program stages.
Stage 1: Line follow until the correct branch is achieved, incrementing the counter and comparing with the branch number as necessary

Stage 2: Turn to the correct direction and raise the gripper to the correct height

Stage 3: Line follow until the ultrasonic sensors sense the stopping distance, then close the gripper claw and raise by small amount

Stage 4: Back up slowly until the side IR sensors detect the main line then turn 90°

Stage 5: Follow the main line, continuing to increment the counter as necessary. If the robot is at the last set of branches, this step is ignored and stage 6 is activated instead

Stage 6: Once the last branch is detected, follow the main line for a certain amount of time then raise the gripper. Move forward slowly and check the ultrasonic sensors for the moving obstacle

> Stage 6b: Activated if the moving obstacle was detected. Stop and wait for the obstacle to move while lowering gripper then activate stage 6c
>
> Stage 6c: Follow the main line until the second set of branches is reached. If the first branch is the correct one, skip straight to stage 8. If the correct branch is not the first one turn left then go to stage 7

Stage 7: Follow main line until the correct branch is achieved, then turn right

Stage 8: Line follow the branch then drop off the item

Stage 9: Reverse until the side IR sensors detect the main line then turn left 90°

Stage 10: Line follow until the ultrasonic sensors detect the final stopping distance

Stage 11: Stop and terminate the program

5.4.2.2 Main Functions

Here we will discuss some of the main functions used to run the program.

Obstacle detection function

The obstacle detection functions use the two front mounted IR sensors. The sensor essentially outputs a pulse from the trig pin and records the return duration with the echo pin. The function returns the distance, which is calculated according to the duration and the speed of sound. This

distance is compared to the opposite sensor and a preset stopping distance.

```
float detectDistanceR(){

  digitalWrite(trigPinR, LOW);
  delayMicroseconds(2);
  digitalWrite(trigPinR, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPinR, LOW);

  durationR = pulseIn(echoPinR, HIGH);
  distanceR = (durationR*.0343)/2;

  Serial.print("Distance: ");
  Serial.println(distanceR);
  delay(0.05);

  return distanceR;
}
```

Figure 40: detectDistanceR function used to return the distance sensed by the right ultrasonic sensor

Line Following

Line following function makes use of the two front-mounted IR sensors, spaced exactly on either side of the venue's black line. Slots in the chassis plate allow for lateral micro-adjustment, meaning physical calibration rather than programming. If neither of the sensors detect the black line, the robot goes forward (fig. a). If one of the two light sensors detect the black line, the robot will turn away from the line (fig. b and c).

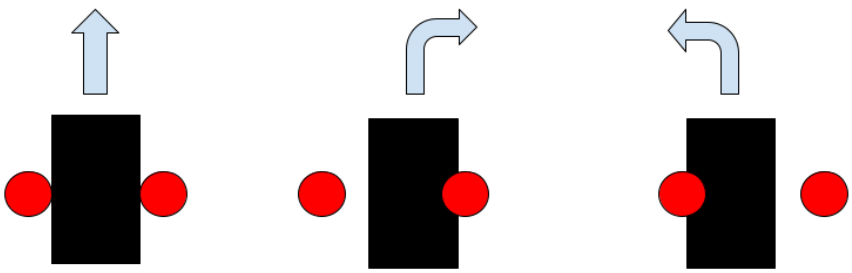


Figure 41: In order of appearance from left to right: fig a, fig b, fig c

Branch Detection: Pickup Branches

The IR remote will initialize two values: the branch number for the pickup branches and the side (left or right) that the robot will turn to for pickup. To detect a branch, the robot will move slowly if both of the front IR sensors detect black. Then the robot will stop once the two side mounted sensors detect black. This centers the robot directly on the branch intersection. Once a branch is detected, the counter is incremented. If the counter is equal to the branch number, the robot will turn to the correct side. Otherwise, the robot will continue line following until it hits the next branch. Figure a) depicts where front sensors detect a branch. The robot then moves forward slowly. Figure b) depicts where the side IR sensors detect the branch. The robot then stops.

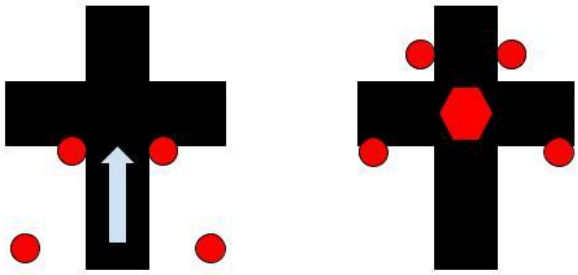


Figure 42: In order of appearance from right to left: figure a) and figure b)

Branch Detection: Pickup Branches

The IR remote will initialize two values: the branch number for the pickup branches and the side (left or The IR remote will initialize the branch number for the drop-off location. Similar to the pickup branches, a counter is incremented every time a branch is detected. If the branch is correct, the robot will turn and execute the drop off. The same set of front IR sensors are used, however only the side IR on the right is used for branch center detection. Figure a) depicts where front sensors detect a branch. The robot then moves forward slowly. Figure b) depicts where the side IR sensors detect the branch. The robot then stops.

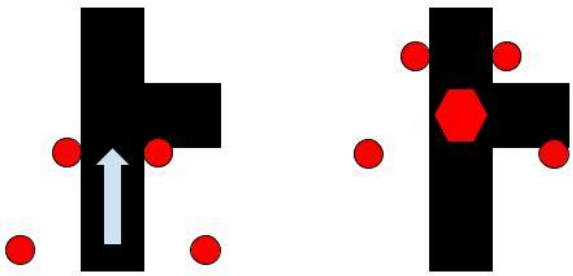


Figure 43: In order of In order of appearance from right to left: figure a) and figure b)

Movement Functions

The movement functions are called by the main code to move the robot. Each function sends the correct speed and direction values to the left and right motors. Depicted below is the forward() function, which can be called to move the robot forward at speed1.

```
void forward(){
    digitalWrite(motorsEnable, HIGH);
    analogWrite(leftWheels, speed1);
    analogWrite(rightWheels, speed1);
    digitalWrite(leftWheelsDir, HIGH);
    digitalWrite(rightWheelsDir, HIGH);

}
```

Figure 44: forward() function used to move the robot forward

6. Product Fabrication

6.1 Chassis

The chassis consists of two stacked acrylic plates fabricated by laser cutting. Originally, the material selected for the chassis was plywood, as we could easily drill holes into it as needed, however we later switched to 0.125” thick acrylic. Acrylic was chosen for its durability and rigidity, as well as how easily obtainable it was.

The chassis was initially designed as a single base plate and was fabricated as such, but we eventually decided to adjust to a two-plate design for additional structural rigidity to account for the relative thinness of the acrylic. We did not have access to any other thickness of acrylic, so using two plates was the next best option. Although most of the mounting holes were built into the CAD from the start, the holes for the four bar linkage specifically were not since it was custom-built and

precise measurements were harder to obtain. In an early version of the chassis, these holes were drilled into the plate after laser cutting the acrylic plate, which proved to be quite the hassle.

Paired with the revisions of mounting locations after testing components, these were the primary setbacks that were faced regarding the chassis. For the final version of the chassis, we ensured that every single mounting hole was accounted for in the CAD so that everything could be laser cut directly into the plate. Measurements were taken for the four bar linkage as a result, and mounting locations were revised as needed throughout testing. Making these adjustments ensured everything could be easily laser cut in the end and reduced the amount of additional machining required. The final design of the chassis plates can be seen below.

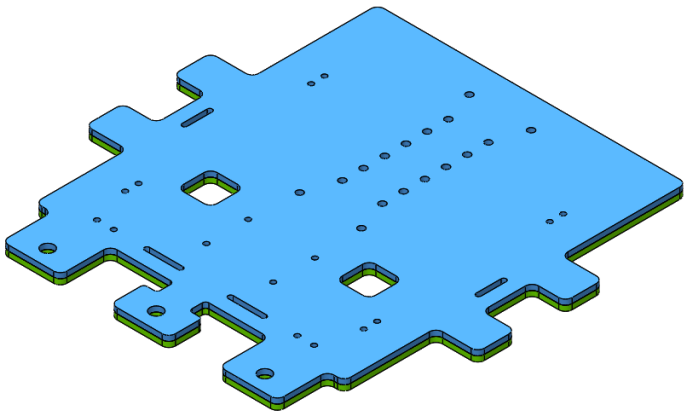


Figure 45: Stacked chassis base plate design.

6.2 Drive System

The drive system consists of the DC gearbox TT drive motors and wheels from the Elegoo kit robot. We initially opted for using FWD with just two drive motors in the front wheels and two free-spinning caster wheels in the rear. The caster wheels were to be mounted to the chassis by a 3D-printed component designed to be the proper height such that the caster wheels would sit on the ground level with the front wheels. The 3D-printed caster wheel mount was also designed to use small rubber bands to aid in controlling the movement of the rear wheels. The drivebase utilizing this initial design can be seen in the figure below.

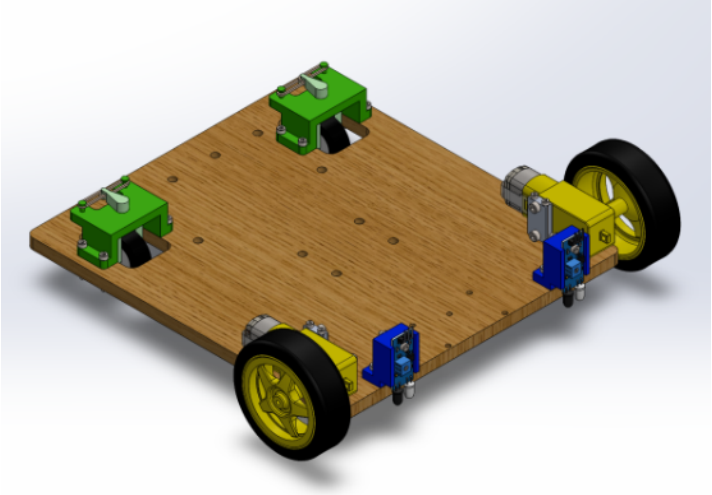


Figure 46: Initial FWD drive base assembly design with rear caster wheels.

However, this design led to some unforeseen problems. Upon 3D printing the caster wheel mounts, it was found that the prints were much smaller in hand than initially imagined and, as a result, were not structurally sound. The thinner parts of the mount were particularly dainty and fragile, breaking off easily after handling. Since there was not enough space on the chassis to increase the size of the caster wheel mounts by much, this idea was determined to not be feasible for our needs.

We then decided to skip the 3D-printing altogether and purchase caster wheels that were attached to a plate that had mounting holes which we could use to directly bolt onto the chassis. However, since we again ran into the issue of needing to make sure the bottom of the caster wheels sat flush with the bottom of the front drive wheels, another mount of some kind was inevitable. We laser cut two small acrylic plates, one for each rear wheel, and 3D printed four spacers for each plate. The caster wheels were then bolted onto each plate, with the spacers designed to be tall enough to ensure all four wheels were level.

However, after testing our robot on the course, we found that the front wheels tended to spin on uneven surfaces (due to the slippery and uneven track) and the caster wheels would lock up. To counteract these problems, we ultimately switched to AWD and powered the rear wheels with the same drive motors as the front. In terms of fabrication, this meant that the mounting plate, spacers, and caster wheel holes in the rear of the chassis plate were removed. Instead, the mounting of the drive motors in the front of the chassis was mirrored in the back for symmetry. The final drive base assembly that we settled on is shown below.

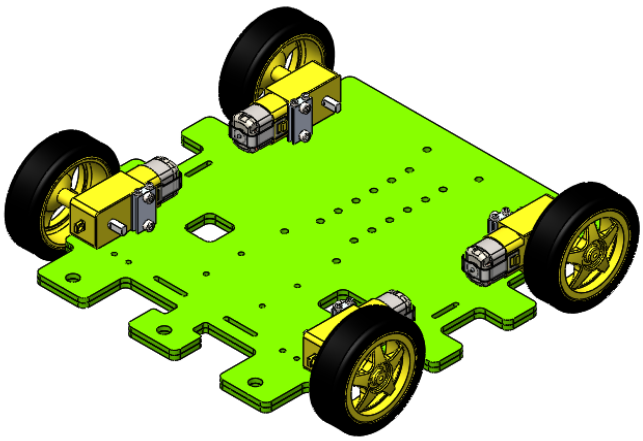


Figure 47: Finalized AWD drive base assembly with four identical motors and wheels.

6.3 Steering System

Regarding fabrication of and for the steering system, there were not many components that needed to be made. Firstly, mounts were designed and 3D printed in order to attach the IR sensors to the chassis plates. A hole was built into the mount for the IR sensor module to bolt onto, and additional holes were added to the 3D-printed mount to bolt onto the chassis. The first rendition of this print had a vertical section that was not tall enough, and since the back of the IR sensor module was not flat, it failed to sit vertically when attached to the mount. The mount was then redesigned with that part lengthened such that the IR sensor would be perpendicular to the ground when attached. The finalized IR sensor mount is depicted in the figure below.

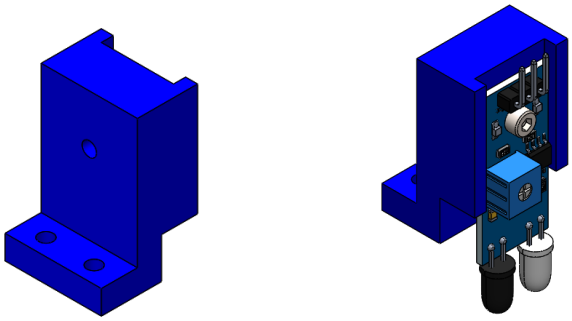


Figure 48: 3D-printed IR sensor mount (left) and IR sensor mount subassembly (right).

With regard to the IR sensors, we previously placed four of them in a line at the front of the chassis for line tracking. However, after testing out the line tracking and code, we decided to move two of the middle IR sensors to the sides of the chassis, with one on the left and another on the right. Using the side-mounted IR sensors vastly improved the line following and branch detection of the robot. This change only required updating a few mounting locations on the chassis plate.

In order to meet the size requirements of the robot, the Arduino and breadboard could not be mounted directly on the chassis plate due to the limited space left after mounting the other

components. Thus, we decided to make use of the vertical space that was available, since our robot was well within the size envelope vertically. This was accomplished by repurposing the mounting plates and spacers previously designed for the rear caster wheels that were scrubbed.

Instead of attaching the caster wheels to the bottom of the plate, we would now rest the Arduino Mega and breadboard on top of each plate. However, in order to avoid interference with the drive motors and side IR sensors, the height of the spacers were lengthened such that they would hover above the other components. Since the spacers were fairly tall and we did not have any bolts long enough to attach the plates to the chassis, we opted to glue the 3D-printed spacers to the chassis on one end and to the acrylic mounting plates on the other. Images of the two subassemblies of the Arduino Mega and breadboard on their respective stands are shown below.

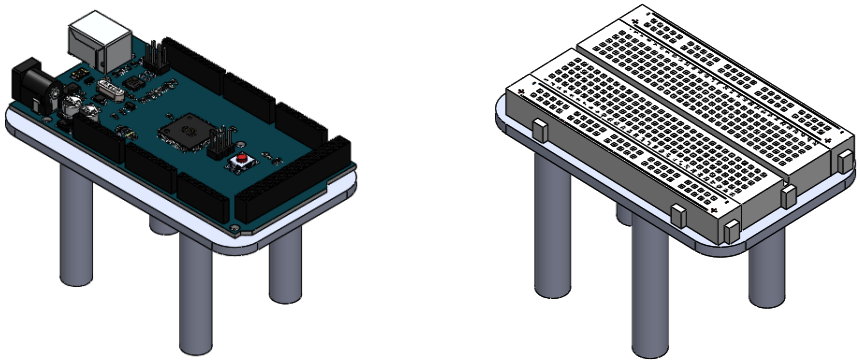


Figure 49: Arduino Mega subassembly (left) and breadboard subassembly (right).

6.4 Juice Box Item Retrieval/Delivery Mechanism

The juice box item retrieval/delivery mechanism we selected consists of a mechanical gripper attached to a four bar linkage. In terms of fabrication, a few components were 3D-printed for the gripper, and some machining had to be done for the four bar linkage.

The mechanical gripper was purchased as part of a kit, however, the built-in claw was rounded and had a relatively small surface area that could contact the juicebox we needed to grab. As a result, we decided to 3D print a claw that was better suited to our needs. This consisted of a gripper finger that would attach to a flat gripper pad in order to form the claw that could then be mounted to the original gripper. This change ensured that we had a greater surface area that would contact the juicebox and ultimately make gripping easier. In order to increase the gripping ability of our mechanism, we cut out strips of neoprene rubber and glued them onto the gripper pads. Shown below are the images of the 3D-printed gripper finger, pad, and combined subassembly.

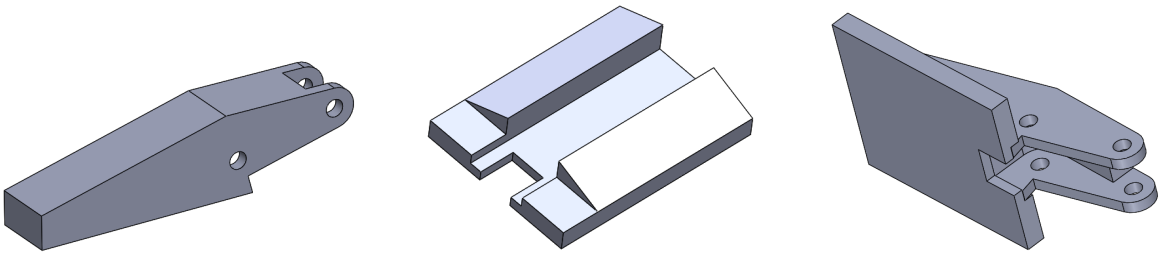


Figure 50: Gripper finger (left), gripper pad (middle), and combined gripper subassembly (right).

Many iterations were made of these 3D-printed gripper claws. At first, some of the tolerancing was off, such that it required a bit of manipulation to fit onto the claw the way we intended. These problems were easily fixed with some minor tweaks to the component dimensions. Later on, after assembling the entire robot and checking whether it was within the required size envelope, we noticed we were just over the requirement at certain gripping positions. Some additional dimensioning changes were made to shorten the length of the new gripper claws and they were 3D printed again to meet the sizing requirements.

As for the four bar linkage, the bulk of the fabrication for this mechanism was machining. We utilized steel rails for the linkage that came as part of the same kit that the gripper was purchased in. However, since the four bar design was custom, this involved cutting the individual rail links to length using a bandsaw. In addition to this, we also hand-filed down some of the drive shafts for proper fitment with the gears and the holes in the steel rail links they were to fit through. For assembly, the links were attached using hardware and the entire mechanism was bolted onto the chassis plates in various locations along the base. The four bar linkage is depicted below with a rail link and drive shaft, not shown to scale.

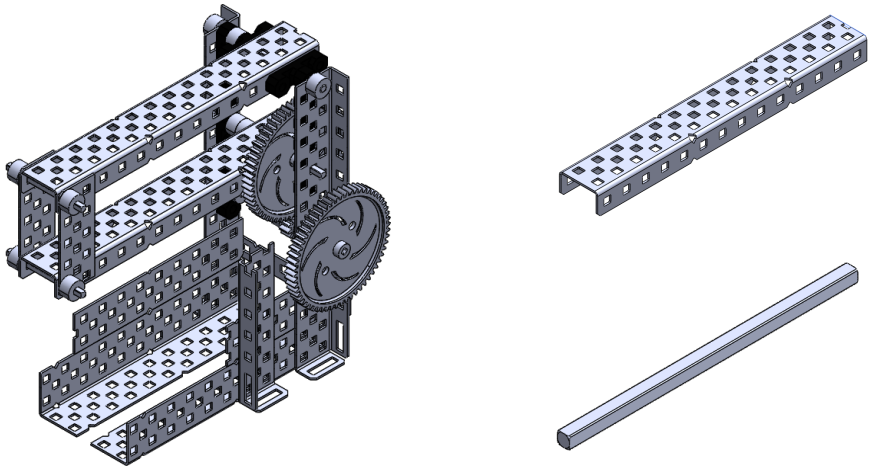


Figure 51: Four bar mechanism (left), steel rail link (top right), and steel drive shaft (bottom right).

6.5 Final Product Pictures

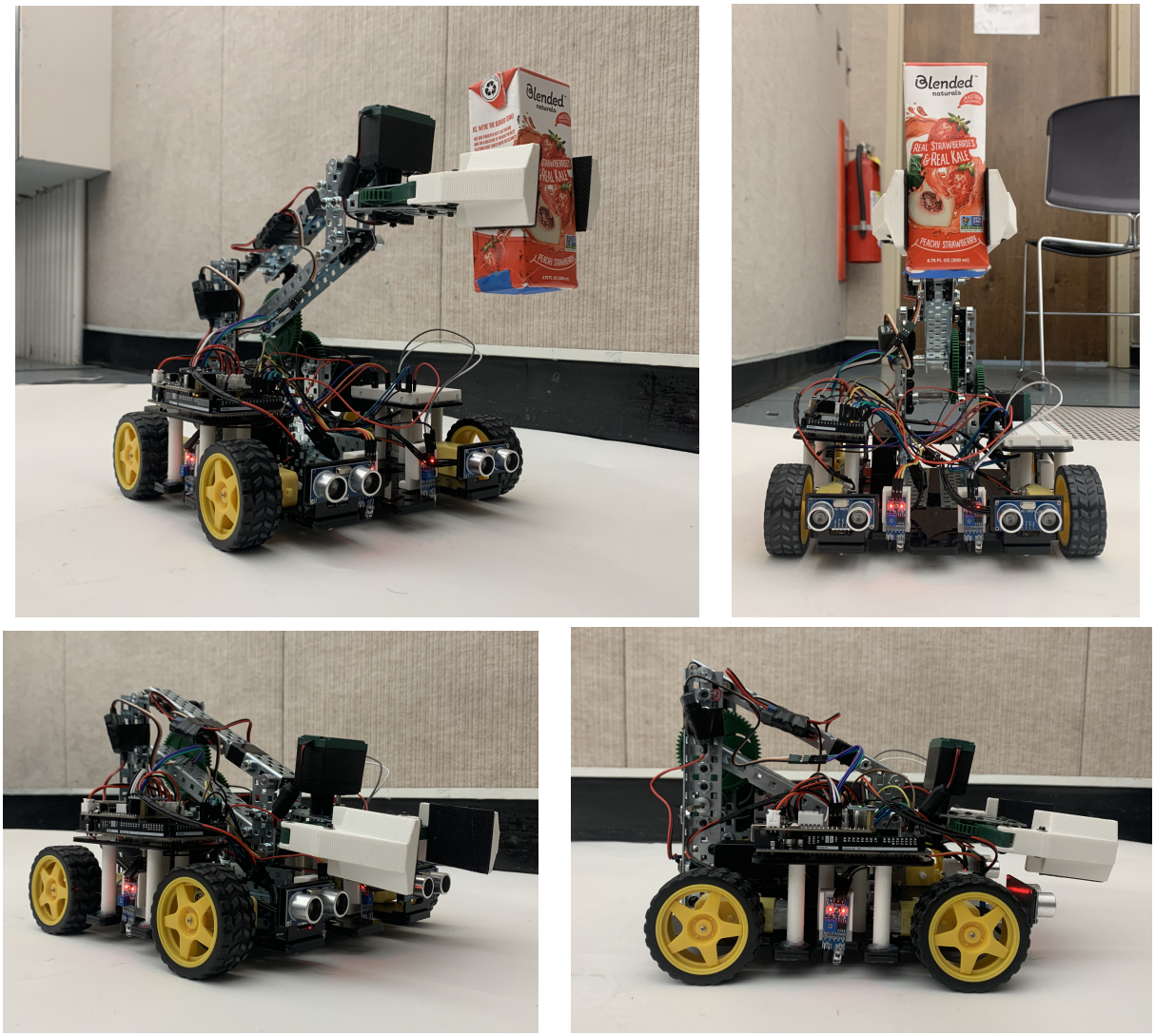


Figure 52: Top: Isometric (left) and front (right) views of the extended lift configuration and closed claw; Bottom: isometric (left) and side (right) views of the collapsed/lowered lift configuration and opened claw. The lowered lift and opened claw (bottom photos) is the starting configuration of the robot.

7. Product Performance Testing and Evaluation

7.1 Run Times

Once the robot was able to consistently complete the course for every pickup and dropoff location, it was then timed to see how long it took the robot to complete each section of the track.

7.1.1 Juice Box Retrieval

The juice box retrieval time started as soon as the robot started moving and ended when it grabbed and lifted the juice box. It is expected that the pick up locations that are further from the start (higher number pickup locations) will take longer for this section because the robot has to travel further. It is also expected that the pickup locations that are equidistant from the start will take the same time for the robot to collect the juice box from.

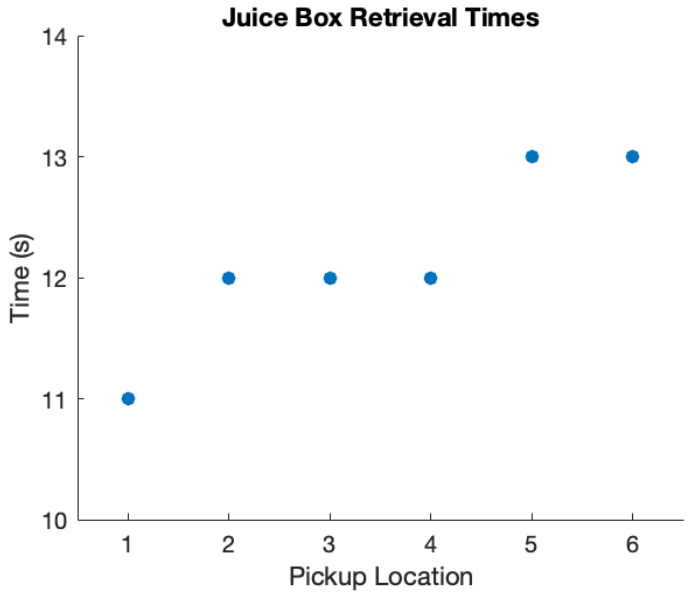


Figure 53: Time vs pickup location for juice box retrieval

As expected the lower pickup location numbers are faster than the higher ones. Also for the most part the pick up locations that are equidistant from the start are the same, except for location 1 and 2 which differ by one second.

7.1.2 Starting Area

The time for this section started when the robot picked up the juice box and ended when the robot's front wheels passed through the doorway. It is expected that the pick up locations closer to the gate (higher number pickup locations) will be shorter for this section because the robot has to travel a shorter distance. Same is in the previous section it is also expected that the pickup locations equidistant from the doorway will take the same amount of time.

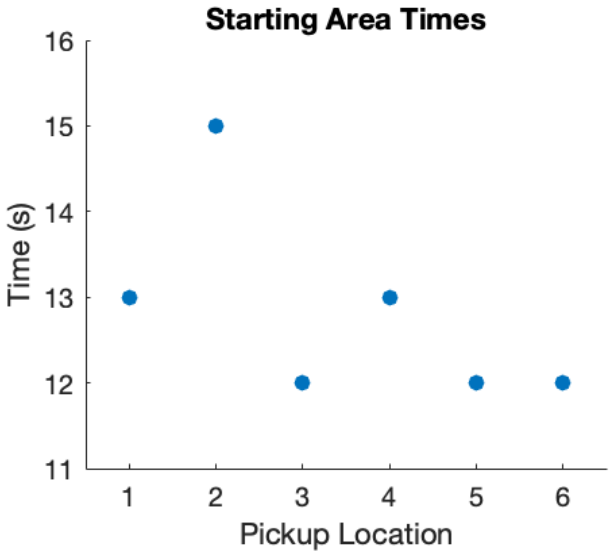


Figure 54: Time vs pickup location for the starting area

With increasing pick up location numbers there is generally a downwards trend in time, which is what was expected. However for pickup locations 1 and 2, which are equidistant from the doorway, the times differed significantly. After reviewing the footage this is because during this trial for pickup location 2 the robot got stuck with its wheel spinning for a second while it was trying to turn back onto the main line.

7.1.3 Obstacle Area

The time for this section started when the robot's front wheels passed through the doorway, and ended when it stopped in front of the moving obstacle. This time should remain the same across all of the trials.

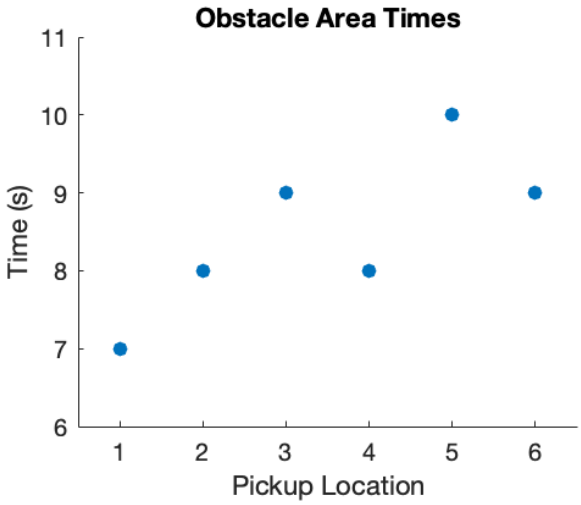


Figure 55: Time vs pickup location for the obstacle area

The average time across all six trials for the obstacle area was 8.5 seconds with a standard deviation of 1. As seen in the figure this short section of the track had times with a range of 3 seconds.

This large range is because the robot had to navigate two 45 degree turns in this section, and since the normal line tracking algorithm was used throughout the section it took the robot highly variable amounts of time to figure out these 45 degree turns.

7.1.4 Wilson Warehouse

The time for this section started once the moving obstacle moved out of the way, and ended when the back wheels exited the bumpy terrain. This time should also remain the same across all of the trials.

Figure 56: Time vs pickup location for the Wilson Warehouse area

The average time across all six trials for the Wilson Warehouse area was 7.7 with a standard deviation of 0.5. It makes sense that the standard deviation of times was low in this section because the robot just had to move in a straight line. Our robot handled the bumps in this section with no problem so driving in a straight line produced consistent results across all of the trials.

7.1.5 Juice Box Delivery

The time for this section started when the back wheels exited the bumpy terrain and ended when the juice box touched the ground in the drop off location. It is expected for this section that the first drop off location will be much faster than the rest because the robot only does one less turn that is relatively time consuming. For the rest of the dropoff locations it is expected that the higher number drop off locations will take longer because the robot has to travel a longer distance before it delivers the juice box.

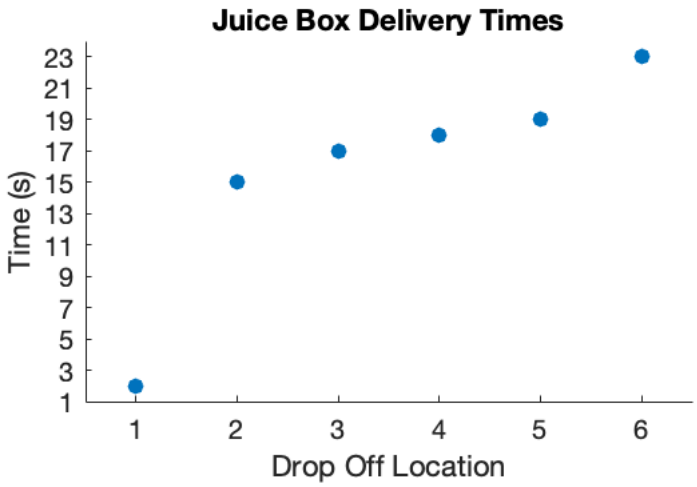


Figure 57: Time vs drop off location for the juice box delivery

This section went fully as expected. The first drop off location was much faster than the rest coming in at two seconds, with the next fastest delivery time being 15 seconds. Also as expected, the delivery time was monotonically increasing with increasing drop off location number.

7.1.6 Finish

The time for this section starts when the juice box touches the ground in the drop off location and ends when the robot stops at the last wall. It is expected for this section that the lower number drop off locations will be slower because they are further from the back wall, so the robot will have to travel further in this section.

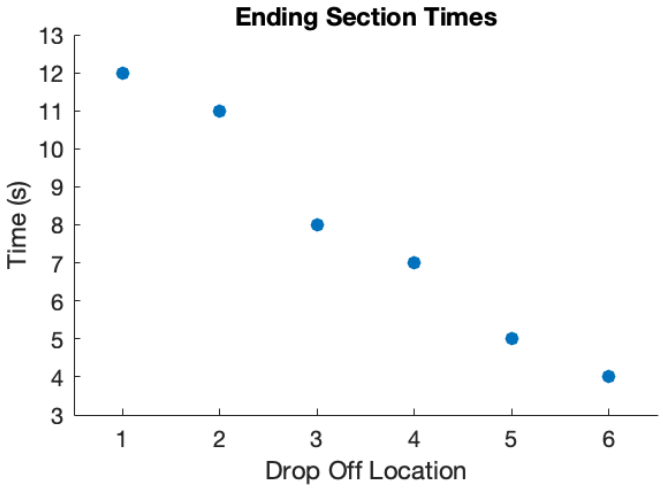


Figure 58: Time vs drop off location for the ending section

This section also went fully as expected with the ending section time monotonically decreasing with increasing drop off location.

7.2 Overall Performance

Overall the robot performance times have been tabulated below. These are not all of the trials we tried, but a sample of the most successful runs after lots of debugging and fine-tuning of the code was performed.

Table 13: Robot performance times for each section and the total time

Pickup and Dropoff #	Juice Box Retrieval (s)	Starting Area (s)	Obstacle Area (s)	Wilson Warehouse (s)	Juice Box Delivery(s)	Finish (s)	**Total Time (s)**
1	11	13	7	8	2	12	**53**
2	12	15	8	8	15	11	**69**
3	12	12	9	7	17	8	**65**
4	12	13	8	7	18	7	**65**
5	13	12	10	8	19	5	**67**
6	13	12	9	8	23	4	**69**

The total time was calculated by adding up all of the section times. This helps to control for the time that the robot is sitting waiting for the moving obstacle to move out of the way.

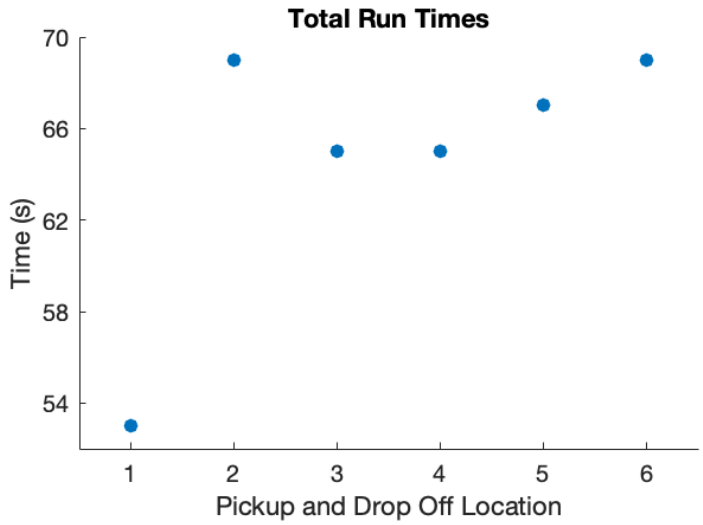


Figure 59: Time vs pickup and drop off location for the entire course.

Overall the robot averaged a total time of 64.7 seconds with a standard deviation of 6 seconds across the six timed trials that were performed. It is clear looking at the figure that the trial with pickup and drop off location number 1 is an outlier from the other trials, coming in much faster than the rest. If trial 1 is excluded, then the robot averaged a total time of 67 seconds with a standard deviation of 2 seconds. The standard deviation is much lower than including trial 1 showing that it was an outlier. All of this saved time came from the juice box delivery section, because at drop off location 1 there is one less turn that needs to be done than all of the other dropoff locations, which is quite time consuming.

7.3 Problems Encountered

Of course throughout the robot design and testing process we encountered some unexpected challenges. In our first iteration of the robot we were using front wheel drive with caster wheels in the rear. When we first started testing this version we found that the front wheels would often lose traction and start spinning, to the point where the robot was unable to move. We think this is because the track is a bit more slippery and uneven than we had anticipated. Our robot also ended up having a more rearward CG than expected because our four bar tower was initially going to be made of wood, but its weight increased significantly when we changed to using the metal vex parts. This more rearward CG also led to the front wheels having less traction. Another issue with the caster wheels was that they would start to lock up when the robot would try to turn 90 degrees, an important aspect of the obstacle course.

The solution to this issue was to change our design to four wheel drive, so we had to significantly redesign our chassis to compensate for this. This instantly fixed the movement and locking up issues but presented issues of its own. Now that the robot was using skid steering it led to inconsistent turning on different sections of the track. This solution to this problem was to actively use the IR sensors while turning to perfectly turn onto the branches, instead of using time delays, which with a bit of tweaking solved the problem.

Another challenge we faced was initially using the ELEGOO line tracking 3 sensor IR module, because while it worked, it wasn't very effective. The sensor placement was too narrow for the line width, so the robot would get very off angle before any sensor moved off the line. Our solution was to use the wider placement of two individual IR sensors that are placed perfectly outside the line as is mentioned in earlier sections, which has been very effective.

Finally, we worked to make the robot faster so that we could attempt to win the competition by removing all of the unnecessary stops that were useful for debugging, and by increasing the robot speed on straightaways.

8. Work Breakdown Schedule

8.1 Work Breakdown Schedule Diagram

Table 14: Work Breakdown Schedule Winter 2023

Winter 2023 Work Breakdown Schedule		
Week	**Meetings**	**Tasks**
1 (1/09)	1/09 - GM	Make design concept sketches
2 (1/16)	Holiday	
3 (1/23)	1/23 - GM	Make our 5-person team and make member introductions
4 (1/30)	1/30 - GM 2/01 - ZM	Select three design concepts Write Design Concept Proposal
5 (2/06)	2/06 - GM 2/08 - ZM 2/12 - ZM	Delegate CAD and tasks by subsystem Design concept solid CAD model Review CAD together and make additional changes
6 (2/13)	2/13 - GM	Spec components that will be used Figure out which parts will be manufactured or purchased
7 (2/20)	2/20 - GM	Start working on Conceptual Design Report Create pairwise comparison chart and objectives tree Select a final design concept from initial three designs Preliminary theoretical hand calculations and analyses
8 (2/27)	2/27 - GM	Finish Conceptual Design Report
9 (3/06)	3/06 - GM	Discuss report feedback
10 (3/13)	3/13 - GM	Start working on Preliminary Design Report Review feedback from Conceptual Design Report Modify robot design according to feedback Adjust hand calculations for accuracy
11 (3/20)	3/22 - ZM	Create and add parts to Bill of Materials Finish Preliminary Design Report

Table 15: Work Breakdown Schedule Spring 2023

Spring 2023 Work Breakdown Schedule		
Week	**Meetings**	**Tasks**
1 (4/03)	4/03 - GM 4/06 - ZM 4/09 - ZM	Create electronics circuit design diagram Research motor encoders Start speccing and sourcing materials and components
2 (4/10)	4/10 - LM 4/10 - GM 4/16 - ZM	Submit the purchase order to TA Laser cut V1 of the chassis plate 3D print gripper claws and IR sensor mounts Adjust size dimensions of chassis Finish speccing and sourcing materials and components
3 (4/17)	4/17 - GM 4/19 - ZM	Fabrication and assembly 3D print rear wheel housings Design state flow diagram with pseudo-code Design low-level control flow diagram of motors and sensor reading Continue research and spec motor encoders
4 (4/24)	4/24 - ZM 4/24 - LM 4/24 - GM	Fabrication and assembly 3D Print adjusted IR sensor mounts Fully assemble the four bar linkage lift
5 (5/01)	5/01 - GM	Wiring, programming, and debugging of gripper
6 (5/08)	5/08 - LM 5/08 - GM 5/11 - ZM	Laser cut V2 of the chassis plates
7 (5/15)	5/15 - GM 5/18 - ZM	Develop and implement code and test robot. First few stages of the code
8 (5/22)	5/22 - LM 5/22 - GM 5/25 - WD	Laser cut V3 final version of the chassis plates 3D Print final version of the gripper claws Assemble the final assembly of the robot with the new chassis plates Continue testing robot and adjusting code
9 (5/29)	5/30 - WD 6/04 - ZM	All team demo in the venue Start working on Oral Presentation slides Continue testing robot and adjusting code Start working on Final Design Report
10 (6/05)	6/05 - GM 6/07 - ZM	Finish Oral Presentation slides; Record presentation video Continue working on Final Design Report
11 (6/12)	6/14 - ZM	Complete competition robot run on venue Finish and submit Final Design Report, Write Peer Evaluation Forms

8.2 Work Breakdown Schedule Dictionary

GM = Group Meeting
ZM = Zoom Meeting
LM = Lecturer/TA Meeting
WD = Work Day

9. BOM and Cost Analysis

9.1 Assembly Drawings

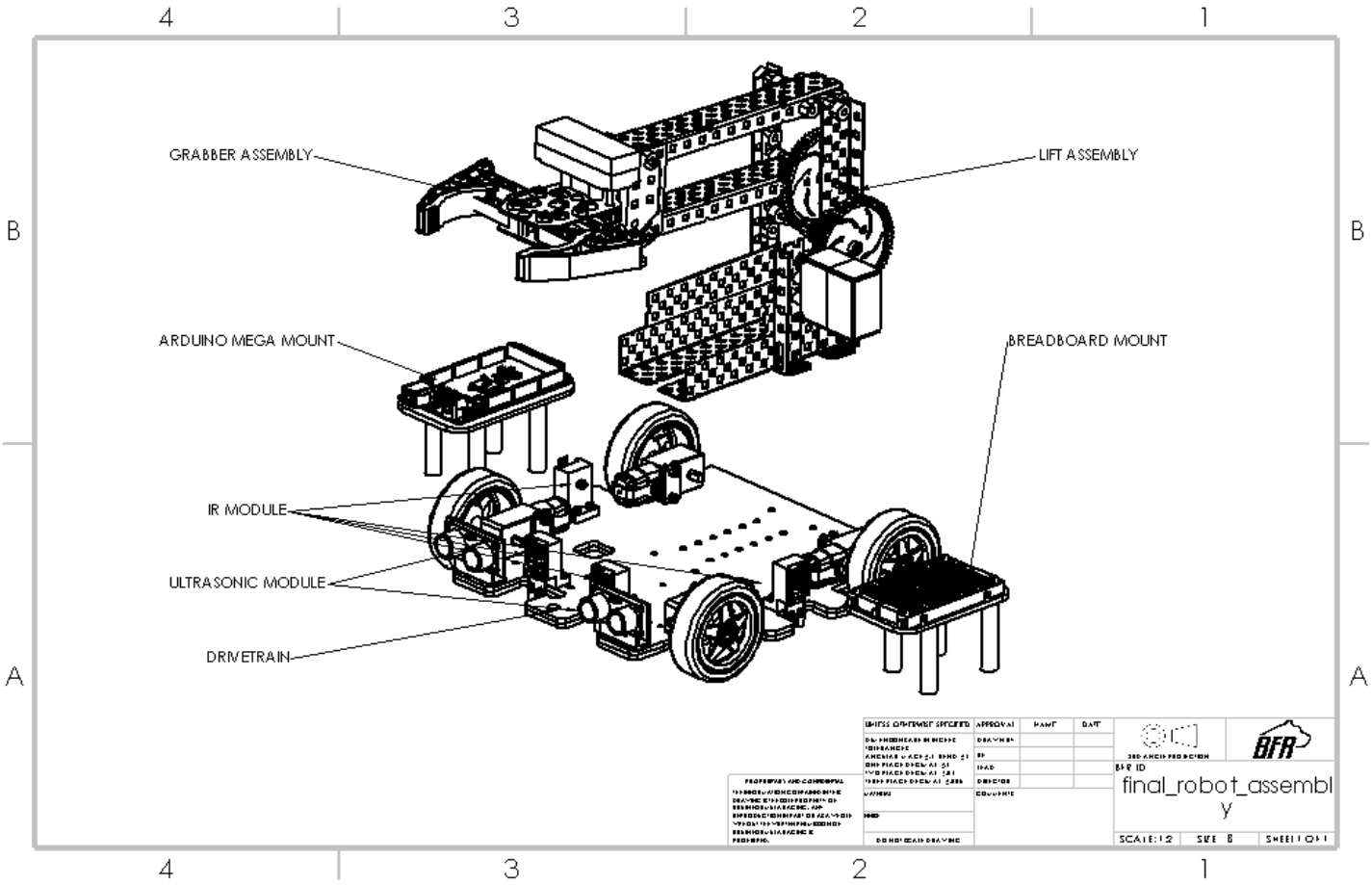


Figure 60: Engineering drawing of top-level assembly exploded view.

9.2 BOM

Index	Subsystem	Description	Vendor	QTY	Price	Total Price
1	Drivetrain/chassis	Wheel module (motor, gearbox, wheel)	Elegoo	4	$0.00	$0.00
2	Drivetrain/chassis	Chassis plate	Makerspace	2	$0.00	$0.00
3	Drivetrain/chassis	Hardware	Provided	50	$0.00	$0.00
4	Lift	VEX clawbot kit	VEX Robotics	1	$108.83	$108.83
5	Lift	Hardware	Provided	10	$0.00	$0.00
6	Gripper	VEX clawbot kit	VEX Robotics	1	$108.83	$108.83
7	Gripper	3D printed grippers	Makerspace	2	$0.00	$0.00
8	Gripper	Rubber adhesive sheet	Amazon	1	$9.49	$9.49
9	Gripper	Hardware	Provided	10	$0.00	$0.00
10	Electronics	Arduino Mega	Amazon	1	$20.99	$20.99
11	Electronics	Arduino Mega mount	Makerspace	1	$0.00	$0.00
12	Electronics	Battery	Provided	1	$0.00	$0.00
13	Electronics	Ultrasonic sensor	Provided	2	$0.00	$0.00
14	Electronics	Ultrasonic sensor mount	Makerspace	2	$0.00	$0.00
15	Electronics	IR sensor	Amazon	4	$0.90	$3.60
16	Electronics	IR sensor mount	Makerspace	4	$0.00	$0.00
17	Electronics	Lift encoder	Amazon	1	$9.00	$9.00
18	Electronics	Force-sensing resistor	Amazon	1	$14.50	$14.50
19	Electronics	Breadboard	Provided	1	$0.00	$0.00
20	Electronics	Breadboard mount	Makerspace	1	$0.00	$0.00
21	Electronics	22 AWG wires	Provided	100	$0.00	$0.00
22	Electronics	Hardware	Provided	50	$0.00	$0.00
					Total Expense	**$275.24**

Figure 61: Top-level bill of materials. Donated and provided materials are $0.

Purchaser	Subsystem	Date Ordered	Vendor	Description	Price	PO Drive Link	Receipt Drive Link	Order Status	Reimbursement Status
Jonathan	Gripper	4/14/2023	VEX Robotics	Clawbot kit	$217.67		https://drive.google.c	Received	
School	Gripper	4/14/2023	Amazon	Gripper servo/sensor	$77.73	https://docs.goo		Received	
School	Electronics	4/24/2023	Amazon	Rotary encoders and casters	$49.25	https://docs.goo		Received	
Jonathan	Lift	4/26/2023	Mcmaster-Carr	Key stock and ball rollers	$28.21		https://drive.google.c	Received	
School	Electronics	5/1/2023	Amazon	Spare Arduino mega	$20.99	https://docs.goo		Received	
School	Drivebase	5/8/2023	Amazon	Missing casters, rubber sheet	$22.78	https://docs.goo		Received	
School	Electronics	5/27/2023	Amazon	Spare Arduino mega	$20.99		https://docs.google.c	Received	
				Total	$437.62				

Figure 62: Purchase order tracking spreadsheet.

As the figures above show, there is a distinction between the money we spent ordering components and the final robot cost. We spent $437.62 on Purchase Orders, still below the given $500 budget for this project. The total spent was substantially more than our final robot cost because there were several components that we ordered that didn't get used on the final robot. This included the caster wheels, servo motor, and ball roller wheels.

9.3 Final Cost Analysis

9.3.1 Material Costs

As shown in the BOM and tracker above, our total expense and robot price is well below the $500 requirement. The majority of the cost came from the VEX robotics kit, which cost $217.67 total. While we didn't make use of the entire cost of the kit, we used many of the parts including the motors and pre-made claw. Many smaller robot components including fasteners, wires, and some sensors were donated to our group by the school. The total cost of our robot came out to $275.24.

Most of our electronics and drivetrain components came from the Elegoo kit bot, which was also provided by the school and thus came at zero cost for us. We use the drivetrain modules, Arduino shield, battery, sensors, and wires. Most of the wires and the breadboard were provided by the class as well and thus $0 on our end.

Lastly, most of the hardware (bolts, nuts, washers, etc.) came from a pre-existing kit from one of our team members, which is thus $0 for our team. All of this combined allowed us to have a very low budget for the project.

9.3.2 Labor Costs

While labor costs aren't associated with the robot, we can analyze and produce a theoretical model for how much this might have cost if it were to be accounted for.

- Estimated entry-level engineer/intern hourly wage: $30/hr
- Hours per week: 10
- Number of weeks: 20
- Number of engineers: 5
- **Total pay for each engineer: $6,000**
- **Total labor cost for company: $30,000**

10. Design Requirement Satisfaction

Table 16: High-Level Design Requirements Satisfaction Status

HLDR	Description	Comment	Satisfaction Status (w/ explanation if not satisfied)
1	Device must complete the full task set	Go to shelf location, remove item, lower item through doorway, navigate two 90-degree turns, avoid all obstacles, deliver item to destination in the upright orientation, stop after delivery	**Satisfied**
2	Device must operate autonomously	Can tell the device results of the die rolls before start of the run and press start	**Satisfied**
3	The device must complete the task set within 10 minutes	Any and all trials must be finished in this time limit	**Satisfied** (avg. trial time for our robot is about 65 seconds)
4	Starting dimensions must be limited to 20x25x30 cm (HxWxL)	Dimensions can change afterwards	**Satisfied**
5	Electric batteries are the only source of power for the device	Choose between disposable and rechargeable batteries (no mixing) Disposable: Ten 1.5V plus two 9V batteries (12 total) Rechargeable: Three 3.7V, three 9V, and six 1.2V batteries (12 total)	**Satisfied**
6	All sensors must be mounted on the device	Should not be a problem	**Satisfied**
7	Device may not leave behind any parts or components	Should not be a problem	**Satisfied**
8	Actuators must be designed and fabricated by the team	Only motors are excluded from this requirement	**Satisfied**
9	Purchasing budget is limited to $500 per team	Microcontrollers, routers, cables provided do not count towards this limit	**Satisfied**

11. Conclusion

The task for this project was to design, manufacture, and operate an autonomous robot capable of (1) retrieving an item, in this case a juice box, (2) navigating through a course with obstacles present while safely carrying the item (no spilling of juice), and (3) delivering the item to a designated location at the end of the course. The robot must perform all these functions autonomously, and will achieve this by line-tracking a solid black line laid out on the floor of the course. In addition to the autonomous operation requirement, there were several other high-level design requirements given for this project, including a maximum monetary budget, battery power source requirements, and maximum starting dimensions of the robot. Furthermore, several low-level design requirements of the robot such as steering/drivetrain type, number of motors, pickup/delivery mechanism, etc. were made by our team based on our selected design concept.

Our team's final selected design concept was a combination of the drivetrain of Design Concept 2 and the intake and lift mechanism of Design Concept 4. Using the drivetrain of Design Concept 2 (4WD) allowed the greatest reliability with turning and traction, and using the lift and gripper/intake concept from Design Concept 4 provided the necessary lift height and gripper strength to grab the juicebox from the shelf, hold it during course navigation, and deliver it on the ground at the end of the course.

A fully-featured computer-aided design (CAD) model of our Final Design Concept, including appropriate material selection for all the components, was created and used with static analysis to perform Preliminary Calculations to ensure the design was capable of completing the robot tasks. These tasks include (1) handling the juicebox load through all phases of pickup, transport, and delivery (weight balance and center of mass will shift as the lift and gripper mechanism moves with the juicebox), (2) maneuvering through slight inclines (up to 5 degrees), and (3) having enough propulsion and motor power to navigate the course, including the bumpy section. Calculations were performed and initial assessment of the results gave us confidence that our design was viable and ready to build.

Our design was built using a combination of off the shelf parts, 3D-printed parts, and laser-cut parts. Once built, coding the robot to perform all its tasks autonomously was the next task. This was a big trial and error effort but in the end our robot was able to successfully complete all its design requirements and complete the course. After comprehensive debugging and fine-tuning of the code, we are satisfied with the consistency and repeatability of the robot completing the course.

This project taught our team a lot in terms of technical engineering work as well as prototyping, project management, and teamwork. We learned that sometimes in the engineering process, it is important to be able to adapt to unexpected challenges and adjust your plan and ideas accordingly. In our case, we initially wanted to do a drivetrain with rear passive caster wheels because it would allow tighter turning, but in our initial testing we realized that the caster wheel setup was not working as expected. So instead we pivoted to a more traditional four-wheel-drive setup and this successfully met our needs. In terms of managing the project and its timeline, we learned that having weekly team meetings in Zoom or in-person to review next steps and assign tasks was crucial for keeping our robot building progress steady and on track. To avoid procrastinating and leaving lots of work to the last minute, we set soft and hard deadlines for our team to finish certain aspects of the robot such as assembling all components to the chassis and coding the gripper claw. We had good text and in-person communication within our team and would help each other out with tasks when needed. Lastly, we were reminded of the importance of asking the TAs and Professor clarifying questions early in the process right when they came up rather than waiting, as this helped us avoid doing unnecessary work and gave us more time to make meaningful progress. Overall, this project was a culmination of engineering education

as well as an exercise in working together as a team, overcoming challenges, and navigating the twists and turns of a full concept-to-deployment engineering process.

12. References

[1] The Verge, https://www.theverge.com/2022/6/21/23177756/amazon-warehouse-robots-proteus-autonomous-cart-delivery; accessed Feb. 1, 2023

[2] AZ Central, https://www.azcentral.com/story/news/local/southwest-valley/2020/10/08/amazon-facility-metro-phoenix-uses-robotics-more-way/3625228001/; accessed Feb 1, 2023

[3] Starship, https://www.starship.xyz/; accessed Feb 1, 2023

[4] TechCrunch, https://techcrunch.com/2019/06/06/how-amazons-delivery-robots-will-navigate-your-sidewalk/; accessed Feb 2, 2023

[5] Circuit Digest, https://circuitdigest.com/microcontroller-projects/line-follower-robot-using-arduino; accessed Feb 3, 2023

[6] University of Florida, https://mae.ufl.edu/designlab/Class%20Projects/Background%20Information/Friction%20coefficients.htm; accessed Feb 28, 2023

[7] Vertical, “DC Gearbox Motor” https://www.verical.com/datasheet/adafruit-brushless-dc-motors-3777-5912007.pdf

[8] VEX Robotics, “2-Wire Motor 393”, https://content.vexrobotics.com/docs/instructions/276-2177-instr-0414_v2.pdf

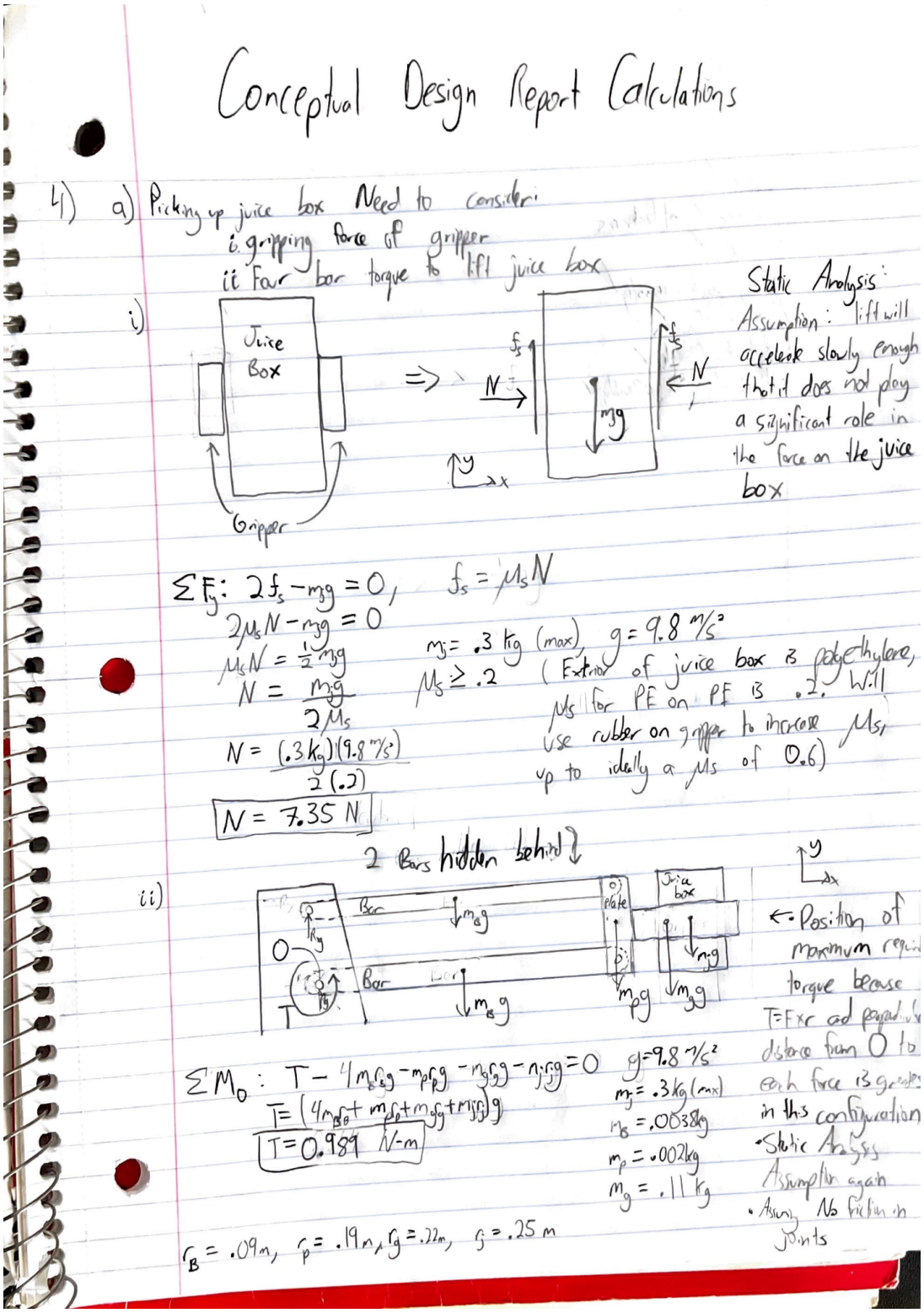
[9] HiLetGo IR Sensor Module, https://www.amazon.com/HiLetgo-Infrared-Avoidance-Reflective-Photoelectric/dp/B07W97H2WS

[10] HC-SR04 ultrasonic sensor, https://www.electroschematics.com/hc-sr04-datasheet/

[11] HC-020K Double Speed Measuring Module with Photoelectric Encoders For Experiment, https://www.amazon.com/HC-020K-Measuring-Photoelectric-Encoders-Experiment/dp/B00EERJDY4

13. Appendix

13.1 Full Hand Calculations



$m = 3.37\ 16 = 1.53\ kg \rightarrow F_{mg} = (1.53\ kg)(9.8\ m/s^2) = 14.994\ N$

L = WHEELBASE = $6.45\ in = 0.1638\ m$

L_c = DISTANCE RW TO CoM = $4.68\ in = 0.1189\ m$

h_c = HEIGHT OF CoM = $2.62\ in = 0.06655\ m$

$\theta : 0 - 5°$

2) b) CALCULATE MINIMUM FRICTION COEFFICIENTS.

$$\mu = \frac{L \sin\theta}{(\gamma_r - \gamma_f)(h \sin\theta - L_c \cos\theta) + \gamma_r L \cos\theta}$$

@ $\theta = 5°$ → @ $\theta = 0°$, $\mu = 0$

FWD: $\gamma_f = 1$, $\gamma_r = 0$ →

$$\mu = \frac{(0.1638\ m)(\sin(5°))}{(0-1)((0.06655\ m)\sin(5°) - (0.1189\ m)\cos(5°)) + 0}$$

$\boxed{\mu = 0.1267}$ (FWD)

RWD: $\gamma_f = 0$, $\gamma_r = 1$ →

$$\mu = \frac{(0.1638\ m)\sin(5°)}{(1-0)((0.06655\ m)\sin(5°) - (0.1189\ m)\cos(5°)) + (1)(0.1638\ m)\cos(5°)}$$

$\boxed{\mu = 0.2825}$ (RWD)

AWD: $\gamma_f = 1$, $\gamma_r = 1$ →

$$\mu = \frac{(0.1638\ m)\sin(5°)}{(1-1)(\quad) + (1)(0.1638\ m)\cos(5°)}$$

$\boxed{\mu = 0.0575}$ (AWD)

2) c) CALCULATE THE NORMAL FORCE BETWEEN REAR WHEELS + GROUND.

$$F_{NR} = N_r = \frac{F_{mg}(L - L_c)\cos\theta + F_{mg} h \sin\theta}{L}$$

$$= \frac{(14.994N)[(0.1638\ m - 0.1189\ m)\cos(5°) + (0.06655\ m)\sin(5°)]}{(0.1638\ m)}$$

$\boxed{F_{NR} = N_r = 4.625\ N}$

2) d.) CALCULATE THE NORMAL FORCE BETWEEN FRONT WHEELS + GROUND.

$$F_{NF} = N_f = \frac{F_{mg} L_c \cos\theta - F_{mg} h \sin\theta}{L}$$

$$= \frac{(14.994\,N)[(0.1189\,m)\cos(5°) - (0.06655\,m)\sin(5°)]}{(0.1638\,m)}$$

$$\boxed{F_{NF} = N_f = 10.312\,N}$$

2) e) CALCULATE THE TOTAL TRACTIVE FORCE FOR YOUR CONCEPT.

$F_{TF} = \mu N_f$ AND $F_{TR} = \mu N_r \rightarrow F = F_{TR} + F_{TF}$

FWD: $F_{TF} = \mu N_f = (0.1267)(10.312\,N) = 1.3065\,N$

$F_{TR} = \mu N_r = (0.1267)(4.625\,N) = 0.586\,N$

$F = F_{TF} + F_{TR} = 1.3065\,N + 0.586\,N \rightarrow \boxed{F = 1.8925\,N}$ (FWD)

RWD: $F_{TF} = \mu N_f = (0.2825)(10.312\,N) = 2.9131\,N$

$F_{TR} = \mu N_r = (0.2825)(4.625\,N) = 1.3066\,N$

$F = F_{TF} + F_{TR} = 2.9131\,N + 1.3066\,N \rightarrow \boxed{F = 4.2197\,N}$ (RWD)

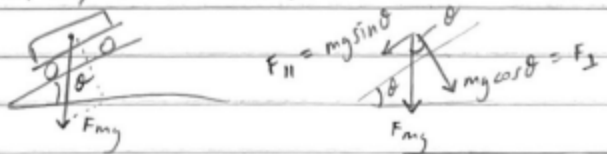
AWD: $F_{TF} = \mu N_f = (0.0875)(10.312\,N) = 0.9023\,N$

$F_{TR} = \mu N_r = (0.0875)(4.625\,N) = 0.4047\,N$

$F = F_{TF} + F_{TR} = 0.9023\,N + 0.4047\,N \rightarrow \boxed{F = 1.307\,N}$ (AWD)

2) f) FORCE OF GRAVITY ACTING ALONG THE SLOPE

$F_{mg} = 14.994\,N$, $\theta = 5°$ (MAX)



$F_{\parallel} = mg\sin\theta = (14.994N)(\sin(5°)) \rightarrow \boxed{F_{\parallel} = 1.3068\,N}$

$F_{\perp} = mg\cos\theta = (14.994N)(\cos(5°)) \rightarrow \boxed{F_{\perp} = 14.937\,N}$

3) b) DETERMINE β (% WEIGHT DISTRIBUTION BETWEEN REAR AND FRONT WHEELS).

$$\beta_f = \frac{N_f}{F_m \cos\theta} = \frac{(10.312N)}{(14.994\,N)\cos(5°)} \rightarrow \boxed{\beta_f = 0.6904}$$

$$\beta_r = \frac{N_r}{F_m \cos\theta} = \frac{(4.625\,N)}{(14.994N)\cos(5°)} \rightarrow \boxed{\beta_r = 0.3096}$$

13.2 Full robot code used in competition

```
////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////
//////////
////////////////////////////////////////////////////////////////////////////// MAE162E GROUP 5
CODE ////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////
//////////


//V12
//Fast and ready competition code


////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////
//////////
////////////////////////////////////////////////////////////////////////////// INCLUDE
LIBRARIES ///////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////
//////////


#include <avr/wdt.h>
#include <stdio.h>
#include <string.h>
#include "DeviceDriverSet_xxx0.h"
#include <Servo.h>


////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////
//////////
////////////////////////////////////////////////////////////////////////////// VARIABLE/PIN
DEFINITIONS /////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////
//////////
```

```
#define IR_SENSOR_RIGHT 30
#define IR_SENSOR_LEFT 32
#define EXT_IR_SENSOR_RIGHT 34
#define EXT_IR_SENSOR_LEFT 23
#define trigPinL 24
#define echoPinL 22
#define trigPinR 28
#define echoPinR 26


//IR Remote
DeviceDriverSet_Motor myMotors;
DeviceDriverSet_IRrecv IRremote;
uint8_t IRrecv_button;


//Motor Control
int leftWheels = 6;
int rightWheels= 5;
int leftWheelsDir = 8;
int rightWheelsDir = 7;
int motorsEnable = 3;
int speed1 = 70 + 25;
int speed2 = 40;
int speed3 = 75;
int speed4 = 110;
int speed5 = 70;


//ultrasonic parameters
float distanceR, durationR;
float distanceL, durationL;
float stoppingDistance = 3;
float stoppingDistance2 = 10;
float stoppingDistance3 = 13;


//IR Sensors
int rightIRSensorValue = 0;
int leftIRSensorValue = 0;
int EXTrightIRSensorValue = 0;
int EXTleftIRSensorValue = 0;
```

```
//Line Tracking and Branch Control
bool stopTracking1 = false;
bool stopTracking2 = true;
bool BranchDetect1 = false;
bool leavingFirstBranches = false;
bool lastBranch1 = false;
int branchNumber1 = 0;
int branchCounter1 = 0;
int branchesLeft1 = 0;
int branchNumber2 = 0;
int branchCounter2 = 0;
int branch1isOnTheLeft = 0;
int shelfPosition1 = 0;
int shelfPosition2 = 0;


//moving obstacle
int delayCounter = 0;
int delayCounter2 = 0;


//IR remote
bool firstButtonPressed = false;
bool secondButtonPressed = false;
int enterPressed1 = 0;
int enterPressed2 = 0;
bool RUN_PROGRAM = false;
int shutoffPressed = 0;
bool stage1 = false;
bool stage2 = false;
bool stage2b = false;
bool stage2c = false;
bool stage3 = false;
bool stage4 = false;
bool stage4b = false;
bool stage4c = false;
bool stage5 = false;
bool stage6 = false;
bool stage6b = false;
bool stage6c = false;
bool stage6d = false;
bool stage6e = false;
```

```
bool stage7 = false;
bool stage7b = false;
bool stage7c = false;
bool stage7d = false;
bool stage8 = false;
bool stage9 = false;
bool stage9b = false;
bool stage9c = false;
bool stage10 = false;
bool stage11 = false;
bool switch1 = true;
bool switch2 = true;
bool switch3 = true;
bool switch4 = true;
bool switch5 = true;


//Pickup
bool initiatePickup = false;
bool pickupComplete = false;


//Gripper and Lift
int gripperMotorPin = 10;
int liftMotorPin = 11;
Servo gripperMotor;
Servo liftMotor;


////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////
//////////
////////////////////////////////////////////////////////////////////////// VOID SETUP
////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////
//////////

void setup() {
Serial.begin(9600);


//IR initialization
```

```
pinMode(IR_SENSOR_RIGHT, INPUT);
pinMode(IR_SENSOR_LEFT, INPUT);
pinMode(EXT_IR_SENSOR_RIGHT, INPUT);
pinMode(EXT_IR_SENSOR_LEFT, INPUT);


//ultrasonic initialization
pinMode(trigPinL, OUTPUT);
pinMode(echoPinL, INPUT);
pinMode(trigPinR, OUTPUT);
pinMode(echoPinR, INPUT);


//IR Remote initialization
IRremote.DeviceDriverSet_IRrecv_Init();
myMotors.DeviceDriverSet_Motor_Init();


//Gripper and lift initialization
pinMode(gripperMotorPin, OUTPUT);
gripperMotor.attach(gripperMotorPin);
pinMode(liftMotorPin, OUTPUT);
liftMotor.attach(liftMotorPin);
}

////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////
//////////
/////////////////////////////////////////////////////////////////// VOID LOOP
////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////
//////////


void loop() {
//Emergency shutoff
shutoffPressed = detectProgramShutoff();
if(shutoffPressed == 1){
Serial.println("Shut Down");
RUN_PROGRAM = false;
stop();
}
```

```
//Getting first button info
if(!firstButtonPressed){
shelfPosition1 = receiveRemoteData1();
if(shelfPosition1 != 0){
firstButtonPressed = true;
Serial.println(shelfPosition1);
if(shelfPosition1 == 1 || shelfPosition1 == 3 || shelfPosition1 == 5){
branch1isOnTheLeft = 1;
} else {
branch1isOnTheLeft = 0;
}
if(shelfPosition1 == 2 || shelfPosition1 == 1){
branchNumber1 = 1;
branchesLeft1 = 2;
}
if(shelfPosition1 == 4 || shelfPosition1 == 3){
branchNumber1 = 2;
branchesLeft1 = 1;
}
if(shelfPosition1 == 5 || shelfPosition1 == 6){
branchNumber1 = 3;
branchesLeft1 = 0;
}
}
}
//Detecting enter pressed for first button
else if(enterPressed1 == 0){
enterPressed1 = detectEnterPressed1();
Serial.println("please press enter 1:");
}
//Getting second button info
else if(!secondButtonPressed){
shelfPosition2 = receiveRemoteData2();
Serial.println("enter 1 pressed:");
if(shelfPosition2 != 0){
secondButtonPressed = true;
branchNumber2 = shelfPosition2;
Serial.println(branchNumber2);
RUN_PROGRAM = true; stage1 = true;
}
}
```

```
//Both shelf positions have been read in
//Now start on the main program
if (RUN_PROGRAM){
Serial.println("Program running");
rightIRSensorValue = digitalRead(IR_SENSOR_RIGHT);
leftIRSensorValue = digitalRead(IR_SENSOR_LEFT);
EXTrightIRSensorValue = digitalRead(EXT_IR_SENSOR_RIGHT);
EXTleftIRSensorValue = digitalRead(EXT_IR_SENSOR_LEFT);
//Start stage1 which is line tracking until the right branch is reached
if (stage1) {
//Line follow
lineDetect(rightIRSensorValue, leftIRSensorValue, stopTracking1);
//If front IRs see a branch: pause
if(rightIRSensorValue == HIGH && leftIRSensorValue == HIGH && BranchDetect1 == false){
stop();
// delay(500);
branchCounter1++;
if(branchCounter1 < branchNumber1){
forward();
delay(500);
}
else if (branchCounter1 == branchNumber1){
stage1 = false; stage2 = true;
}
}
}
////Go forward until external irs are on line and raise lift
else if(stage2) {
forwardSlow();
if(EXTrightIRSensorValue == HIGH && EXTleftIRSensorValue == HIGH){
stop();
motorControl(80, liftMotor);
delay(3500);
motorControl(0, liftMotor);
stage2 = false; stage2b = true;
}
}
// go forward until ext irs are off black
else if(stage2b) {
forwardSlow();
if(EXTrightIRSensorValue == LOW || EXTleftIRSensorValue == LOW){
stop();
```

```
stage2b = false; stage2c = true;
}
}
// turn right or left until opposite ir is on branch
else if(stage2c){
if(branch1isOnTheLeft == 1){
turnLeft();
if(switch4){
delay(1500);
switch4 = false;
}
else if(rightIRSensorValue == HIGH) {
stop();
stage2c = false; stage3 = true;
}
}
else {
turnRight();
if(switch4){
delay(1500);
switch4 = false;
}
else if(leftIRSensorValue == HIGH) {
stop();
stage2c = false; stage3 = true;
}
}
}
//Going up to the wall and grabbing juice box
else if(stage3) {
distanceR = detectDistanceR();
distanceL = detectDistanceL();
lineDetectAndAvoidObstacles(rightIRSensorValue, leftIRSensorValue, false);
if(distanceR <= stoppingDistance && distanceL <= stoppingDistance){
stop();
//Grab juice box and pickup slightly
motorControl(-40, gripperMotor);
delay(1700);
motorControl(0, gripperMotor);
motorControl(80, liftMotor);
delay(300);
motorControl(0, liftMotor);
stage3 = false; stage4 = true;
```

```
}
}
//Reverse back to central black line
else if(stage4) {
backwardSlow();
if(EXTrightIRSensorValue == HIGH && EXTleftIRSensorValue == HIGH){
stop();
// lower lift
motorControl(-80, liftMotor);
delay(2200);
motorControl(0, liftMotor);
stage4 = false; stage4b = true;
}
}
// go forward until ext irs are off line
else if(stage4b) {
forwardSlow();
if(EXTrightIRSensorValue == LOW || EXTleftIRSensorValue == LOW){
stop();
// delay(300);
stage4b = false; stage4c = true;
stop();
// delay(200);
backwardSlow();
delay(2000);
stop();
// delay(200);
}
}
//turn correct way back on to line and stop when opposite front sensor sees line
else if(stage4c) {
if(branch1isOnTheLeft == 1){
turnRightLeftWheelsOnly();
if(switch5){
delay(3500);
switch5 = false;
}
if(leftIRSensorValue == HIGH) {
stop();
if(branchesLeft1 == 0){
stage4c = false; stage6 = true;
} else {
stage4c = false; stage5 = true;
```

```
}
}
}
else {
turnLeftRightWheelsOnly();
if(switch5){
delay(3500);
switch5 = false;
}
else if(rightIRSensorValue == HIGH) {
stop();
if(branchesLeft1 == 0){
stage4c = false; stage6 = true;
} else {
stage4c = false; stage5 = true;
}
}
}
}
//line follow until end of starting junctions
else if(stage5) {
lineDetect(rightIRSensorValue, leftIRSensorValue, false);
if(EXTrightIRSensorValue == HIGH && EXTleftIRSensorValue == HIGH) {
stop();
if(branchesLeft1 > 0) {
forward();
delay(500);
branchesLeft1--;
}
if (branchesLeft1 == 0) {
forward();
delay(500);
stage5 = false; stage6 = true;
}
}
}
//deal with moving obstacle
else if(stage6){
delayCounter++;
if(delayCounter < 160){
lineDetect(rightIRSensorValue, leftIRSensorValue, false);
}else if (delayCounter == 160){
stop();
```

```
//raise lift
motorControl(80, liftMotor);
delay(2600);
motorControl(0, liftMotor);
motorControl(-40, gripperMotor);
delay(500);
motorControl(0, gripperMotor);
}
else if(delayCounter > 160){
distanceR = detectDistanceR();
distanceL = detectDistanceL();
lineDetectAndAvoidObstacles(rightIRSensorValue, leftIRSensorValue, false);
if(distanceR <= stoppingDistance3 || distanceL <= stoppingDistance3){
stop();
stage6 = false; stage6b = true;
}
}
}
//wait for obstacle to move, lower lift
else if(stage6b) {
distanceR = detectDistanceR();
distanceL = detectDistanceL();
if(distanceR > stoppingDistance3 + 1 && distanceL > stoppingDistance3 + 1)
{
delay(1000);
motorControl(-80, liftMotor);
delay(2100);
motorControl(0, liftMotor);
stage6b = false; stage6c = true;
}
}
//line follow until start of ending junction
else if(stage6c) {
lineDetect2(rightIRSensorValue, leftIRSensorValue, false);
if(EXTrightIRSensorValue == HIGH && EXTleftIRSensorValue == HIGH) {
branchCounter2++;
stop();
if(branchNumber2 == 1){
forwardSlow();
delay(500);
stop();
stage6c = false; stage8 = true;
} else {
```

```
stop();
backwardSlow();
delay(2050);
stop();
stage6c = false; stage6e = true;
}
}
}
//drive forward until ext ir sensors are off black //skip this step
else if(stage6d) {
forwardSlow();
if(EXTrightIRSensorValue == LOW || EXTleftIRSensorValue == LOW){
stop();
stage6d = false; stage6e = true;
}
}
//Turn left until front right ir sensor is on line
else if(stage6e) {
turnLeftRightWheelsOnly();
if(switch1){
delay(3500);
switch1 = false;
}
else if(rightIRSensorValue == HIGH) {
stop();
stage6e = false; stage7 = true;
}
}
//line follow until correct junction and stop
else if(stage7) {
lineDetect2(rightIRSensorValue, leftIRSensorValue, false);
if(EXTrightIRSensorValue == HIGH) {
stop();
branchCounter2++;
if(branchNumber2 != branchCounter2){
forward();
delay(500);
}
else if(branchNumber2 == branchCounter2) {
stop();
backwardSlow();
delay(2200);
stop();
```

```
stage7 = false; stage7c = true;
}
}
}
//Drive forward until ext right ir is off black //skip this step
else if(stage7b) {
forwardSlow();
if(EXTrightIRSensorValue == LOW){
stop();
stage7b = false; stage7c = true;
}
}
// turn right until front left ir sensor is on branch
else if(stage7c) {
turnRightLeftWheelsOnly();
delay(3600);
stop();
stage7c = false; stage8 = true;
}
//drop off juice box
else if(stage8) {
motorControl(-80, liftMotor);
delay(200);
motorControl(0, liftMotor);
motorControl(70, gripperMotor);
delay(1200);
motorControl(0,gripperMotor);
delay(300);
stage8 = false; stage9 = true;
}
//Reverse back to main line
else if(stage9) {
backwardSlow();
if(EXTrightIRSensorValue == HIGH || EXTleftIRSensorValue == HIGH){
stop();
stage9 = false; stage9b = true;
}
}
//forward until ext ir sensors are off
else if(stage9b) {
forwardSlow();
if(EXTrightIRSensorValue == LOW || EXTleftIRSensorValue == LOW){
stop();
```

```
stage9b = false; stage9c = true;
}
}
//turn left until right ir sensor is on main line
else if(stage9c) {
turnLeft();
if(switch3){
delay(1000);
switch3 = false;
}
else if(rightIRSensorValue == HIGH) {
stop();
stage9c = false; stage10 = true;
}
}
// go until last wall is reached
else if(stage10) {
lineDetect2(rightIRSensorValue, leftIRSensorValue, false);
distanceR = detectDistanceR();
distanceL = detectDistanceL();
if(distanceR <= stoppingDistance2 && distanceL <= stoppingDistance2){
stop();
stage10 = false; stage11 = true;
}
}
else if(stage11) {
stop();
}
}
}


////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////
//////////
////////////////////////////////////////////////////////////////////// FUNCTION
DEFINITIONS ////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////
//////////


void lineDetect(int rightIRSensorValue, int leftIRSensorValue, bool stopTracking1){
```

```
if(stopTracking1 == false){
//If none of the sensors detects black line, then go straight
if (rightIRSensorValue == LOW && leftIRSensorValue == LOW)
{
forward();
}
//If right sensor detects black line, then turn right
else if (rightIRSensorValue == HIGH && leftIRSensorValue == LOW )
{
turnRightLine();
}
//If left sensor detects black line, then turn left
else if (rightIRSensorValue == LOW && leftIRSensorValue == HIGH )
{
turnLeftLine();
}
//If both the sensors detect black line, then stop tracking
else
{
forward();
}
}
}

void lineDetect2(int rightIRSensorValue, int leftIRSensorValue, bool stopTracking1){
if(stopTracking1 == false){
//If none of the sensors detects black line, then go straight
if (rightIRSensorValue == LOW && leftIRSensorValue == LOW)
{
forward2();
}
//If right sensor detects black line, then turn right
else if (rightIRSensorValue == HIGH && leftIRSensorValue == LOW )
{
turnRightLine();
}
//If left sensor detects black line, then turn left
else if (rightIRSensorValue == LOW && leftIRSensorValue == HIGH )
{
turnLeftLine();
}
//If both the sensors detect black line, then stop tracking
else
```

```
{
forward2();
}
}
}


void lineDetectAndAvoidObstacles(int rightIRSensorValue, int leftIRSensorValue, bool
stopTracking2){
if(stopTracking2 == false){
//If none of the sensors detects black line, then go straight
if (rightIRSensorValue == LOW && leftIRSensorValue == LOW)
{
forwardSlow();
}
//If right sensor detects black line, then turn right
else if (rightIRSensorValue == HIGH && leftIRSensorValue == LOW )
{
turnRightLine();
}
//If left sensor detects black line, then turn left
else if (rightIRSensorValue == LOW && leftIRSensorValue == HIGH )
{
turnLeftLine();
}
//If both the sensors detect black line, then stop tracking
else
{
//nothing
}
}
}


float detectDistanceR(){
digitalWrite(trigPinR, LOW);
delayMicroseconds(2);
digitalWrite(trigPinR, HIGH);
delayMicroseconds(10);
digitalWrite(trigPinR, LOW);


durationR = pulseIn(echoPinR, HIGH);
```

```
distanceR = (durationR*.0343)/2;
Serial.print("Distance: ");
Serial.println(distanceR);
delay(0.05);


return distanceR;
}


float detectDistanceL(){
digitalWrite(trigPinL, LOW);
delayMicroseconds(2);
digitalWrite(trigPinL, HIGH);
delayMicroseconds(10);
digitalWrite(trigPinL, LOW);


durationL = pulseIn(echoPinL, HIGH);
distanceL = (durationL*.0343)/2;
Serial.print("Distance: ");
Serial.println(distanceL);
delay(0.05);


return distanceL;
}


int receiveRemoteData1(){
//if(firstButtonPressed == false){
IRremote.DeviceDriverSet_IRrecv_Get(&IRrecv_button);
switch(IRrecv_button)
{
case 6:
//firstButtonPressed = true;
branchNumber1 = 1;
return branchNumber1;
break;
case 7:
//firstButtonPressed = true;
branchNumber1 = 2;
return branchNumber1;
```

```
break;
case 8:
//firstButtonPressed = true;
branchNumber1 = 3;
return branchNumber1;
break;
case 9:
//firstButtonPressed = true;
branchNumber1 = 4;
return branchNumber1;
break;
case 10:
//firstButtonPressed = true;
branchNumber1 = 5;
return branchNumber1;
break;
case 11:
//firstButtonPressed = true;
branchNumber1 = 6;
return branchNumber1;
break;
default:
return 0;
break;
}
//}
}


int receiveRemoteData2(){
//if(secondButtonPressed == false){
IRremote.DeviceDriverSet_IRrecv_Get(&IRrecv_button);
switch(IRrecv_button)
{
case 6:
//secondButtonPressed = true;
branchNumber2 = 1;
return branchNumber2;
break;
case 7:
//secondButtonPressed = true;
branchNumber2 = 2;
return branchNumber2;
```

```
break;
case 8:
//secondButtonPressed = true;
branchNumber2 = 3;
return branchNumber2;
break;
case 9:
//secondButtonPressed = true;
branchNumber2 = 4;
return branchNumber2;
break;
case 10:
//secondButtonPressed = true;
branchNumber2 = 5;
return branchNumber2;
break;
case 11:
//secondButtonPressed = true;
branchNumber2 = 6;
return branchNumber2;
break;
default:
return 0;
break;
}
//}
}


int detectEnterPressed1(){
IRremote.DeviceDriverSet_IRrecv_Get(&IRrecv_button);
switch(IRrecv_button)
{
case 5:
//secondButtonPressed = true;
enterPressed1 = 1;
return enterPressed1;
break;
default:
return 0;
break;
}
}
```

```
int detectProgramShutoff(){
IRremote.DeviceDriverSet_IRrecv_Get(&IRrecv_button);
switch(IRrecv_button)
{
case 1:
//secondButtonPressed = true;
//enterPressed1 = 1;
return 1;
break;
default:
return 0;
break;
}
}


int detectEnterPressed2(){
IRremote.DeviceDriverSet_IRrecv_Get(&IRrecv_button);
switch(IRrecv_button)
{
case 5:
//secondButtonPressed = true;
enterPressed2 = 1;
return enterPressed2;
break;
default:
return 0;
break;
}
}


void forward(){
digitalWrite(motorsEnable, HIGH);
analogWrite(leftWheels, speed1);
analogWrite(rightWheels, speed1);
digitalWrite(leftWheelsDir, HIGH);
digitalWrite(rightWheelsDir, HIGH);
}

void forward2(){
```

```
digitalWrite(motorsEnable, HIGH);
analogWrite(leftWheels, speed5);
analogWrite(rightWheels, speed5);
digitalWrite(leftWheelsDir, HIGH);
digitalWrite(rightWheelsDir, HIGH);
}


void forwardSlow(){
digitalWrite(motorsEnable, HIGH);
analogWrite(leftWheels, speed2);
analogWrite(rightWheels, speed2);
digitalWrite(leftWheelsDir, HIGH);
digitalWrite(rightWheelsDir, HIGH);
}


void turnLeft(){
digitalWrite(motorsEnable, HIGH);
analogWrite(leftWheels, speed4);
analogWrite(rightWheels, speed4);
digitalWrite(leftWheelsDir, LOW);
digitalWrite(rightWheelsDir, HIGH);
}


void turnLeftLine(){
digitalWrite(motorsEnable, HIGH);
analogWrite(leftWheels, speed4 - 20);
analogWrite(rightWheels, speed4);
digitalWrite(leftWheelsDir, LOW);
digitalWrite(rightWheelsDir, HIGH);
}


void turnLeftRightWheelsOnly(){
digitalWrite(motorsEnable, HIGH);
analogWrite(leftWheels, speed2);
analogWrite(rightWheels, speed4);
digitalWrite(leftWheelsDir, LOW);
digitalWrite(rightWheelsDir, HIGH);
}
```

```
void turnLeft2(){
digitalWrite(motorsEnable, HIGH);
analogWrite(leftWheels, 100);
analogWrite(rightWheels, 15);
digitalWrite(leftWheelsDir, LOW);
digitalWrite(rightWheelsDir, LOW);
}


void turnRight(){
digitalWrite(motorsEnable, HIGH);
analogWrite(leftWheels, speed4);
analogWrite(rightWheels, speed4);
digitalWrite(leftWheelsDir, HIGH);
digitalWrite(rightWheelsDir, LOW);
}


void turnRightLine(){
digitalWrite(motorsEnable, HIGH);
analogWrite(leftWheels, speed4);
analogWrite(rightWheels, speed4 - 20);
digitalWrite(leftWheelsDir, HIGH);
digitalWrite(rightWheelsDir, LOW);
}


void turnRightLeftWheelsOnly(){
digitalWrite(motorsEnable, HIGH);
analogWrite(leftWheels, speed4);
analogWrite(rightWheels, speed2);
digitalWrite(leftWheelsDir, HIGH);
digitalWrite(rightWheelsDir, LOW);
}


void turnRight2(){
digitalWrite(motorsEnable, HIGH);
analogWrite(leftWheels, 15);
analogWrite(rightWheels, 100 );
digitalWrite(leftWheelsDir, LOW);
digitalWrite(rightWheelsDir, LOW);
```

```
}


void backward(){
digitalWrite(motorsEnable, HIGH);
analogWrite(leftWheels, speed1);
analogWrite(rightWheels, speed1);
digitalWrite(leftWheelsDir, LOW);
digitalWrite(rightWheelsDir, LOW);
}


void backwardSlow(){
digitalWrite(motorsEnable, HIGH);
analogWrite(leftWheels, speed2);
analogWrite(rightWheels, speed2);
digitalWrite(leftWheelsDir, LOW);
digitalWrite(rightWheelsDir, LOW);
}


void stop(){
digitalWrite(motorsEnable, LOW);
analogWrite(leftWheels, 0);
analogWrite(rightWheels, 0);
digitalWrite(leftWheelsDir, LOW);
digitalWrite(rightWheelsDir, LOW);
}


int motorControl(int value, Servo motor){ // Gripper: + open, - close; Lift: + up, -
down
motor.write(map(value,-100,100,1150,1850));
}
```