ENGR 2620

Integration

Spring 2025

Autonomous Vehicle Project Final Report

Instructor(s): Irvin Jones

Prepared By: Maya Tarnowski
Braidyn Sheffield
Zee Burgos Resendiz
Ashton Taylor

Report Grade: _____

Table of Contents

Table of Figures

Table of Tables

Executive Summary

This report presents the design, development, and successful implementation of an autonomous robotic system created for BioComp, LLC's cleanroom environment. The system was engineered to perform two primary tasks: obstacle avoidance and the pickup, placement, and return of fragile boxes of varying sizes and colors, all while operating within strict spatial, functional, and component constraints.

The final robot integrates several subsystems that work seamlessly together to meet key performance, functional, and additional requirements that were extensively investigated. Key design features include a gripper-mounted pushbutton for box size detection, outsourced color sensors for improved path-following and box color accuracy, and an onboard accelerometer that enables precise 90° turns and enhanced line following. Sensor mounts and aesthetic additions were incorporated to improve both functionality and presentation. Power management was achieved using two buck converters to regulate voltage across components, and a Teensy microcontroller served as the central processor, coordinating all subsystem interactions.

Four conceptual designs were initially proposed and evaluated using a decision matrix. After selecting the optimal concept, risks were assessed early in the process to ensure proper mitigation strategies and avoid technical or scheduling setbacks. A state diagram guided the development of a robust control algorithm, which was implemented in Arduino code to govern system behavior.

Throughout the project, individual subsystems were rigorously tested, leading to a refined and cohesive final system. A mode-switching mechanism was introduced via a pushbutton, allowing the robot to transition between obstacle avoidance and pickup/placement modes. Final demonstrations confirmed that the robot could complete all tasks effectively, reliably, and at a moderate speed, validating the success of the design and the strength of the integrated system architecture.

1. Introduction

BioComp, LLC aims to develop a specialized, compact autonomous robot to operate within the company's cleanroom environment. The robot developed in this project was tasked with reliably performing complex tasks, including picking up fragile boxes of various sizes and colors, navigating color-coded paths to correctly place these boxes on designated platforms, and returning to its starting position without causing any damage. Additionally, the system had to demonstrate the ability to quickly and accurately navigate an obstacle-filled field, avoiding collisions while adhering to strict size, budget, and component constraints. The purpose of the project is to develop a robotic system that meets the company's requirements and excels in dynamic operational scenarios.

This report describes the comprehensive journey of designing, building, and testing the robotic system. It begins with an overview of the project management approach and clearly defines the essential system requirements, including functional and performance criteria. The report then details the user interface and the iterative design process that led to the selection of the final design. Following this, the potential risks related to the system's operation are identified and discussed. The core of the report focuses on the detailed design, covering the mechanical structure, wiring, and software architecture, including an explanation of the robot's operational states such as obstacle avoidance, pickup, and box size detection. Additional special features integrated into the system are also described. Subsequent sections present the results of subsystem testing and a comprehensive performance analysis of the completed robot.

Supporting materials such as parts lists, wiring schematics, CAD drawings, and the implemented code are provided in the Appendices. Together, these sections provide a thorough understanding of the design considerations and development efforts that contributed to the successful creation of BioComp's robotic system.

2. Project Management Plan

The Gantt Chart in Figure 1 outlines the progression of the project, detailing key phases and milestones. Some phases were allotted more time than others due to varying degrees of difficulty and constraint. Each major task is listed in the "To-Do" section with its corresponding timeline, reflecting the relative complexity and constraints of each phase. For instance, the "Final Report" phase was prioritized as a way for the team to take notes through the entire project to ensure consistent documentation and alignment. Some phases, like "Testing," were intentionally scheduled to begin early and include buffer time to accommodate unforeseen challenges, particularly those related to algorithm development. This planning approach helped maintain flexibility and mitigate risks that could affect the overall timeline.

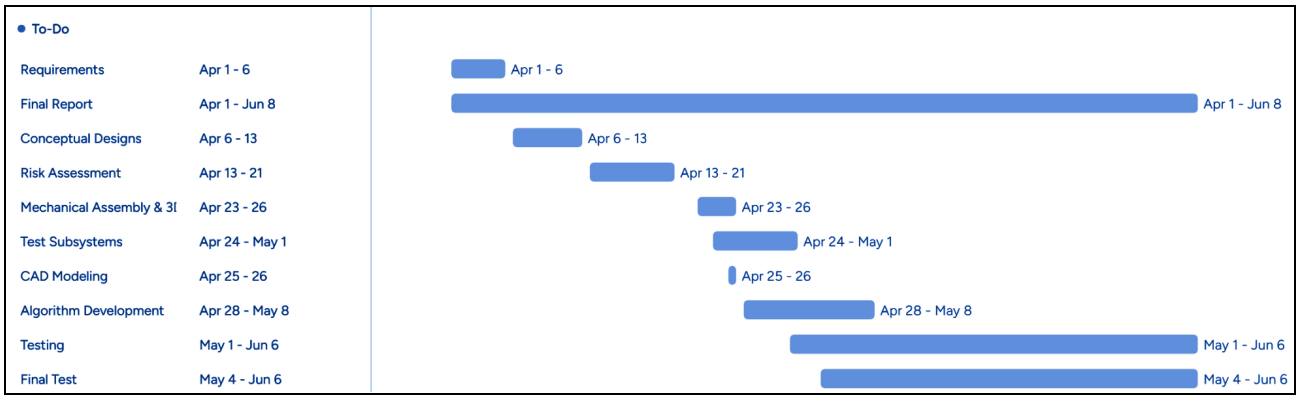

Figure 1. Gantt Chart: Project Timeline Overview

More importantly, a Project Management plan was followed to stay on track. Table 1 shows the detailed plan that was implemented and followed on a week-by-week basis. Outlined in weeks 1-10, the due dates and timeline from Figure 1's Gantt Chart were followed to present a sequence of weekly project updates as well as a modified plan based on what was completed in the prior week.

Table 1: Weekly Project Management Plan

Week #	Dates	Goals / Milestones	Notes *All Thursdays at 2:30 Team Meeting*
1	March 31 – April 6	Finalize design requirements document, pick up lab kits, inventory kits, and test all components in the kit	
2	April 7 – April 13	Create conceptual designs and complete feasibility analysis (via decision matrix). Finalize on one design.	
3	April 14 – April 20	Begin CAD modeling and mechanical assembly. Analyze key risks for the project. Begin assembling base parts for the vehicle. 3D-print necessary parts from CAD model.	Need to finalize design for second layer (3D-printed table) on vehicle that will hold the breadboards and battery in place.
4	April 21 – April 27	Finalize mechanical assembly (3D-printing and component additions). Test drive train of vehicle. Test movement and operation of gripper and gripper arm. Test subsystems, obtain right/left turn times (motor test), find desired arm positions, test all motors.	During team meeting on Thursday: Project Presentation
5	April 28 – May 4	Begin Coding/Testing Process Create State Diagram and Flow Chart Wire Sensor Components	

Week #	Dates	Goals / Milestones	Notes *All Thursdays at 2:30 Team Meeting*
6	May 5 – May 11	Continue Coding/Testing Complete Testing Table	
7	May 12 – May 18	Final Testing Further Algorithm development and subsystem integration	Finalize the system and brainstorm aesthetic (extra credit) additions if time allows.
8	May 19 – May 25	Aesthetic Additions and Final Checks	
9	May 26 – June 1	Task Demonstration	
10	June 2 – June 7 (Finals)	Task Demonstration	Final Report Done

3. System Requirements

For the autonomous robot system design for BioComp, LLC, there were necessary requirements for the system to meet. This section outlines the detailed requirements that guided the development and integration of the system. They are categorized into Functional, Performance, Operational & User, and Non-Functional groups, ensuring a comprehensive understanding of the system's capabilities and constraints. Each requirement is followed by further details and a justification to provide clarity on how the system must meet both customer needs and project expectations. The distinction between features the customer requires versus desires is emphasized (shall vs. should). This section supports the goals of usability, and cost-effectiveness, illustrating the necessary aspects needed to achieve the successful integration and deployment of a robot system that aligns with the project expectations.

3.1 Functional Requirements

3.1.1. Requirement 1

The system shall allow the customer to select and customize the color of the path to be followed. This supports visual clarity and usability, and it also gives users a sense of control and personalization in their interaction with the system. The algorithm allows for multiple color paths to be followed.

3.1.2. Requirement 2

The system shall approach the "pickup" platform. This is essential for completing the "pickup" task, and it ensures that the system functions without human intervention.

3.1.3. Requirement 3

The system shall identify the color and size of the box in the pickup area. This requirement supports the vehicle system correctly recognizing and interacting with the appropriate object, which is critical for accurate pickup and placement of the object (box) being recognized.

3.1.4. Requirement 4

The system shall follow the designated colored path to the correct placement platform based on the color and size of the box. This ensures the box is delivered to the accurate platform according to the customer's specifications.

3.1.5. Requirement 5

The system shall follow a path designated by the customer. This allows for customizable routing based on the colored path selected by the user. This flexibility supports diverse user needs and ensures that the system adapts to different path customizations and operational preferences.

3.1.6. Requirement 6

The system shall return to the initial pickup location after the box has been placed in the correct location. This requirement makes sure that the system will follow the path back to the initial position, also ensuring that the system is ready for the next task cycle. This supports continuous and efficient operation.

3.1.7. Requirement 7

The system shall not damage the box by crushing it or dropping it in any way at any point along the path. This is essential to maintain the quality of the transported item and ensure the system is reliable and safe for handling delicate materials.

3.1.8. Requirement 8

The system shall be able to start from any of the three points during the move with obstacle avoidance operation. This requirement ensures flexibility in deployment and makes sure that the vehicle does not hit any obstacles based on sensor detection.

3.1.9. Requirement 9

The system shall be able to drive through the gap in the walls without hitting them. This requirement makes certain that the vehicle stays within the boundary of the obstacle course, ensuring precise navigation.

3.2 Performance Requirements

3.2.1. Requirement 10

The system should complete the pickup and drop off operation without damaging the box. This includes handling the box during pickup (lifting), transport, and placement to maintain the box's structural integrity.

3.2.2. Requirement 11

The system shall move directly between the left and right Decision Points, following the designated path without deviation. This ensures the robot maintains accuracy in sensor navigation and demonstrates reliable path-following behavior necessary for consistent system performance.

3.2.3. Requirement 12

The system shall perform within the white boundary border and detect when it has reached said border. This makes certain that the robot remains within the defined workspace which is critical for task reliability and compliance with the obstacle course environment constraints.

3.2.4. Requirement 13

The system shall avoid knocking down any obstacles during the obstacle avoidance operation. This requirement ensures precise navigation and control, demonstrating that the system can maneuver through obstacle-filled environments without physically disturbing its surroundings.

3.2.5. Requirement 14

The system shall navigate back to the correct path if a collision occurs. This ensures the system has the capability to recover from unexpected deviations, improving its reliability. This also allows continued operation without manual intervention.

3.2.5. Requirement 15

The system should complete the course with obstacle avoidance operation without knocking over obstacles. It is desired that no obstacles would be impacted during the obstacle avoidance portion. The vehicle should make it through the entire obstacle course avoiding all obstacles. This requirement also reflects the desired level of precision and control of the system. It ensures that the vehicle successfully navigates the full course with effective obstacle detection and maneuvering.

3.3 Operational & User Requirements

3.3.1. Requirement 16

The system shall be fully autonomous once placed in the starting area. This means that no human intervention should be required after activation, aligning with the project's goal of

creating a self-operating vehicle that follows paths, avoids obstacles, and ultimately completes tasks independently.

3.3.2. Requirement 17

The system shall allow the user to place a box of different colors and size in the pickup area. The user can choose which box color/size is picked up. This gives them control over which box is selected for transport, supporting flexibility in testing and showcasing the system's ability to identify and handle the varying box attributes.

3.4 Functional Requirements

3.4.1. Requirement 18

The system shall be only made of the approved/allowed hardware and components. This requirement makes sure that the system maintains quality, safety, and compatibility. It also guarantees compliance with standards and regulations, reducing risks and ensuring reliable performance.

3.4.2. Requirement 19

The system cost shall not exceed the budget of $30. Any extra resources that are purchased outside of allowed hardware and components shall be listed and their cost should be quantified and included in the budget. This requirement supports financial control and accountability by ensuring that the system stays within the allotted budget. It also ensures transparency by requiring any additional resources to be documented and their costs included.

4. CONOPs (Concept of Operations)

The "Concept of Operations" (CONOPs) outline the end-to-end interaction between the user and the robotic system, including system setup, operational procedures, and user decision points. They define how the operator engages with the system to execute a successful delivery task or obstacle avoidance while ensuring the system meets its intended objectives for autonomous pickup, transport, and placement of an object as well as effective obstacle avoidance.

To begin operation, the user uploads the finalized Arduino code to the Teensy 4.1 microcontroller via the Arduino IDE. Once the code is uploaded, the user disconnects the USB connection between the computer and the microcontroller. Two pushbuttons mounted on the vehicle enable user interaction: one for mode selection and the other to start the system. The system features a blue LED indicator that confirms which mode is selected, either obstacle avoidance mode or pickup/placement Mode (see Section 7.5.3 later).

If pickup/placement is selected, the user places a box of their choice (either large or small, and colored red or blue) on the designated pickup platform. The system is programmed to identify the box's size and color. Based on this information, the robot autonomously follows the

corresponding path to deliver the box to the appropriate drop-off location. If obstacle avoidance is selected, the user simply starts the system.

The user is not required to provide further input after initiating the system and selecting the desired mode. All navigation, pickup, and placement tasks are performed autonomously. System status can be visually confirmed through the blue LED and through successful execution of the desired operations.

In terms of support and maintenance, the system is designed to be maintained. Key components (including the gripper, sensors, drivetrain, and power system) are accessible for quick inspection, testing, and replacement if necessary. Algorithmic modifications can be implemented by reconnecting the Teensy to a computer and uploading revised code files.

This entire process enables users to efficiently deploy the robot for repeated tasks, achieving the system's goal of hands-free object delivery and navigation.

5. Conceptual Design

To present the most optimal design for BioComp, LLC, multiple conceptual designs were created and investigated to ultimately arrive at the recommended design solution. This section presents four distinct conceptual designs, each developed by a different team member. Each design outlines the essential functions the autonomous robot system must perform to satisfy the project specifications, including pickup, placement, path following, and obstacle avoidance. Detailed explanations are provided for component placement, system operation, and estimated cost. Guided by the principle that "function begets form," the physical structure and layout of each robot are shaped by the required system behaviors. A decision matrix is then used to evaluate the concepts based on defined constraints, leading to the selection of the most feasible and effective design.

5.1 Design Concept 1

Design 1 utilizes components from the provided lab kit alongside additional material components to attach Ultrasonic Sensors to the vehicle. Shown in Figure 2, this design implements the provided gripper system, grid plate, omni wheels, microcontroller, battery, color sensors, microcontroller, motor controller, servo motors, and additional components (nuts and bolts). Also incorporated into this design are 3D-printed Ultrasonic and Color Sensor mounts which are added to Figure 3. These mounts are provided, but for placing the Ultrasonic Sensors in line with the gripper, two small sheets of Machined Steel would be incorporated to attach the sensors further from the vehicle body and in line with the Gripper, further ensuring that collisions would not happen with obstacles. This aspect of the design is added because the Ultrasonic Sensors can sense up to 50cm ahead of them, but for optimal accuracy, having them farther away from the body could create a more efficient vehicle.

Using Arduino and the Teensy Microcontroller, the system operates with closed-loop control, incorporating feedback from the sensors and gripper system. One important aspect of

this robot system is it follows the designated colored path. In order to fulfill the specifications and requirements to make the robot follow the line without leaving the designated path, an Arduino programming system would be instated with while loops that ensured the vehicle would not stray from the designated path, and if it would, it would resituate itself with “Search Mode” to reach “Follow Mode” (fully aligned path following) once again. It would also follow the requirement of avoiding obstacles in its path (sensed by the Ultrasonic Sensors). An algorithm to simplify the program code development would be executed with a setup, loop, and important functions, subroutines, and device drivers.

One issue with this design is that the internal components are exposed, as shown in Figure 3. This is a challenge because it is more ideal to limit potential user interface with internal components like the motor controller and connecting wires. To mitigate this issue, further in the design process casing could be added to the design if the budget allows, but this design currently lacks additional casing. The total estimated cost of this design is $1 for the use of 1lb of scrap material. This leaves about $29 left in the budget for potential breaks in components or additional casing materials from 3D-printing. All services involved in the production of the vehicle would be free, as the team members will be the ones performing the operations.

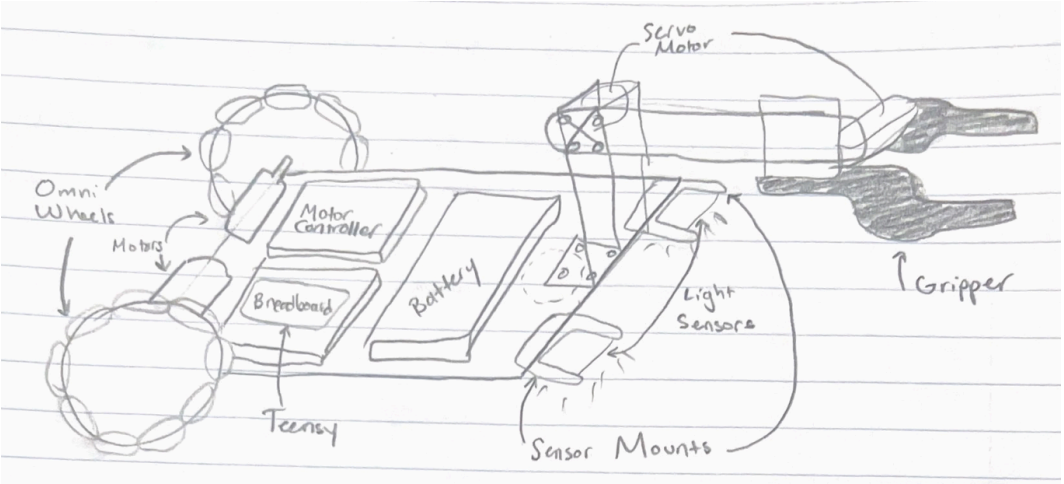

Figure 2: Initial Vehicle and Gripper System

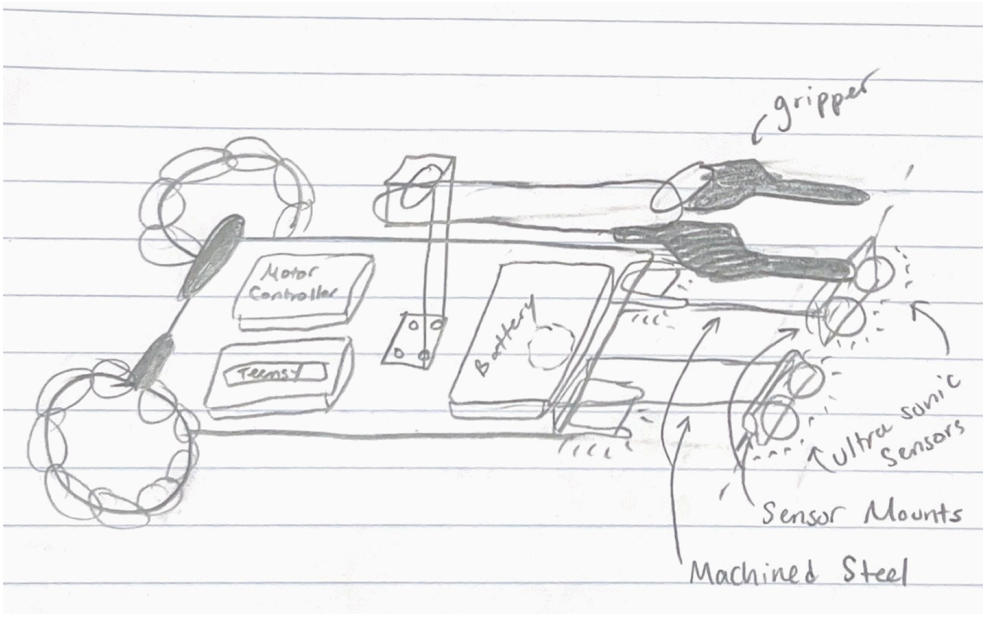

Figure 3: Conceptual Design One Component Placement

5.2 Design Concept 2

This system works with a four wheel drive (4WD) setup that allows the vehicle to move each of the four wheels individually, allowing the vehicle to make tighter turns. This moves the center of rotation from in between the rear wheels to the center of the vehicle. Since the vehicle is 4WD, it requires 2 motor controllers. In this design, the two motor controllers are stacked on top of each other by a CAD model that would be 3D-printed and mounted to the main plate.

The external battery is placed in the rear of the vehicle directly in front of the rear wheel assemblies. A mount would be CAD modeled to ensure that the external battery will not move while the vehicle is operating. To the right of the motor controller, the breadboard would be mounted with the Teensy and the buck converters. Two buck converters are required because the servos on the claw mechanism require 7.4V, and the Teensy requires 5V.

The vehicle is equipped with six different sensors: four color sensors and two ultrasonic sensors. Three of the color sensors are designed to keep the vehicle on the line it needs to follow and the fourth color sensor is designed to detect the color of the block it is moving. The ultrasonic sensors on the front are for obstacle avoidance in front of the vehicle, and the ultrasonic sensor on the right side of the vehicle is designed to detect objects on the right side of the vehicle. The reasoning for the ultrasonic sensor on the right side of the vehicle is based on the projected path for the vehicle during the “move with obstacle avoidance” section of the challenge.

The gripper assembly is mounted directly behind the ultrasonic sensor that is mounted on the front of the vehicle. This allows the vehicle to move to the initial location of the cube and pick it up and place it in front of the fourth color sensor. When the vehicle is in motion, the

gripper assembly is moved to the “up” position so that the center ultrasonic sensor can be operational during movement. The gripper assembly is supplied in the lab kit.

All components and sensors are provided in the lab kit, excluding the external battery and the mounts for the motor controllers that must be 3D-printed. This will bring a small cost from the budget estimated at one dollar. The required jumper wires are also not in the lab kit and will be obtained elsewhere. The physical design outline is pictured in Figure 4.

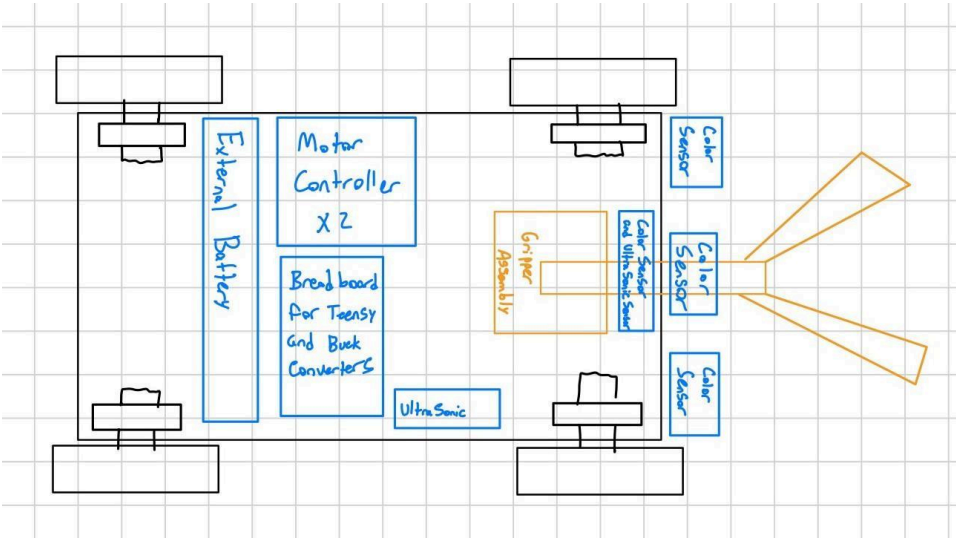

Figure 4: Conceptual Design Two Component Placement

5.3 Design Concept 3

Design three proposes to use components given in the lab kit. This design also proposes making a new 3D-design mount for the light and ultrasonic sensors. As shown in Figure 5, two light sensors are placed at the front-middle ends of the grid plate. A custom mount for the ultrasonic sensor is also presented in this design to be placed in between the light sensors. The ultrasonic sensor should have a mount that allows the sensor to be elevated, with the purpose of using electronic components to give an aesthetic look to the robot simulating ‘eyes’ with this sensor. The estimated cost for the 3D-printed mounts in this design are $10 total; this does not account for prototyping or extra parts that do not fit or work. The gripper arm in this design is proposed to be the gripper that has been used in past class applications with the mounting components provided in the lab kit. This design suggests placing the gripper an inch to two inches behind and laterally from the ultrasonic sensor mount. Proper placing, distancing, and aesthetic design of the sensors is taken into consideration for this robot to ensure functionality and friendly looks of the robot. Placing these sensors is also important for obstacle avoidance and the “search/following” modes of the vehicle for pickup/placement. With 3D-printed mounts and watching the placement of the sensors in the system, this design can be cost effective by using its own electronic components to give proper functionality and aesthetically pleasing looks.

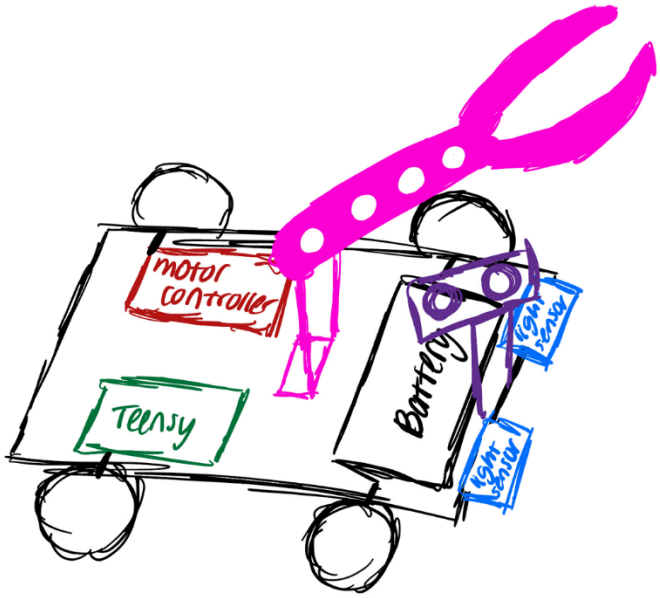

Figure 5: Conceptual Design Three Component Placement

5.4 Design Concept 4

This design iteration shown in Figure 6 builds on the core functionality of previous vehicle prototypes by using both standard and custom components to improve mechanical simplicity. It reuses components from the lab kit while also introducing two new innovations in the drive system and gripper.

The sensor layout is similar to the other designs, with ultrasonic sensors and color sensors positioned at the front of the vehicle. These sensors play a crucial role in environmental awareness, object detection, and navigation. Their placement ensures the best line-of-sight and coverage, which makes path-following and obstacle avoidance reliable.

A robotic arm is included, inspired by previous gripper mechanisms. It has the ability to lift and lower itself as needed for task execution. However, instead of a simple open/close claw, this design features a custom gripper modeled after a crescent wrench. The core of this system is a screw gear that rotates to drive a pinion gear, which in turn opens and closes the jaws of the gripper. This configuration provides controllable and gradual motion, offering improved precision in the speed and position of the gripper's jaws. This is ideal for delicate object manipulation and offers more consistent force distribution than basic clamping designs.

The second major innovation in this design is its two wheel drive (2WD) system powered by a single motor. The motor drives a differential gear, which distributes torque to two rear wheels, reducing mechanical complexity and energy usage. This eliminates the need for dual motors, simplifying wiring and power management. Directional control is achieved through a

servo motor at the front axle, connected to the front wheels by an arm subsystem. This enables steering which mimics car steering for smoother path tracking and turning dynamics.

By combining a screw-drive gripper and a differential-powered 2WD system, this design provides a compact and controllable vehicle suitable for both autonomous navigation and interactive tasks.

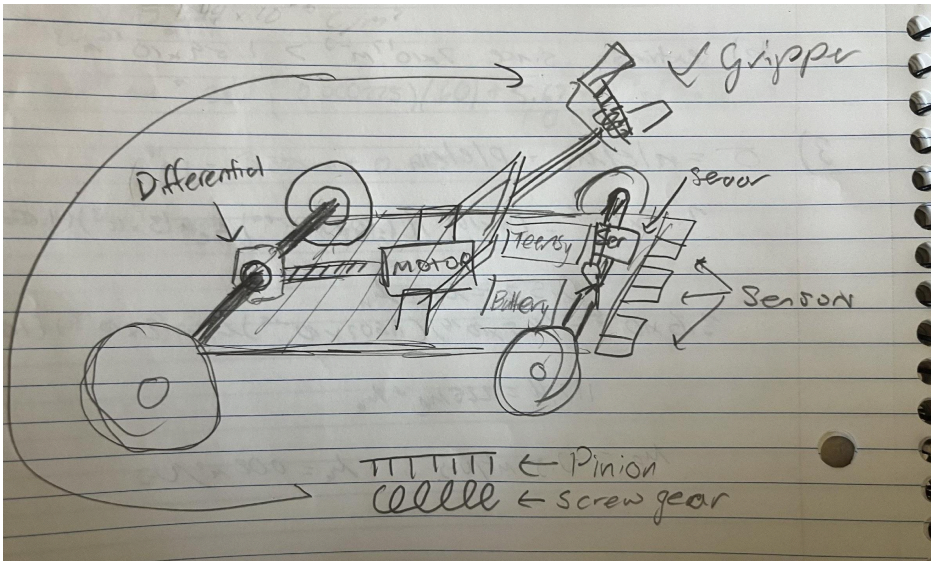

Figure 6: Conceptual Design Four Component Placement

5.5 Feasibility Analysis: Decision Matrix

The Design Matrix in Table 2 is designed to compare multiple conceptual designs and rank each design on a certain amount of criteria that is decided based on the wants/needs of the user. The Design Matrix shows what conceptual design is the best one based on the point value associated with the predetermined criteria. The type of Design Matrix used includes five criteria, and it ranks each of the four conceptual designs from one to five based on how well the design meets the criteria. In this design, one means that it matches the criteria poorly, and five meaning it matches the criteria excellently.

The first criteria that was chosen was the anticipated functionality. This was chosen because it is based on the expected ability that the conceptual design will meet all of the design requirements and therefore successfully perform the tasks required of the vehicle. The next criteria is the estimated cost of the design. Some of the designs rely on machining parts for it or making a CAD model for it and 3D-printing it. A five on the scale shows that the design requires no extra parts that are not included in the kit, and a one on the scale shows that a majority of the budget would go to modifying or buying new parts that are not in the supplied kit. The third criteria is the ease to make. This criterion is based on the difficulty of the design and if things need to be created to meet the designs requirements. The final criteria is based on the internal components being visible to the user. This entails any component that the user is not required to interact with, should be hidden.

Rating Scale: 1 to 5 scale with 1 = Poor, 2 = Fair, 3 = Acceptable, 4 = Good, to 5 = Excellent

Table 2: Design Matrix

Criteria Product	Anticipated Functionality	Small Estimated Cost	Ease to make and operate	Looks (Aesthetic)	Hidden Internal Features	Score
Conceptual Design 1	(Should fully function, meeting all functionality requirements) **5**	(Extra cost for metal addition: $1, low cost) **4**	(Many parts already provided, very easy) **4**	(Internal features displayed) **2**	(Very little casing, internal features displayed) **1**	**16**
Conceptual Design 2	(Should fully function, meeting all functionality requirements) **5**	(No additional cost) **5**	(All parts provided) **5**	(Internal features displayed) **2**	(Internal features displayed) **1**	**18**
Conceptual Design 3	(Should fully function, meeting all functionality requirements) **5**	(Extra cost for 3D-printed parts: low cost) **5**	(Many parts already provided, very easy) **4**	(Internal features displayed) **2**	(Internal features displayed) **1**	**17**
Conceptual Design 4	(Should fully function, meeting all functionality requirements)**5**	(Extra cost for differential and gripper (est. $10) **2**	(Extra materials will increase build complexity) **3**	(Internal features displayed) **2**	(Internal features displayed) **1**	**13**

5.6 Final Design Recommendation

Conceptual Design 2 (detailed and described in Section 5.2) was recommended from the design matrix as a justification (Table 2). This design was chosen due to its highest score on the design matrix of 18 as compared to other scores of 16, 17, and 13. This design is easier to create due to its included component accessibility. Without adding outsourced components to the design, all the components have been provided in the lab kits. This makes the design inexpensive to create, leaving room in the budget for possible broken parts or creative aesthetic additions

later on. Some aspects about this design that are preferable as well is its use of 4 disk wheels rather than using a castor wheel in addition to the increased reliability and data from using 4 color sensors rather than two. With design 2 having four disk wheels and a 4WD setup, this will allow for ease of programming and controlling the vehicle during the pick up and place part of the challenge and the obstacle avoidance part of the challenge. To conclude, design 2 was chosen to create due to its low cost and anticipated functionality.

6. System Risk Management

In developing the vehicle system, effective risk management was essential for ensuring reliability and performance. The system includes a complex integration of sensors, electronics, mechanical components, and programming logic, all of which must function in harmony to achieve the project's objectives, satisfying the requirements. For proper risk management, both technical and programmatic risks were evaluated and analyzed for likelihood and impact. Corresponding mitigation strategies were outlined to proactively reduce the chance of failure or disruption, and these tactics were incorporated into the Project Management Plan. The evaluation of risks was useful in maintaining system stability, staying within scope and budget, and delivering a functioning robot within the timeframe.

6.1. Risk 01

6.1.1. Risk 01 Description

Risk 1 details possible Sensor Failure and Calibration Challenges. This is a technical risk, where the Color or Ultrasonic sensors could be miscalibrated, leading to poor line following or failed obstacle avoidance. The likelihood of this being an issue was medium, but there could also be a high impact on the entire system. Inaccurate readings could result in the robot misinterpreting the path or failing to respond appropriately to obstacles. This would critically impair the robot's ability to complete its objectives and may require significant debugging time. Failure in sensor performance could cascade into other systems, such as movement and object pickup, making this a high-impact risk.

6.1.2. Risk 01 Mitigation

To mitigate this risk, each sensor was tested individually before integration into the project. Additionally, implementing real-time printing on the Serial Monitor in Arduino about what the program is doing contributes to debugging fixes, so this was done during calibration. Also, fallback behaviors could be implemented into the code to help mitigate this risk. For example, in the code, if the sensor data is too noisy, the vehicle could stop, recalibrate, and reassess before restarting.

6.2. Risk 02

6.2.1. Risk 02 Description

Risk 2 describes the integration of multiple subsystems together to create one large system. The integration complexity, being combining 4WD, controllers/motors, sensors, mechanical building pieces, and the gripper, is a developmental technical risk. There could be difficulty involved in integrating multiple subsystems that could lead to system errors or delays. This risk could also lead to component failure when fully integrated. The likelihood of this risk was high, with a high impact. Poor integration could cause cascading failures, inconsistent performance, or component damage. Time lost during troubleshooting could delay the overall timeline.

6.2.2. Risk 02 Mitigation

To avoid this risk delaying or harming the system integration, subsystems were tested incrementally before the full integration. When debugging issues arose, each subsystem could be tested again with specific code used particularly for it, and the subsystem could also be tested as a whole, and its own subsystem components could be tested. For example, the gripper movement could be tested by separating the subsystem into even more subsystems, testing the servo horns, the gripper, and the gripper arm. In the project timeline, integration-specific checkpoints were assigned, making sure the team was held accountable for keeping up with the project timeline.

6.3. Risk 03

6.3.1. Risk 03 Description

Risk 3 refers to a programmatic scheduling risk. This risk regards the timeline provided for the project to be fully completed. A risk that could be encountered is scheduling delays and time constraints. If there was a lack of time for sufficient testing due to delayed 3D-printing or component/subsystem setup, there could be a significant impact on the project, with it not being completed by testing day. Additional personal emergencies could prevent the team from meeting at ser times as well, which would impact the overall timeline of the project. This programmatic risk stems from the limited timeframe available for completing a complex integration project. Delays in earlier phases—such as 3D-printing parts, hardware setup, or code debugging—could compress the available time for full system testing and fine-tuning. If critical bugs emerged close to the due date, there could be insufficient time to resolve them, jeopardizing overall project success. The likelihood that time constraints and scheduling delays would impact the project timeline was high, but mitigation and a plan of action for situations like this prevented extreme detrimental impacts on the project.

6.3.2. Risk 03 Mitigation

To prevent Risk 3 from having a serious impact on the project, strategies of mitigation were employed. Testing began as early as possible to avoid procrastination of any kind. The

timeline that was created was designed with buffer time, allowing space for scheduling delays and important due dates. With a designated timeline in play, additional contingency plans could be created if the schedule was “down to the wire”, and sacrifices needed to be made. For example, if need be, downgrades from extra credit implementation could be put into place to prioritize having a working system over a semi-working system with cool extra credit aspects.

6.4. Risk 04

6.4.1. Risk 04 Description

Risk 4 entails possible power distribution and electrical failures. With two buck converters (providing different voltage levels 7.4V for servos, 5V for the Teensy), an external battery, and other electrical components being used, electrical instability could occur, which could possibly damage the components used. This would harm the components used as well as the budget. Incorrect wiring or overloads could damage sensitive components or result in inconsistent performance. Electrical issues could impact any part of the system, from sensor readings to motor power, making this a high-impact risk. The likelihood of this happening was medium, but the impact would be high. If properly prevented, electrical damages were avoidable.

6.4.2. Risk 04 Mitigation

To mitigate this risk, the voltage compatibility was checked by all team members separately to ensure that no power related mistakes were made. Using a multimeter, the voltage was checked, and the wiring was also checked for each component, testing power delivery isolation before attaching the key components.

6.5. Risk 05

6.5.1. Risk 05 Description

The last programmatic risk, Risk 5, refers to unexpected costs that could arise throughout the process due to broken components, added (3D-printed or machined) features, and even aesthetic extra credit additions. If large numbers of unexpected costs were to emerge, the total cost of the project would be impacted, possibly leading the project to surpass the budget, therefore making the system too costly. This risk was low, but the impact would be medium, as it depends on how many additions were integrated into the system and on how much those additions cost.

6.5.1. Risk 05 Mitigation

To prevent this risk from damaging the project exorbitantly, the budget of $30 was closely followed, with significant buffer room for additional costs. The chosen design had a small projected budget, with many components having been provided by the lab kit. This means that there was room in the budget if parts were to accidentally fail or if 3D-printed sensor mounts needed to be created iteratively. Also, if ultimately the system was completed with extra time to

make it aesthetically unique, the remaining budget could be used to make aesthetic casing and fun additions for the system.

6.6. Risk Assessment

The most imminent risks to the project were determined to be Risk 01: Sensor Failure and Calibration and Risk 02: Subsystem Integration Complexity—both of which carried high impacts and moderate-to-high likelihoods. These technical challenges could significantly hinder the robot's functionality if not addressed early and thoroughly. Risk 03 (Scheduling Delays) also posed a high likelihood, particularly due to the time-sensitive nature of hardware testing and integration. Electrical issues (Risk 04) were less likely but still posed a high-impact threat to component functionality and budget. Risk 05 (Unexpected Costs) was the least critical, but it was important to monitor through conservative budgeting.

To visualize this evaluation, a Risk Assessment Matrix (Figure 7) maps each risk by likelihood and severity. This matrix helped prioritize mitigation efforts; sensor accuracy and system integration are addressed with early testing and modular debugging, while schedule flexibility and careful budget tracking address time and cost-related risks. This risk assessment was critical for prioritizing certain mitigation techniques depending on risk probability and impact.

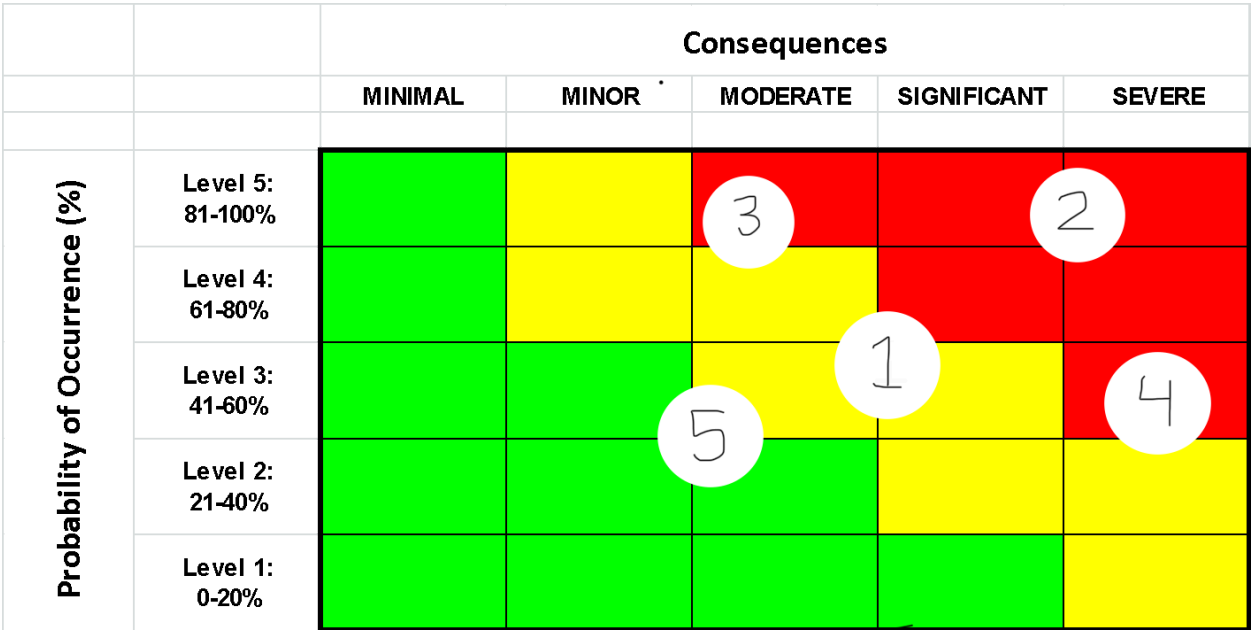

Figure 7: Risk Assessment Graph

7. Detailed Design

The design of the system integrates mechanical, electrical, and software components into a cohesive, task-oriented vehicle capable of navigating complex environments, identifying and sorting objects, and responding to real-time sensor data. This section provides a breakdown of

the system's design, including structural decomposition of hardware subsystems, CAD and 3D-printed component details, electronic details, wiring diagrams, and software architecture. Each element was developed and tested to ensure compatibility, safety, and functionality within the full system. Special features, such as an emergency power cutoff, custom sensor mounts, and an onboard accelerometer, demonstrate design considerations that go beyond the project's basic requirements to enhance performance, organization, and reliability.

7.1 Structural Decomposition

7.1.1 Mechanical Assembly

Shown in Appendix A, the physical parts that were used to assemble the final vehicle design are described by amount and their corresponding details. The important subsystems require mechanical assemblies to effectively run and complete the requirements. The final physical robot design is shown in Figure 8.

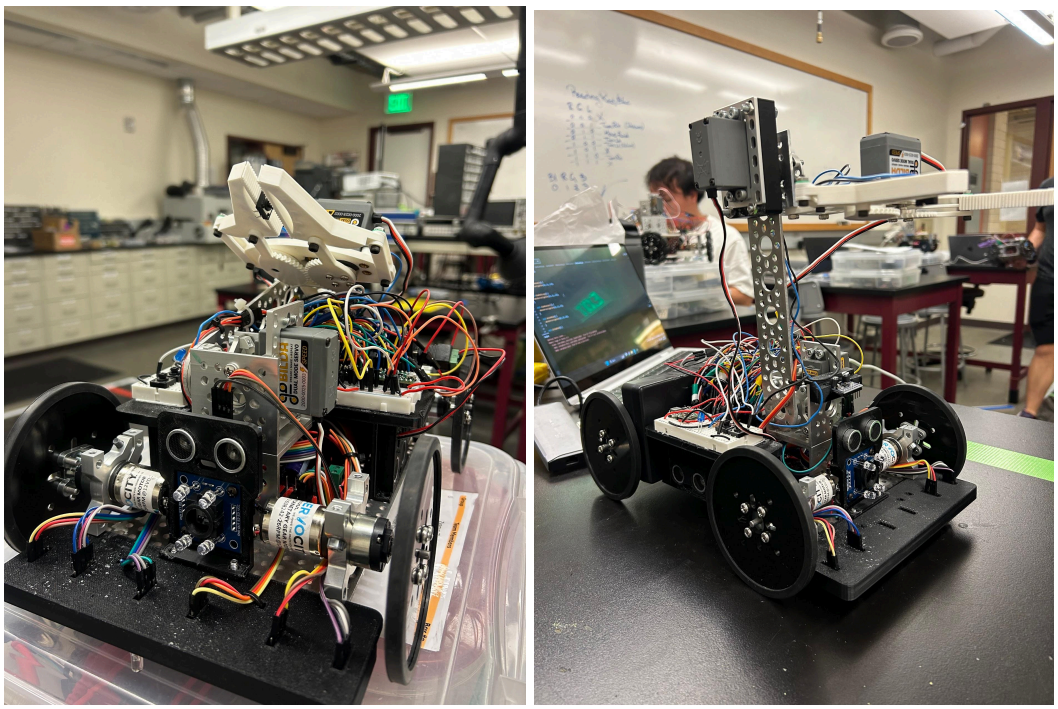

Figure 8: Finished Robot at Rest (Left) and in Use (Right)

7.1.2 Breadboard Table

A breadboard table was created to encase the motor controllers and excess wiring. This table (Figure 9) holds the two breadboards on a separate platform, creating more space on the vehicle for the wiring. Behind the table, there is a space for the battery to rest (in between the rear wheel motors and the breadboard table). This is important so that the battery does not fall off the vehicle when it moves.

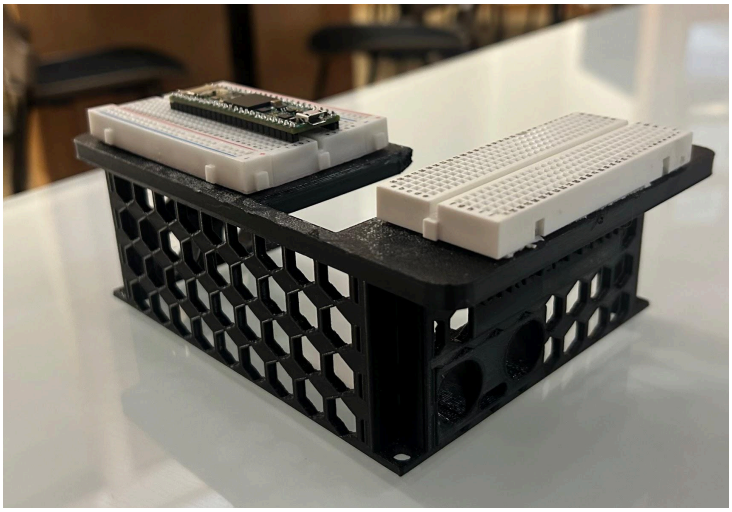

Figure 9: Casing Table

7.1.3 Driving System

The vehicle is a 4W drive vehicle, using four disc wheels and four 26 RPM servo motors. This aspect of the design was critical for creating a vehicle that turns effectively. With four wheels, the vehicle turns with more precision and direction than it might with two wheels and a castor wheel for example.

7.1.4 Gripper and Arm Assembly

The gripper-arm assembly consists of metal arm components and a gripper, forming a double-jointed mechanical subsystem with two axes of rotation. As detailed in their respective sections of the parts list of Appendix A, this configuration enables a wide range of motion, allowing the arm to meet the functional task requirements of detecting the box color, transporting the box to the drop-off platform, and placing it down. The dual-joint design was essential for achieving the flexibility and precision required for each stage of box handling specifically. The parts used for the gripper specifically can be seen in Appendix B (Figure 27).

7.2 Auto Desk Inventor

The 3D-printed components used in the system were designed using Auto Desk Inventor. Technical drawings and figures of these elements are included in Appendix B (Figures 28-38). Precise measurements of key components (such as sensors, structural mounts, and arm spacers) were critical to ensure proper fit and functionality. STL files of commercial components were integrated into an Inventor assembly design to enable accurate modeling and alignment. These files were also important in verifying the sensor dimensions that were used in the creation of the 3D-printed mounts. To avoid material waste, ensure cost-efficiency, a virtual assembly was crafted using both the acquired STL files and custom-designed parts. This allowed for verification of dimensions, alignment, and fit before committing to physical prints. This

pre-validation step was essential to confirm compatibility and avoid printing errors, supporting both the mechanical assembly and budget constraints. The assembly that modeled the design is shown in Figure 10.

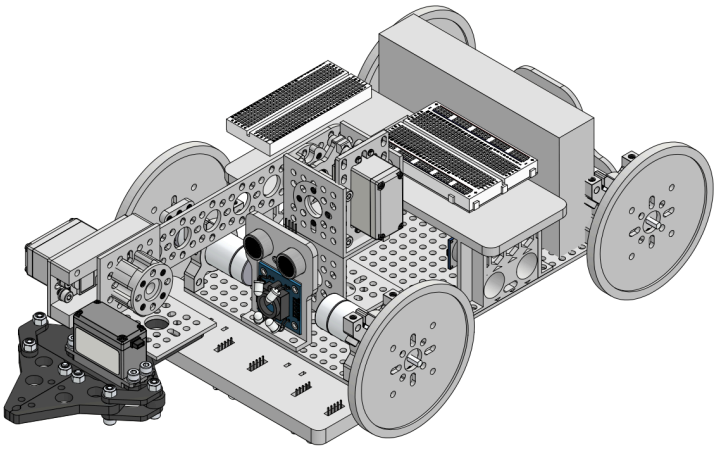

Figure 10: CAD Assembly

7.3 Circuit Schematics

Certain circuit schematics were followed for the accurate operation of the components connected to the microcontroller. The motor controller pins were connected to their respective Teensy pins. The same was true for the motors and sensors. These schematics are detailed in a KiCAD schematic file. All components were added to KiCAD such that a clean professional schematic can be easily followed when wiring the vehicle. The schematic can be found in Appendix C.

7.4 Wiring

In Microsoft Excel, a wiring table was created and used throughout the system development and operation to document and manage all electrical connections. The table detailed the assigned Teensy 4.1 microcontroller pins, corresponding wire colors, and the specific component connections. It also recorded wire-to-wire extensions used for added length, including color pairings (e.g., red-yellow), ensuring traceability for each segment.

This documentation was essential for efficient troubleshooting and reassembly. In the event of a disconnection or error, the table allowed for quick identification and accurate reconnection, minimizing wasted time spent on locating the missing connection. By clearly mapping out the system's electrical layout, the risk of miswiring or power failures was significantly reduced (see Section 6.4). The complete wiring reference table is provided in Appendix C, Table 8.

7.5 Software Architecture

The software architecture was designed to manage the robot's core functionalities through a structured, state-based control system. The development process began with algorithm planning, where a state diagram was created to map out the robot's decision-making and transitions between key behaviors. This laid the foundation for the programming logic implemented in the final system. Using the Arduino IDE and a Teensy 4.1 microcontroller, the algorithm was translated into modular code that governs the robot's performance across all operational states, from obstacle avoidance to pickup and placement tasks.

7.5.1 Algorithm Development

To guide the development of the final state algorithm, a state diagram in the form of a flowchart was created (Figure 11). This diagram outlines the major system states, labeled by the task that the state will be in charge of doing, and the transitions between them using directional arrows. The process begins at the initial state, labeled as IDLE_STATE. From there, the system enters one of two functional modes: obstacle avoidance (Mode 1) or pickup/placement (Mode 2) based on how the "start button" is pressed. In obstacle avoidance mode, the vehicle moves forward, and the ultrasonic sensors determine if a wall is detected. Depending on which sensor is triggered, the vehicle turns accordingly to avoid the obstacle. Once the finish line is reached, the system transitions back to the initial state (IDLE_STATE). In pickup/placement mode, the system identifies attributes of the box (size and color) during pickup using a front color sensor and pushbutton. Based on the detected color, the robot follows the corresponding colored line to deliver the box to the correct drop-off spot. After placement, the robot returns to the green line and navigates back to the initial state (IDLE_STATE), ready to begin the next task cycle. To summarize this, the state diagram was how the algorithm was developed to ultimately accomplish the functional and performance requirements of the system.

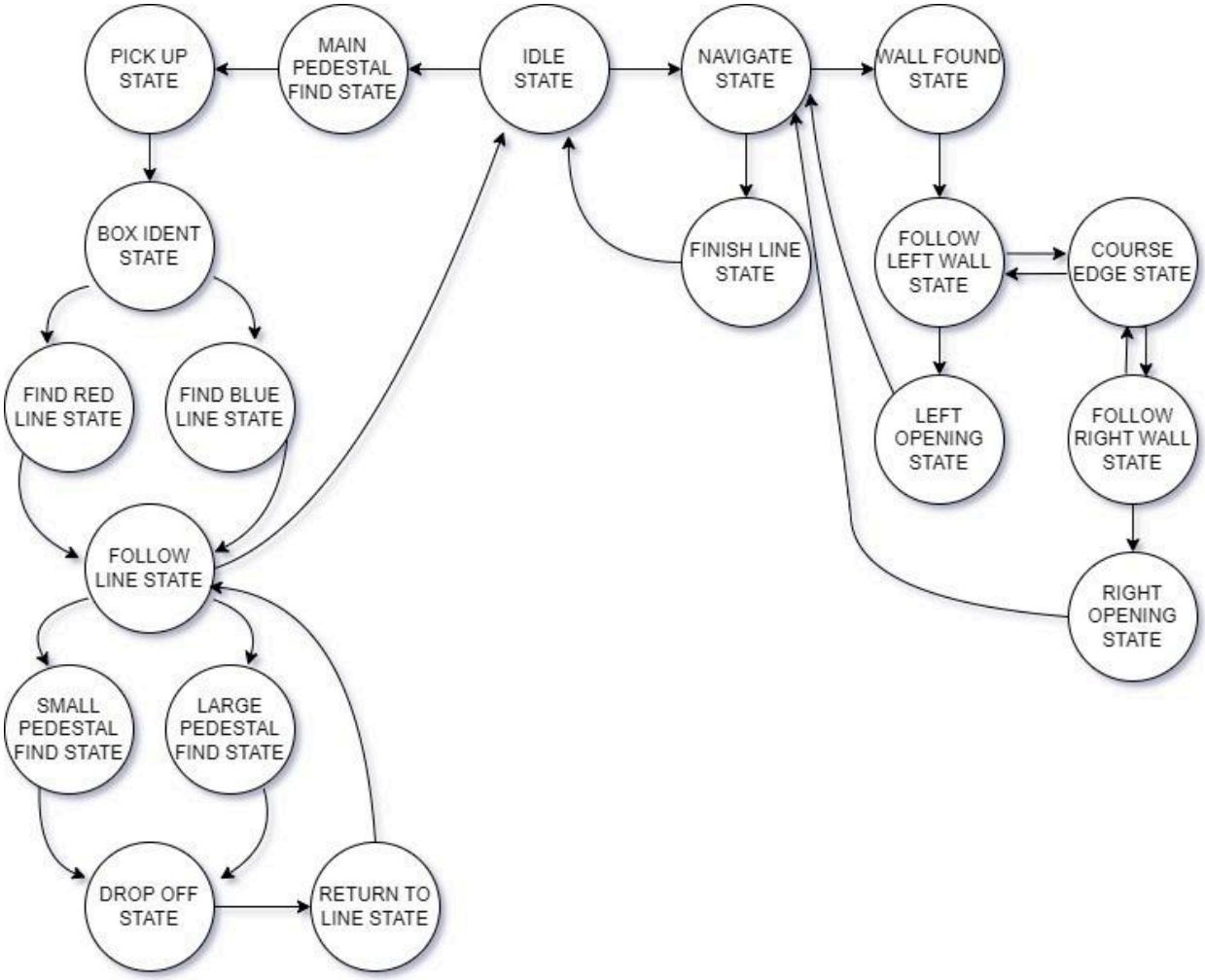

Figure 11: Initial State Diagram

7.5.2 Program Code

The system was programmed using the Arduino IDE in conjunction with the Teensy 4.1 microcontroller. Functions were written in separate files and integrated into the main program. The main code file consisted of the state machine and control initialization. Then each electrical subsystem has a respective code file that handles the control functions of that specific sensor, as well as a header file to tie the sensor code files to the main code file. The complete code, provided in Appendix E (Figures 41-51), was developed based on state diagrams and predefined algorithms to ensure reliable operation across all system states.

From environmental sensing to mechanical manipulation and navigation, the system is composed of coordinated subsystems that work in sequence to accomplish complex tasks. These subsystems were guided by the logic outlined in Figure 11, which maps the robot's decision-making process and task flow. This flow is outlined in the subsequent sections.

7.5.3 Mode Indicator

On the smaller breadboard, displayed in Figure 12, a blue LED and two pushbuttons are used for mode switching and indication. When the system enters Mode 1 (object avoidance), the light is off, and when the system enters Mode 2 (pickup and placement), the light is on. To switch between modes, the button closest to the rear is pressed, and to start the actual system, the front button is pressed.

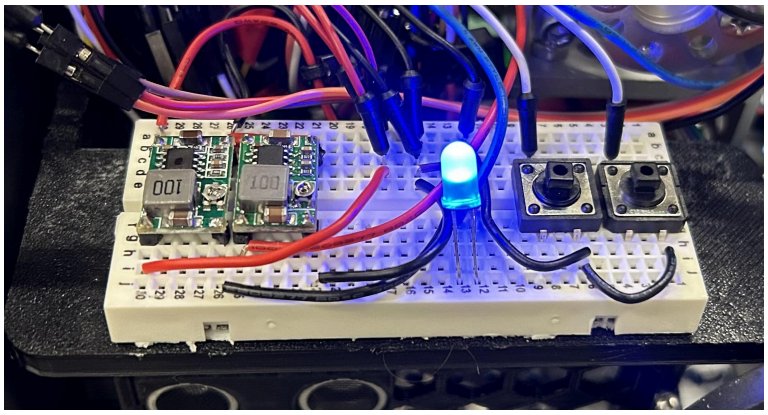

Figure 12: Mode Indicating Subsystem

7.5.4 Pickup

Using the front ultrasonic sensor, the vehicle senses when the pickup platform is a gripper arm distance away (about 14 inches from the center of the vehicle to the center of the box). Once it senses the platform in front of it, the gripper moves to its “fully open” position (Figure 13). Then, the gripper arm tilts outwards, stopping at the location of the box on the platform. The gripper begins to close and it then detects the box size.

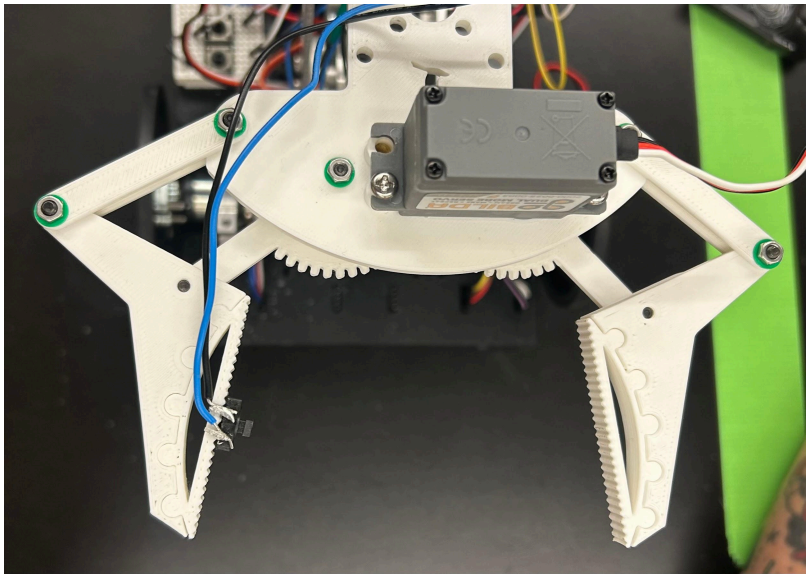

Figure 13: Gripper Fully Opened Position

7.5.5 Box Size Detection

To detect the size of the box, a third pushbutton was used (Figure 14). It was attached to the inside of the gripper with soldering and longer wires. When the button is pressed by the box it picks up, it stops squeezing, therefore detecting the box size. The two box sizes are shown in Figure 15, with the left box being the large size and the right box the small size.

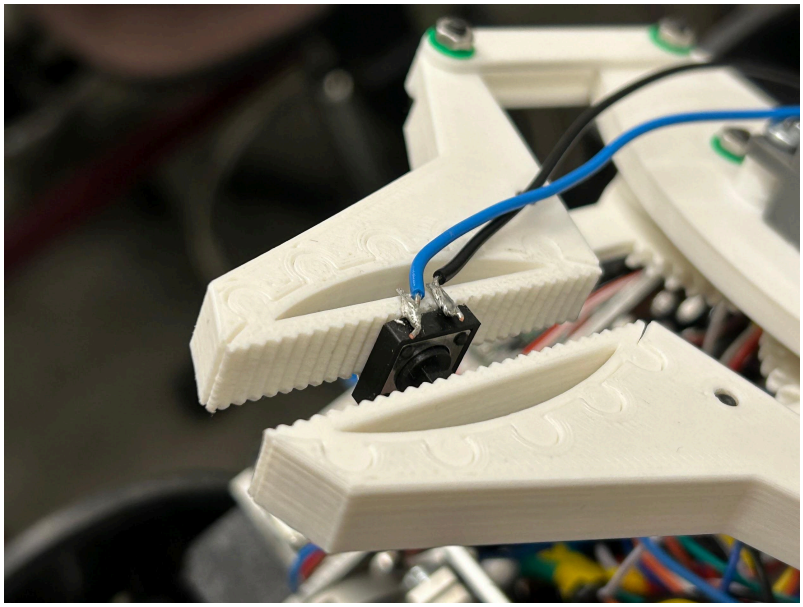

Figure 14: Pushbutton-Gripper Attachment

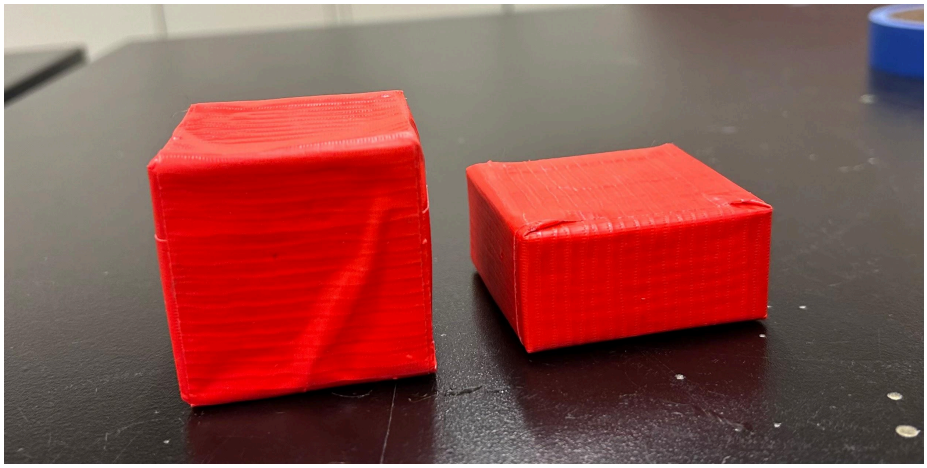

Figure 15: Box Sizes (Large and Small)

7.5.6 Box Color Detection

Once the box is picked up, the vehicle moves backwards until the green line on the course is detected to avoid contact with the platform on the next move. The gripper tilts towards the front color sensor, and this sensor reads the detected color, blue or red. The two different box colors that can be detected are red or blue, and this color detection prompts the robot to follow the corresponding colored path back to the green line. Figure 16 shows the small box size and its different colors for example.

Figure 16: Box Colors (Red and Blue)

7.5.7 Line Following

Depending on which box color is detected, the robot moves in the corresponding direction (see Figure 17 course map). If the box is red, the robot turns right and follows the green line until it hits the red line with the right color sensor. Likewise, if the box is blue, the vehicle

turns, and it follows the green line until the left color sensor detects the line. When the line is detected, the vehicle moves forward slightly, adjusting itself to align its center with the center of the line. Then, it makes the respective 90° turn necessary to follow the line to the pickup platform (right for red and left for blue). Here, the robot follows the colored path, making 5° turns (determined with the accelerometer) necessary to readjust the robot's direction to follow the line if the path color is detected on either side. When the robot reaches the fork in the path, at the same time, the color sensors both detect the colored line, prompting the robot to make the necessary right or left 90° turn depending on the box size (determined with the gripper pushbutton, see Section 7.5.5). If the box size is small, the robot turns right, and then turns left when the path's sharp turn is detected with the color sensors. If the box size is large, the robot turns left, follows the path, and then makes the sharp right turn when the color sensors detect it. From here, in both cases, the system moves forward, switching into the "placement" mode.

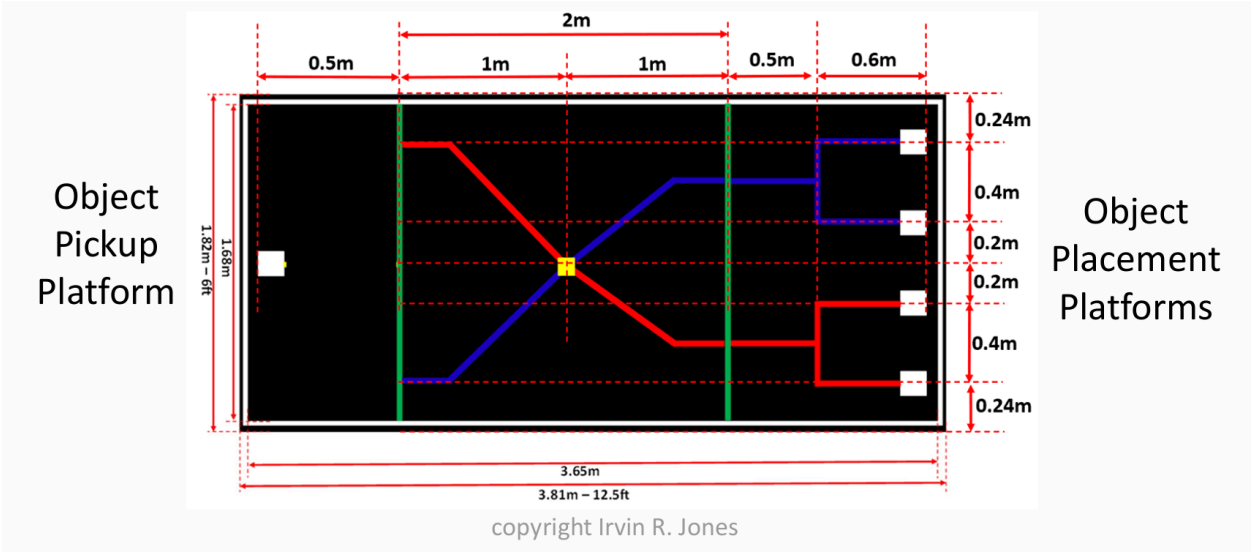

Figure 17: Pickup/Placement Course Layout

7.5.8 Placement

In "placement" mode, when the front ultrasonic sensor detects the platform about 14 inches away, the robot stops. Then, the arm extends, and the gripper releases the box on the platform. Next, the vehicle moves backward until the left or right color sensor detects the colored line. The robot makes a 90° turn towards that line, and moves forward again, repeating the 90° turn if the line is detected again. The vehicle continues to move forward, and it senses the green line, but continues on its path, following the colored path it is prompted to (via box color earlier detected). When the vehicle reaches the second green line, it stops, finally completing the Pickup-Placement-and-Return task. At this point, the vehicle moves back into "idle mode."

7.5.9 Obstacle Avoidance

The "Obstacle Avoidance" mode was essential to meet the functional and performance requirements of the robot. As illustrated in Figure 18, the robot navigates a predefined obstacle course (similar to the example in the figure) using a state-based control system. Upon activation of the mode from the designated pushbutton, the robot transitions into the initial “Navigate” state. In this state, the vehicle moves forward slowly.

When the front ultrasonic sensor detects an obstacle wall within the predefined distance, the robot turns 90° left, transitioning into the “Follow Right Wall” state. In this state, the robot proceeds forward, and if the right ultrasonic sensor no longer detects the wall, or if the front color sensors identify the boundary-marking white tape, the system transitions into the “Course Edge” state. Here, the robot performs a 180° turn and enters the “Follow Left Wall” state, mirroring the behavior of the previous state but with orientation flipped to the opposite wall.

However, when the side ultrasonic sensors detect the end of a wall, the robot determines which side has opened up and transitions into either the “Right Opening” or “Left Opening” state. In these states, the vehicle executes the appropriate 90° turn and re-enters the “Navigate” state to continue forward movement. This sequence repeats until the front color sensors detect a green line, signaling the “Finish Line” state and the end of the course.

To ensure consistent and accurate directional changes to both the left and right, an accelerometer was integrated into the system (see Section 7.8.6). This replaced previously fixed time-based estimates (e.g., 3.5 and 7 seconds) with real-time inertial data. The accelerometer allowed the robot to reliably determine when a turn was complete, minimizing the risk of under- or over-rotation, which could negatively impact the robot’s alignment and subsequent pathfinding accuracy.

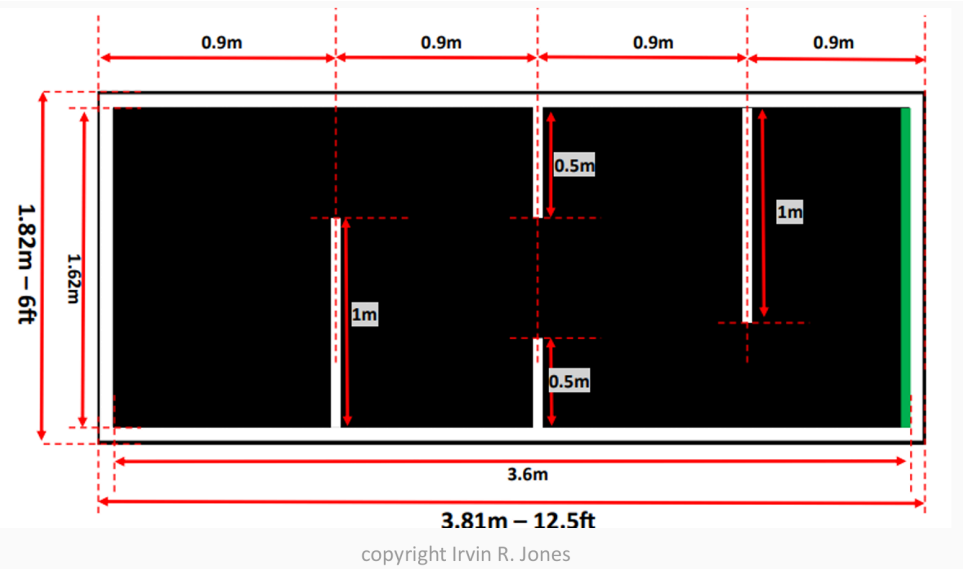

Figure 18: Obstacle Avoidance Course Layout (Example)

7.6 Technologies Employed

7.6.1 Ultrasonic Sensors

Three ultrasonic sensors were implemented in the system to measure distance by emitting high-frequency sound waves and measuring the time it takes for the echo to return (MaxBotix). One front-facing sensor was used to detect obstacles or pickup platforms, while the left and right sensors were used during obstacle avoidance to follow walls and detect openings. These sensors were critical for enabling real-time navigation and precise stopping distances in both the Obstacle Avoidance and Pickup-Place-and-Return tasks.

7.6.2 Color Sensors

Color sensors were used to detect and differentiate colored lines and box colors throughout the robot's navigation and sorting initiatives. These sensors function by emitting light and analyzing the reflected wavelengths to determine the surface color beneath them (ROHM). Mounted at the front of the robot and on the gripper, they were essential for path-following behaviors and for identifying whether a box was red or blue, informing the correct placement path to follow.

7.6.3 Motor Controllers

Motor controllers were used to regulate the power sent to the DC motors responsible for the robot's 4WD system. These components interpret signals from the microcontroller and translate them into appropriate voltage and current outputs to drive the high-torque motors. The system required precise speed and direction control for turns and navigation. The motor controllers enabled forward and reverse motion which is essential for accurate maneuvering in tasks like obstacle avoidance and line-following.

7.6.4 Servo Motors

Three servo motors were used for the arm system to handle object interaction tasks. Two high-torque servos powered the rotational joints of the arm, providing the strength needed to lift and position the payload accurately. A third servo (speed) was used to open and close the gripper. Unlike continuous motors, servos allow for precise control over position, which made them ideal for executing arm movements and securing boxes without damaging them. This combination of torque and speed servos ensured stable lifting and functional gripping.

7.7 Standards Used

While no official industry standards (such as ISO or IEEE) were formally required for the vehicle system, several basic design and engineering practices were followed to ensure

reliability, organization, and safety throughout the system. Standard wiring conventions were used to keep power and signal lines clearly separated and color-coded, which helped with debugging and safe operation. Additionally, the wiring for the motor systems was grouped and secured using zip ties, helping to keep cables organized and reducing the risk of loose or tangled wires interfering with moving parts. Much of this wiring was also enclosed beneath the breadboard table to keep these elements of the system protected. Also included in Section 7.8.4, a safety power cable was incorporated for additional risk management surrounding power-related risks (see Section 6.4 Risk 04).

Datasheet specifications for all components, such as the Teensy 4.1 microcontroller, ultrasonic sensors, motors, and motor controllers, were created and followed to ensure safe operation within rated voltage and current limits (12V battery). Because the system is powered by a 12V battery, two buck converters are included to step down the voltage to appropriate levels for various components. This allowed the microcontroller and sensors to operate within their lower voltage requirements (e.g., 5V and 3.3V), while isolating power distribution between high-current and sensitive low-power subsystems. The Teensy pin diagram is provided in Appendix C, Figure 39.

On the software side, specific programming practices were applied to organize the code into clear functions and state machines (Appendix E). This approach made the algorithmic system easier to develop, troubleshoot, and maintain. These practices contributed to a safe and organized final design.

7.8 Special Features

In addition to the core components used in the physical assembly, several supplementary elements were integrated to enhance the design's overall functionality, performance, organization, safety, and aesthetics. This section outlines these enhancements, explaining their purpose, significance, and contribution to the system's effectiveness.

7.8.1. Team Name Extrusion

Shown in Figure 19, for company branding, the team name of "Ctrl Alt Elite" was extruded in Inventor on the side of the breadboard-holding table (see Section 7.1.2). This component was 3D-printed with a honeycomb-like pattern on the sides in addition to the team name for two main reasons. First, this casing was meant to hide the internal motor controller features that the user does not interact with. Second, these additions were used not just for practicality, but for aesthetic. The addition of the team name on the side of the robot promotes its individuality and uniqueness, giving it an aesthetically pleasing look that also reinforces the team name and brand.

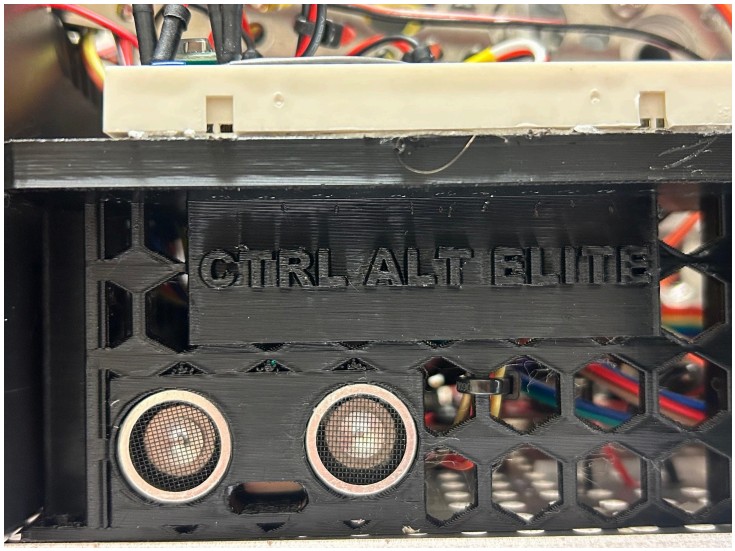

Figure 19: Team Name Side Addition

7.8.2 Arm-Gripper Spacers

To ensure proper alignment and full range of motion, the arm-gripper subsystem required precise spacing on the robot. Custom 3D-printed components, shown as attached in Figure 20, were added to correctly position the arm and gripper during rest, pickup, transport, and placement.

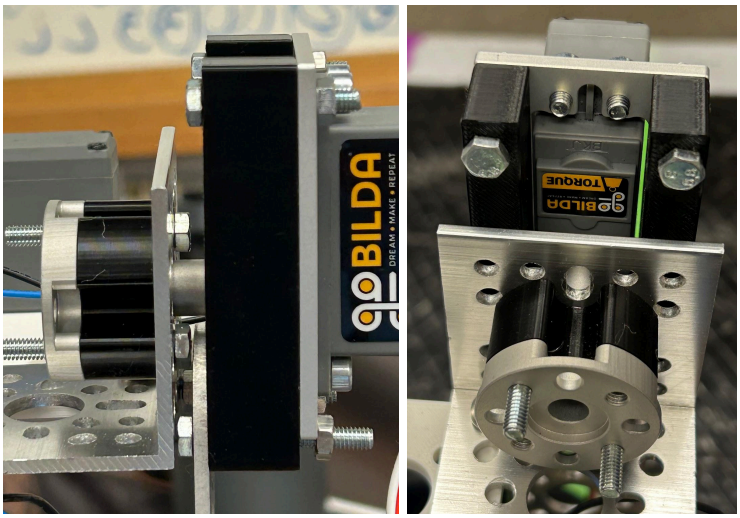

Figure 20: Gripper-Arm Spacer Connections

7.8.3 Sensor Mounts

As outlined in the parts table (Appendix A), the system utilized three color sensors, three ultrasonic sensors, and an accelerometer. To ensure stability and consistent performance, custom sensor mounts were designed and implemented throughout the vehicle.

The two side ultrasonic sensors were integrated into the sides of the 3D-printed breadboard table (see Section 7.1.2), providing a fixed and secure position for accurate wall detection from the robot's side.

The front ultrasonic sensor required its own dedicated 3D-printed holder, which ensured it remained firmly in place for precise obstacle detection (Figure 21). Directly below this sensor, a color sensor was mounted within the same 3D-printed part to detect box colors during pickup. This dual-sensor mount allowed for compact integration and aligned both sensors optimally for their respective tasks.

An accelerometer mount was added to the top of the front ultrasonic sensor mount for optimal sensor spacing and casing (Figure 21). This allowed the accelerometer to stay firmly in place during vehicle movement and operation.

For path-following and ground color detection, the remaining two color sensors were mounted in the custom 3D-printed casing fixed to the front of the grid plate (Figure 22). Initially, 3 color sensors were used for line following, but to allow extra pin space for the Accelerometer (see Section 7.8.6), the middle color sensor was unplugged.

All of these mounts were essential for maintaining consistent sensor orientation, enabling reliable detection of colors and obstacles. Overall, the use of custom sensor mounts significantly improved the system's accuracy, stability, and repeatability throughout operation.

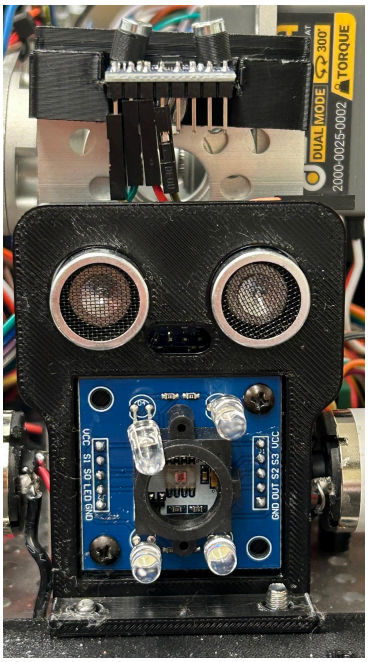

Figure 21: Front Color Sensor, Ultrasonic Sensor, Accelerometer Mount (bottom to top)

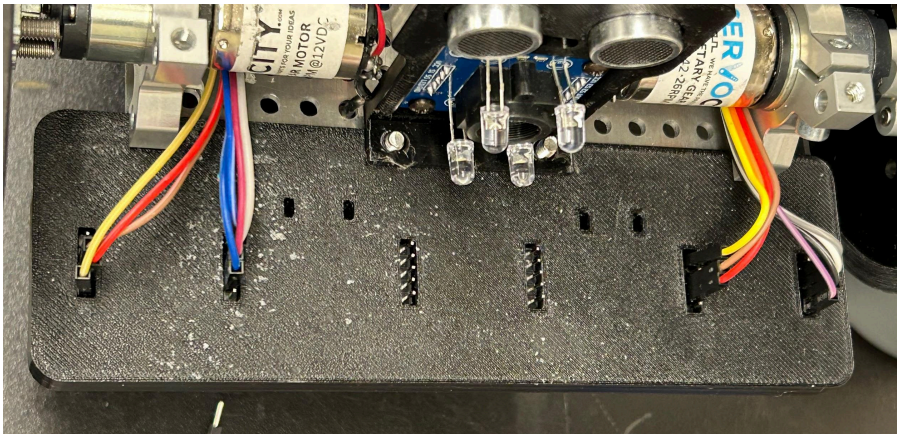

Figure 22: Ground Color Detecting Color Sensors Mount

7.8.4 Emergency Power Cable

To protect the Teensy microcontroller and connected external sensors from potential electrical damage during testing, an emergency power cutoff feature was implemented. As shown in Figure 23, a removable wire connects the Teensy's 5V (Vin) pin to the 5V power rail on the breadboard. In the event of unexpected electrical behavior or system malfunction, this wire can be quickly unplugged by the user to immediately cut power to the Teensy and halt the system. This simple but effective safeguard helped mitigate the risk of electrical failure, as outlined in Risk 04 of Section 6.4.

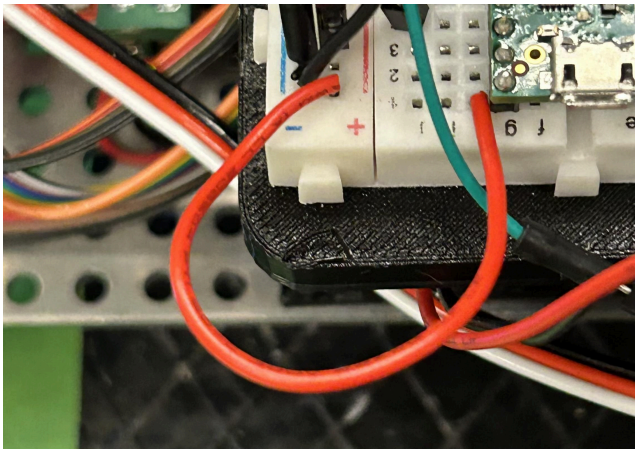

Figure 23: Power Cable

7.8.5 Outsourced Color Sensors

Initially, the provided color sensors were used for the system design. However, these sensors required tedious and unreliable calibration consistently. A decision was made to invest in different outsourced color sensors that were thoroughly researched before purchasing (AliExpress) (see Figure 24). These added sensors were implemented to improve color sensor

reliability because during the color sensor subsystem tests with the first sensors, the calibration was unreliable, and the wrong colors were often sensed. The addition of superior color sensors improves the systems capability to be reliable for the customer. The system encounters less challenges related to color sensor calibration and therefore correct path detection/following and box color detection.

Figure 24: Color Sensor (AliExpress)

7.8.6 Accelerometer

The integration of an accelerometer significantly enhanced the precision of directional changes in the obstacle-avoiding mode (see Figure 25). By measuring linear acceleration (along X, Y, and Z axes), the accelerometer enables the robot to monitor changes in orientation during rotational maneuvers. When a turn is initiated, acceleration data is used to estimate the angular displacement by integrating the motion over time. This real-time feedback allows the control system to determine when the robot has completed a 90° or 180° turn, ensuring consistent and accurate navigation. The addition of the accelerometer improved turn accuracy and contributed to the overall reliability of the robot's obstacle-avoidance capabilities (AliExpress).

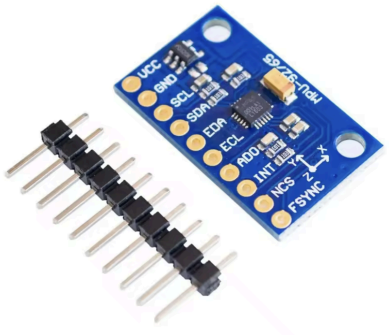

Figure 25: Gyroscope Accelerometer (AliExpress)

7.8.7 Name Plate

With leftover space in the allotted budget, a custom name plate was added to the back of the robot to enhance its visual identity and personalize the design. The plate features a red, extruded robot face and the name “R.U.D.I” (Robotic Unit for Delivery and Identification), providing a bold contrast against the black background (Figure 26). While the team name, “Ctrl Alt Elite,” appears on the side of the vehicle, the inclusion of the robot’s own name reinforces individual branding and gives the design a unique character.

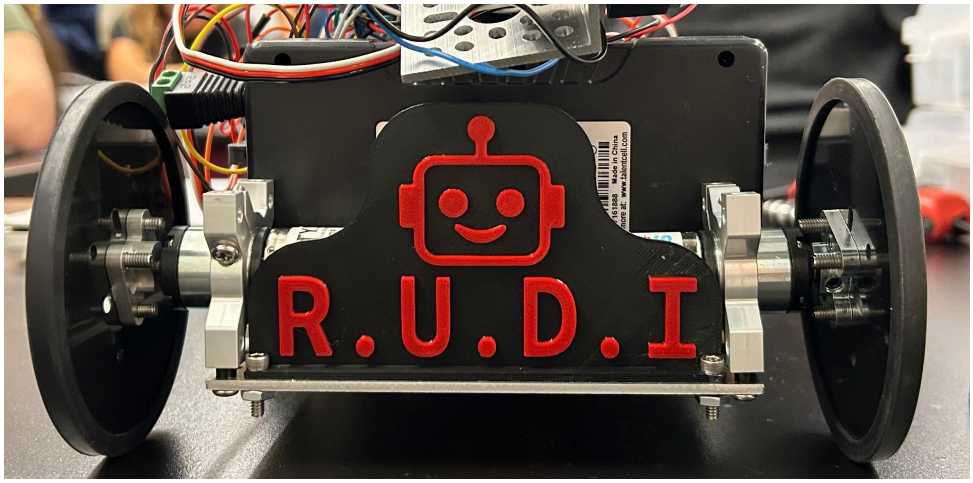

Figure 26: Name Plate

8. System Test and Verification

This section outlines the Test Evaluation Management Plan (TEMP) used to verify system requirements. Testing followed a structured, iterative process designed to ensure all functional and performance criteria were met, as outlined in Section 3. Each requirement was verified through inspection, demonstration, and system testing, with expected outcomes compared against actual results. Testing was conducted methodically; it was critical for each subsystem to be validated independently before being integrated into the full system for comprehensive evaluation.

Subsystems such as the gripper mechanism, drivetrain, ultrasonic sensors, and color detection were individually tested to confirm functionality. Once validated, these components were integrated and tested on the test course. Testing confirmed the system's ability to meet critical requirements, including box pickup and placement, color-based navigation, obstacle detection and avoidance, and collision recovery. The results are summarized in Table 3, with detailed analyses provided in the following subsections.

Table 3. Verification and Validation

Component	Requirement	Verification / Validation			
		Inspection	Demonstration	Test	Analysis
Box Pickup-and-Placement System	Requirement 7 (3.1.7)	X	X	X	See 8.1 Box Size Detection
	Requirement 10 (3.2.1)	X	X	X	See 8.2 Box Security
DriveTrain System	Requirement 2 (3.1.1)	X	X	X	See 8.3 Pickup Platform Approach
	Requirement 4 (3.1.4)	X	X	X	See 8.4 Color Path Alignment
Ultrasonic Sensor	Requirement 8 (3.1.8)	X	X	X	See 8.5 System Start
	Requirement 9 (3.1.9)	X	X	X	See 8.6 Driving through Obstacles
	Requirement 13 (3.2.4)	X	X	X	See 8.7 Obstacle Avoidance
	Requirement 14 (3.2.5)	X	X	X	See 8.8 Post Collision Correction
Obstacle Avoidance Task	Requirement 2 (3.1.1)	X	X	X	See 8.3 Pickup Platform Approach
	Requirement 13 (3.2.4)	X	X	X	See 8.6 Driving Through Obstacles
	Requirement 14 (3.2.5)	X	X	X	See 8.7 Obstacle Avoidance

8.1 Box Size Detection

Requirement 3.1.7 states the system shall not damage the box by crushing it or dropping it in any way at any point along the path. This requirement was tested and met after adding a pushbutton to the gripper (see Section 7.5.5). This pushbutton addition detects when the gripper contacts the box, triggering it to stop and preventing damage.

8.2 Box Security

Requirement 3.2.1. states the system should complete the pickup and drop off operation without damaging the box. The pushbutton on the gripper is activated when pressure from the box presses it in, signaling the gripper to stop closing. This prevents the box from being crushed while still ensuring it remains securely held during pickup, transport, and placement. Proper grip pressure is critical to avoid damage while maintaining stability.

8.3 Pickup Platform Approach

Requirement 3.1.2 states the system shall approach the “pickup” platform. Testing was conducted using the designated test and demonstration courses. After verifying each subsystem individually and then as an integrated system, with appropriate delay times and distances calibrated, the pickup and placement phase was evaluated (see Section 7.5.4). The vehicle was required to approach the platform and, upon detecting the T-shaped marker, stop and transition into the "pickup" state.

8.4 Color Path Alignment

Requirement 3.1.4 states that the system shall follow the designated colored path to the correct placement platform based on the color and size of the box. The front color sensor was used to identify the box color and ensure the vehicle followed the corresponding path (see Section 7.5.6). If a red box was detected, the vehicle entered “search” mode and located the red path to follow; the same behavior occurred for the blue colored boxes. This functionality was essential to meet BioComp LLC’s requirement that each box be matched with its respective delivery route. After independently testing the gripper and line-following subsystems, they were integrated and validated through multiple full-system runs on the test course following final code implementation for pickup, transport, and placement.

8.5 System Start

Requirement 3.1.8 states the system shall be able to start from any of the three points during the move with obstacle avoidance operation. This functionality was tested after finalizing and iterating the obstacle operation code (see Section 7.5.9). To verify that the vehicle met the requirement, the system was tested multiple times from various starting positions to ensure consistent and reliable obstacle-avoidance performance.

8.6 Driving through Obstacles

Requirement 3.1.9 states that the system shall be able to drive through the gap in the walls without hitting them. Once the state machine and code were finalized, this requirement was validated on the test course. The robot’s maneuverability was confirmed through repeated testing. The code was designed so that when the side ultrasonic sensor no longer detected a wall, the vehicle would move forward slightly before turning. This ensured smooth navigation through the gaps between obstacles.

8.7 Obstacle Avoidance

Requirement 3.2.4 states that the system shall avoid knocking down any obstacles during the obstacle avoidance operation. This requirement was tested after the final code was implemented and initially verified during early functionality checks of the ultrasonic sensors. Additional testing on the full course confirmed the robot’s obstacle avoidance capabilities.

Accurate turning, enabled by the accelerometer, was critical in ensuring the robot did not make physical contact with any obstacles.

8.8 Post Collision Correction

Requirement 3.2.5 states that the system shall navigate back to the correct path if a collision occurs. In the code, the vehicle is prompted to reroute if a collision occurs. The vehicle will continue on its path, and if it detects more obstacles, it will navigate accordingly. This was tested on the test course.

8.9 Obstacle Completion

Requirement 3.2.6 states that the system should complete the course with obstacle avoidance operation without knocking over obstacles. This functionality was validated on the test course after finalizing the code. The robot successfully completed multiple runs across various setups, consistently avoiding and not disturbing any obstacles

9. System Analysis and System Performance Metrics

The overall performance, behavior, and reliability of the robotic system is analyzed in this section. Through both quantitative and qualitative analysis, the effectiveness of the system is assessed in meeting its physical constraints and additional requirements. The system's dimensions, energy consumption, and task completion times are analyzed alongside its mechanical reliability and adaptability during testing.

9.1 System Measurements

It was necessary for the design to meet BioComp LLC's dimensional constraints of a maximum length of 21 inches and a maximum width of 11 inches. Shown in Table 4, in its resting configuration, the robot measures 12 inches in length, 7.65 inches in width, and 9.5 inches in height. When fully extended, the length reaches 20 inches, and the height in "stow" mode increases to 12.25 inches. The width remains constant in both active and resting states. These final dimensions fall within the specified limits, ensuring the design complies with BioComp LLC's size requirements. The total weight of the system was measured to be 5.25 pounds using the "Health-o-Meter" scale in the testing lab.

Table 4: System Measurements

Measurement	**At Rest**	**Fully Extended (In Use)**
Length (in)	12	20
Width (in)	7.65	7.65
Height (in)	9.5	12.25

Weight (lbs)	5.25

9.2 Gripper Stripping

Gripper failure could occur when the gripper applies excessive force beyond its mechanical limits, potentially damaging internal components or the objects being manipulated (boxes in this case). Preventing this issue was essential for the greatest reliability, safety, and longevity of the system. In the system's testing, early trials revealed that without a proper stopping mechanism, the gripper would occasionally continue closing on a box without detecting its final position. This led to increased stress on the servo motor, which began to strain during these events. To mitigate this, the dedicated gripper pushbutton was added to serve as a manual override, meaning when it is triggered, the jaws stop from closing further. This safeguard not only protects the mechanical integrity of the gripper but also prevents boxes from alteration due to too much gripper force.

9.3 Maximum Power Utilization

The system's maximum power utilization was assessed by summing the average power consumption of all components, based on datasheet specifications. This included the power draw from the microcontroller, sensors, motors, and the other supporting electronics. Using this total power consumption value and the rated capacity of the battery (66.6 kW/hr), the estimated runtime for the system was calculated. The analysis showed that the system can operate continuously for approximately 577.7 hours, or about 24 days, before the battery is fully depleted. This demonstrates that the system is highly energy-efficient and capable of extended operation without requiring frequent recharging, making it applicable for long-term situations.

9.4 Task Speed

During the final obstacle avoidance demonstrations, the system was tested on two unique courses designed by the TAs. Shown in Table 5, the first course was completed in 1 minute and 38.70 seconds, and the second in a slightly faster time of 1 minute and 37.44 seconds. Both runs were completed without intervention, showcasing consistent performance across the two trials.

For the Pickup-Place-and-Return demonstrations, the system was once again tested, first to pick up and place the large blue box,and second to pick up and place the small red box, situations designated by the TAs. As shown in Table 5, the first demonstration was successfully completed in 2 minutes and 15 seconds, and the second was completed in 2 minutes and 7 seconds, once again showing the reliability of the system.

Table 5: Final Task Speeds

Task	Demonstration 1 Speed (min)	Demonstration 2 Speed (min)
Obstacle Avoidance	1:38.70	1:37.44
Pickup-Place-and-Return	2:15.00	2:07.00

9.5 Speed Analysis

For the obstacle avoidance performance, the close timing between the two runs highlights the reliability of the system's performance. Although the vehicle was not among the fastest (placing seventh) in comparison to some other teams, which recorded times closer to one minute, the system maintained consistent speeds without sacrificing control or accuracy. This balance between speed and dependability contributed to the vehicle's success in navigating each course effectively.

The pickup/placement times were also close to each other, at only 8 seconds apart. These times were fast compared to other teams. and the system placed second overall. This concludes that the vehicle's speed was relatively fast (for a system powered by torque motors).

9.6 Quality Analysis

A quality assessment was conducted for the Pickup-Place-and-Return task, focusing on two key components: line following and turning accuracy.

The vehicle demonstrated a 95% line-following accuracy, with only minor deviations that were quickly corrected. This high performance is largely attributed to the integration of the accelerometer, which enabled smoother course corrections. Specifically, the corrective turning angle was reduced from 10° in initial trials to 5°, improving the precision of the adjustments. The smaller the correction angle, the more finely the vehicle could stay on track, which directly contributed to the improved accuracy. This result highlights the system's reliability in consistently following the designated path.

Right/Left turning performance was rated at 90% accuracy for the demonstration. While the vehicle generally executed turns reliably, performance was affected by ripples in the demonstration mat. These irregularities disrupted wheel traction and alignment during turns. On a more taut surface, turn accuracy could reach near-perfect levels.

9.7 Budget Analysis

A strict budget limit of $30 was maintained to meet Requirement 19 (see Section 3.4.2). A detailed budget tracking table (Table 9 of Appendix D) was created at the beginning of the project, and it was continuously updated to reflect all expenses. This table includes itemized

costs for all components, including 3D-printed parts, purchased hardware, and any other outsourced items not included in the provided kit.

This ongoing documentation helped guide purchasing decisions and minimized unnecessary expenses, ensuring that all additions to the system were cost-effective and essential to functionality. The final cost was $19.74, which is significantly under the allocated budget. Careful resource management and successful adherence to financial constraints were demonstrated without compromising the quality or performance of the final design.

9.8 Color Sensor Calibration

To ensure accurate color detection during all performance tests, the color sensors were recalibrated at the start of each testing day. Environmental conditions such as lighting, mat ripples/glare, and shadows varied from day to day, significantly impacting how the sensors interpreted colors. Also if the LEDs moved, the color detection would be different. Recalibration helped minimize errors caused by these variations and ensured that the sensors consistently recognized the intended target colors.

The calibration process involved adjusting the acceptable RGB value ranges for each sensor and color category based on live readings under the current testing conditions. These updated thresholds were then hardcoded into the control system for use during the demonstration. Table 6 summarizes the RGB value thresholds used for each sensor (Left Front, Right Front, and Middle Upper) to classify detected surfaces as red, blue, green, black, or white. These ranges reflect sensor-specific variations and environmental tuning, which were essential for navigation and box-sorting logic.

Table 6: Color Sensor Calibration Values

Color Sensor	**Red Values**	**Blue Values**	**Green Values**	**Black Values**	**White Values**
Left Front	r >= 60 && r <= 100 && g >= 170 && g <= 280 && b >= 140 && b <= 240	r >= 180 && r <= 320 && g >= 90 && g <= 160 && b >= 60 && b <= 100	r >= 70 && r <= 120 && g >= 60 && g <= 100 && b >= 90 && b <= 140	r > 180 && g > 180 && b > 160	r < 50 && g < 50 && b < 50
Right Front	r >= 110 && r <= 170 && g >= 500 && g <= 750 && b >= 370 && b <= 540	r >= 650 && r <= 900 && g >= 220 && g <= 340 && b >= 110 && b <= 180	r >= 140 && r <= 210 && g >= 110 && g <= 150 && b >= 170 && b <= 260	r > 180 && g > 180 && b > 160	r < 60 && g < 60 && b < 60
Middle Upper	r < 200 && g > 100 && b > 90	r > 250 && g > 100 && b > 70 && r > g	N/A	N/A	N/A

		&& r > b			

9.9 Final Test Analysis

For the final demonstrations, the system was evaluated on a different mat than the one used during development. This new setup introduced unexpected challenges including tarp ripples and light glare from nearby windows, which initially interfered with the color sensors' ability to detect the white boundary lines during the obstacle avoidance task. However, through a series of recalibrations, the sensors were successfully adapted to the new conditions. Once calibrated, the vehicle completed two full obstacle avoidance courses on the final mat, successfully navigating all obstacles. These results showed the adaptability of the system under varied conditions.

The final demonstrations completed all tasks effectively, reliably, and in full alignment with the requirements set by BioComp LLC. The system proved to be a robust and capable design, successfully avoiding obstacles, following colored paths, and handling boxes with varying properties through a combination of precise sensing and smart control. With features like accurate line following and adaptive pickup mechanisms, the final design showcases a well-integrated, high-performing solution. Overall, it is considered a success, delivering both technical functionality and practical reliability.

10. Summary

This project resulted in the successful development of an autonomous robotic system tailored to meet BioComp, LLC's needs. The robot was designed to complete two main tasks: navigating an obstacle course while avoiding collisions and performing pickup, placement, and return operations with fragile boxes of varying sizes and colors. The final system completed both tasks reliably, accurately, and in a timely manner.

The design process began with defining clear performance, functional, and other system requirements. Four conceptual designs were evaluated using a decision matrix to determine the most effective overall approach. Throughout development, potential risks were identified, and mitigation strategies were implemented to avoid setbacks and to meet deadlines. Each subsystem was tested independently before being integrated into the complete system. A Teensy microcontroller coordinated all hardware through a state-driven algorithm implemented in Arduino code.

Key features of the final system include a gripper-mounted pushbutton for box size detection, outsourced color sensors for improved path tracking and color identification, and an accelerometer for sharp and accurate turns. Buck converters were used to manage power distribution, ensuring reliable operation across all components. Additional sensor mounts and aesthetic enhancements improved both functionality and appearance. The robot was also

equipped with a mode-switching pushbutton for the user to easily transition between obstacle avoidance and pickup/placement modes.

Overall, the project demonstrated a successful design and development process that led to an effective solution. The final robot met all specified requirements and proved to be a capable system for BioComp's intended application.

.

11. Documentation and Acknowledgments

The Integration 1 & 2 lecture slides and figures were used. Past assignment templates, and documents from Integration 1 on Canvas were used as a guide to the project and report. Components like the accelerometer and color sensors were sourced from AliExpress. Collaborating with other students during testing helped us solve problems and improve our design. ChatGPT was used for help with coding and troubleshooting.

Appendix A

Table 7: Parts List

Element	Amount	Details
26 RPM Servo Motor	4	Brushed DC Motor – 26 RPM Premium Planetary Gear Motor
Disc Wheels	4	3" Acrylic Disc Wheel
Ultrasonic Sensor	3	Ultrasonic Module HC-SR04 Distance Sensor
Color Sensor	4	AliExpress
Motor Controller	2	L298 Motor Driver Module
Servo Torque Motor	2	Dual Mode Servo (25-2, Torque)
Servo Speed Motor	1	Dual Mode Servo (25-3, Speed)
Grid Plate	1	1116 Series Grid Plate
Motor-Wheel Mounts	4	Metal Grid Plate Elements Provided
Servo Mount (Arm Plates)	4	Metal Grid Plate Elements Provided
Breadboard	2	400 Point Breadboard
LED	1	LED Blue Diffused 6MM Round T/H
Pushbutton	3	Momentary Pushbutton Switch 12mm Square Tactile
Microcontroller	1	Teensy 4.1 Microcontroller
Buck Convertor	2	Mini 360 3A DC-DC Voltage Step Down Power Converter Buck Module
Battery	1	12V DC Power Connector

		and 12V Battery
Gripper Assembly	1	• (2x) 1120 Series U-Channel (1 Hole, 48mm Length) • (1x) 1105 Series Round-End Pattern Plate (7 Hole, 176mm Length) • (2x) 1801 Series Servo Plate • (2x) 1221 Series 2-Side, 2-Post Pattern Mount (32-2) • (1x) 1113 Series L-Channel (1 Hole, 48mm Length) • (1x) 1505 Series 32mm OD Counterbored Pattern Spacer (6mm Length) • (1x) 1910 Series Servo Hub Shaft (25 Tooth Spline, 10mm Shaft Diameter, 33mm Length)
Color Sensor Mount	1	3D-Printed
Breadboard Table	1	3D-Printed
Gripper - Gripper Arm Connection	1	ServoCity Servo-Driven Parallel Gripper Kit
Front Ultrasonic/Color Sensor Holder	1	3D-Printed
Accelerometer	1	AliExpress

Appendix B

Figure 27: Gripper Assembly (i.e. Irvin Jones, Integration 1)

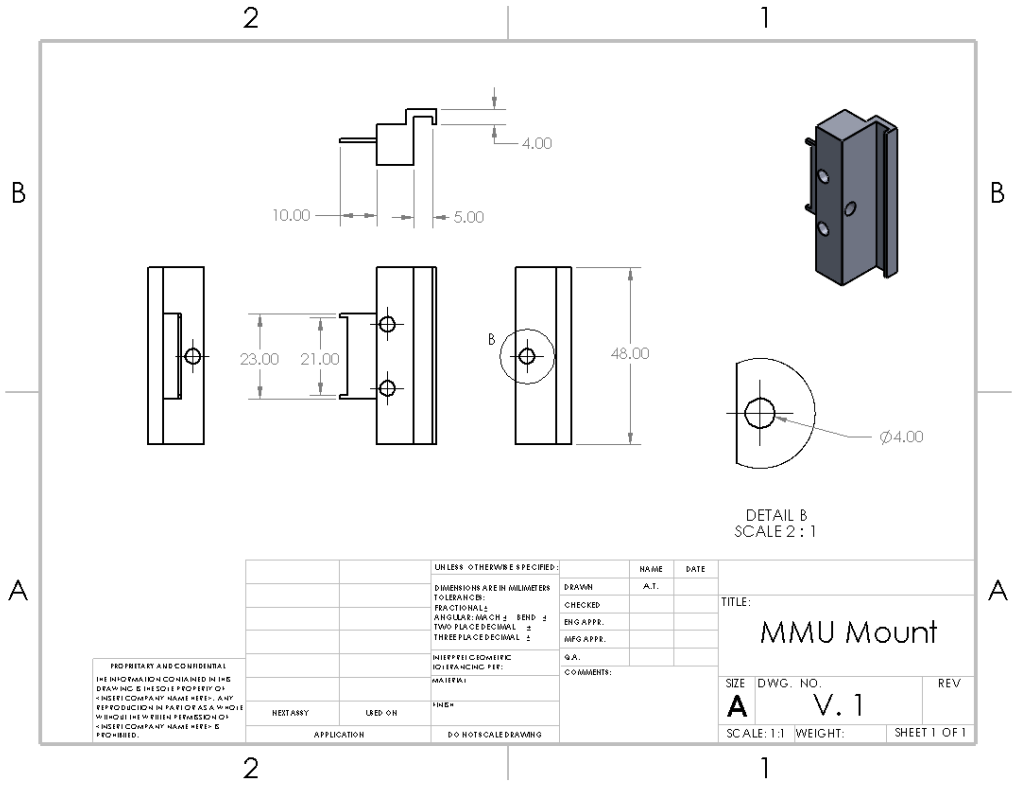

Figure 28: Accelerometer Mount CAD Drawing

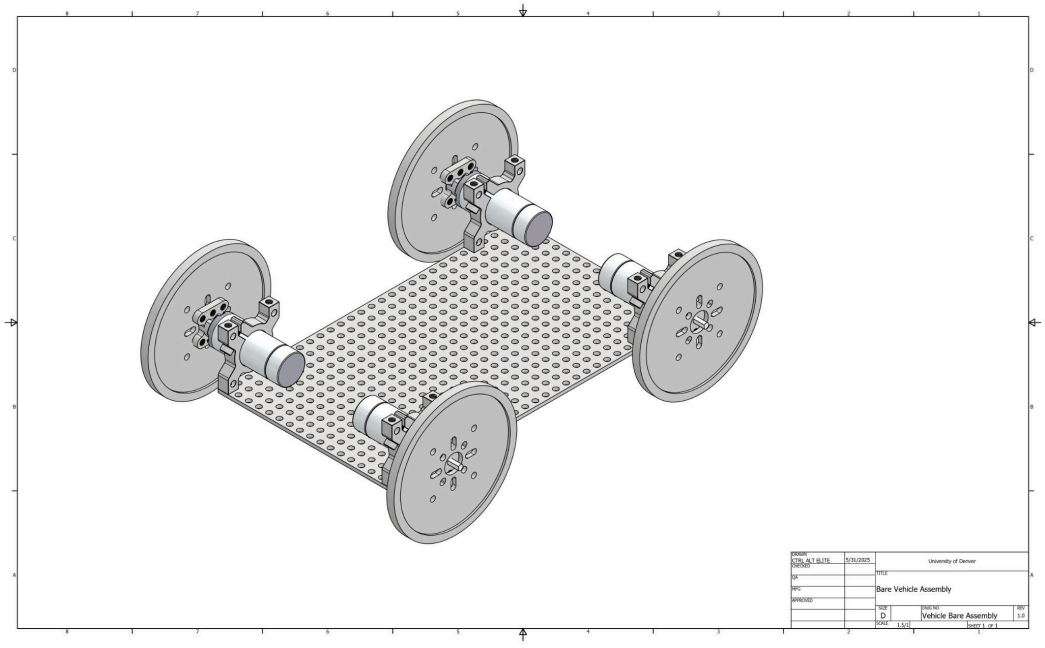

Figure 29: Bare Vehicle CAD Drawing

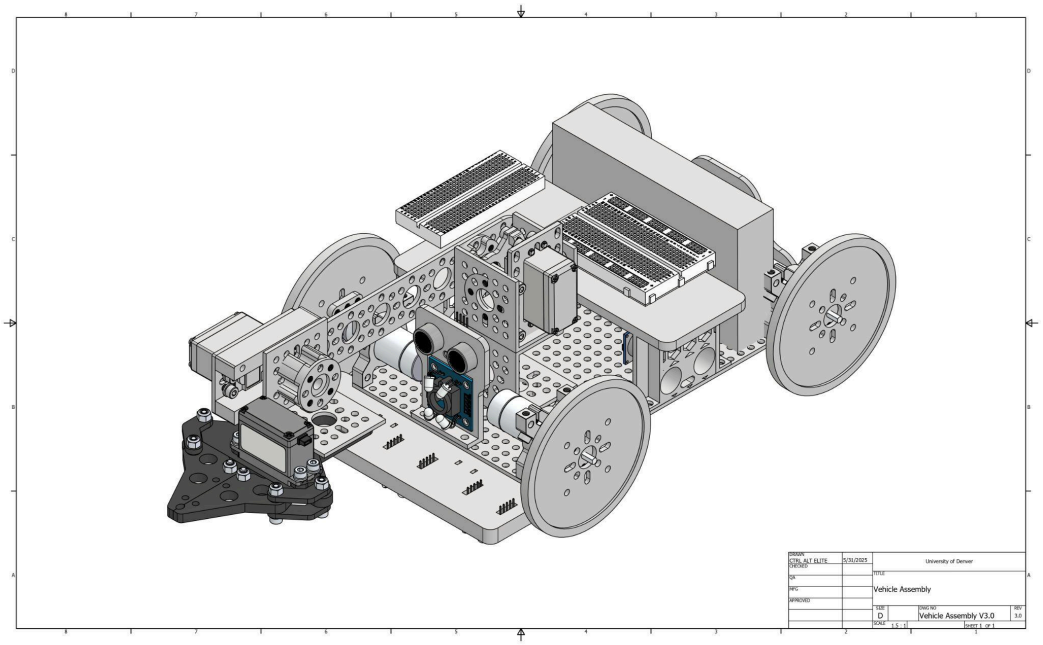

Figure 30: Assembled Vehicle CAD Drawing

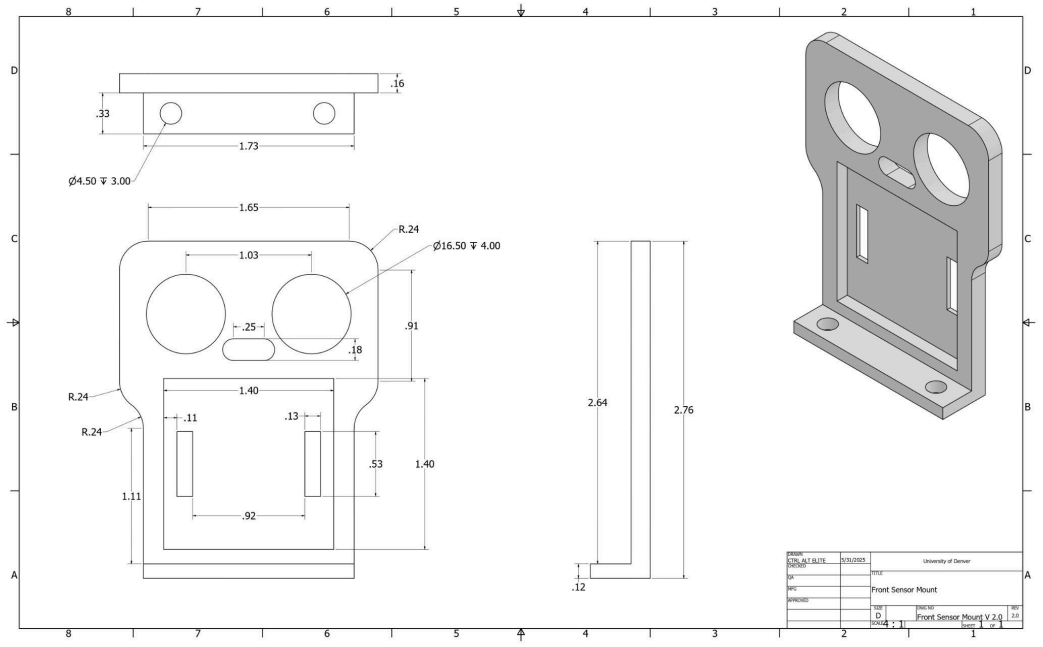

Figure 31: Front Sensor Mount CAD Drawing

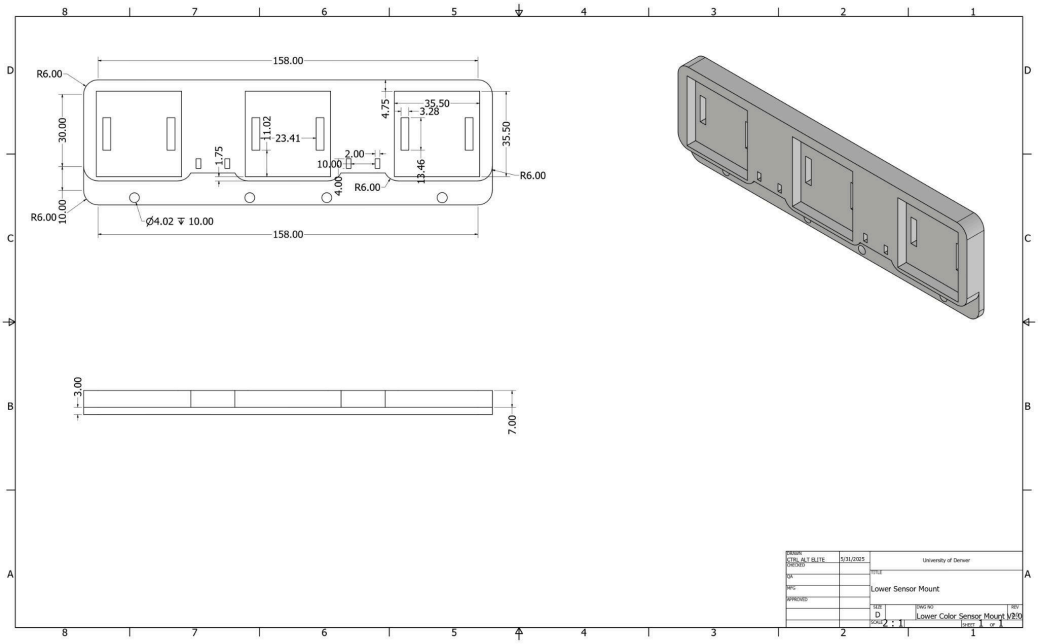

Figure 32: Lower Color Sensor Mount CAD Drawing

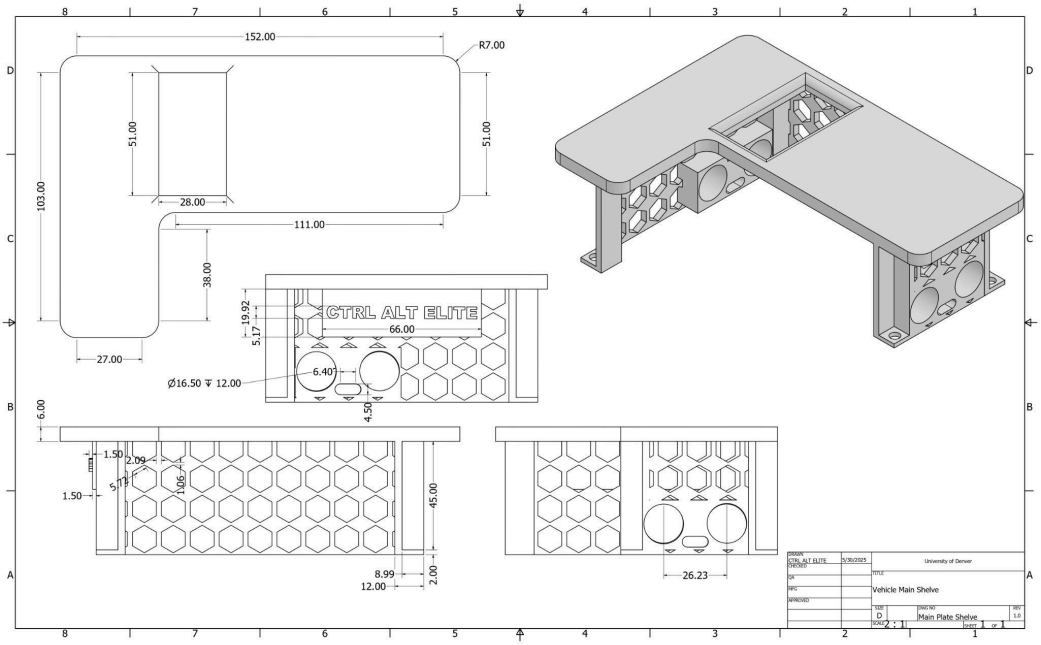

Figure 33: Main Plate Shelf CAD Drawing

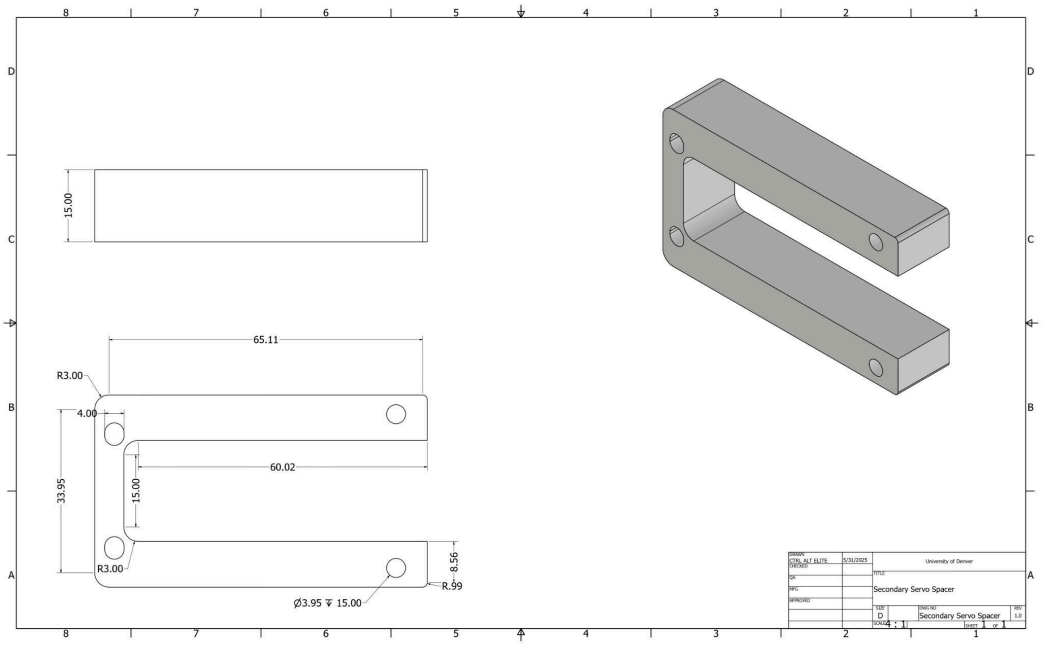

Figure 34: Secondary Servo Spacer CAD Drawing

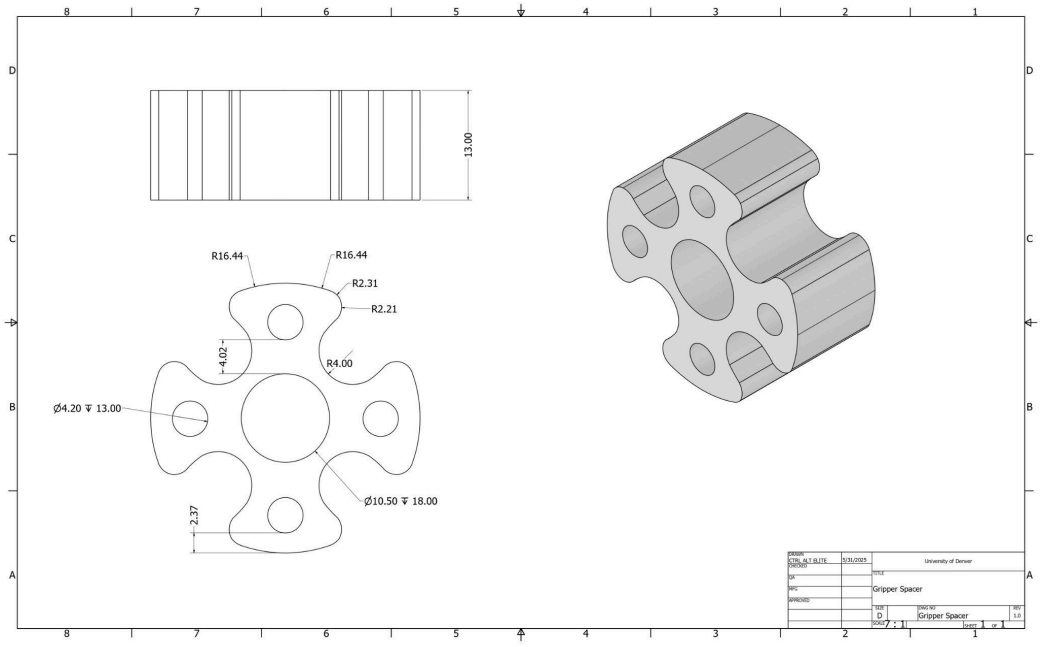

Figure 35: Gripper Spacer CAD Drawing

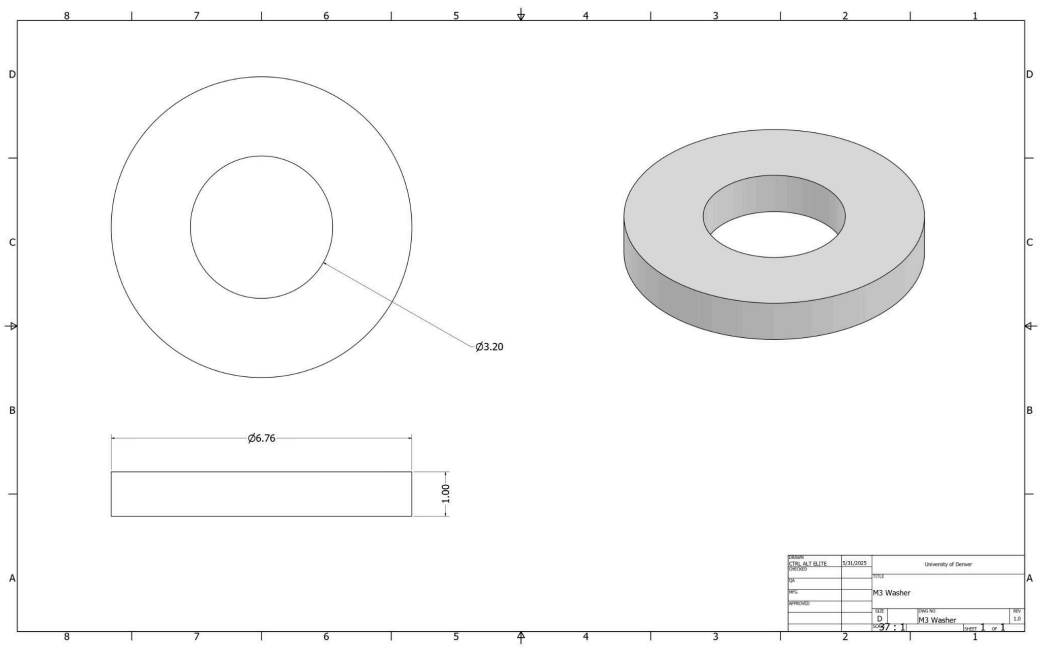

Figure 36: M3 Washer CAD Drawing

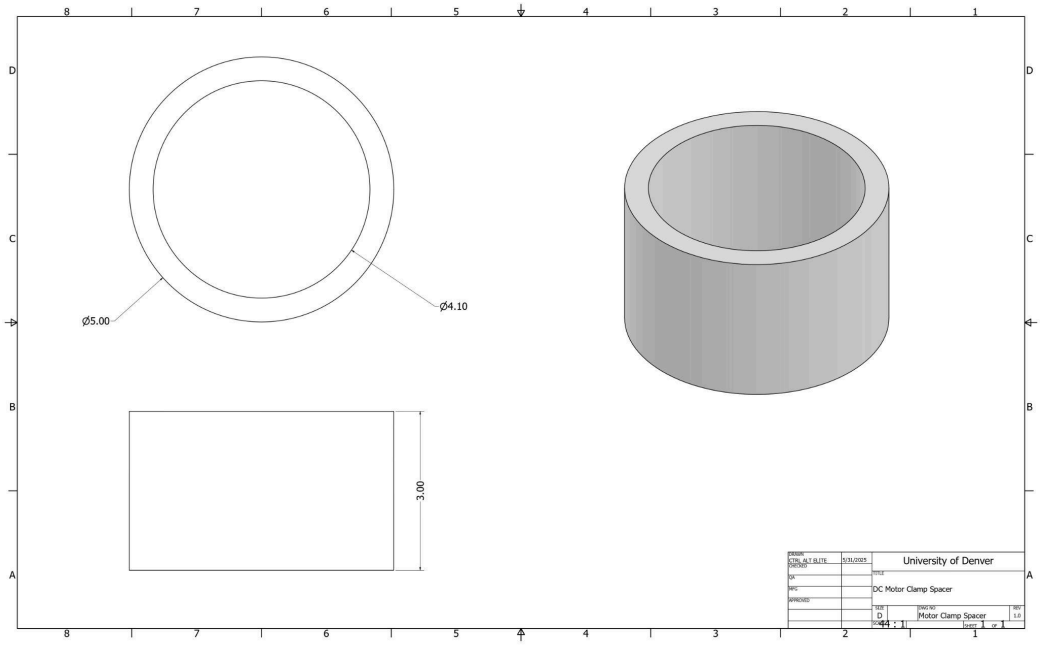

Figure 37: Motor Clamp Spacer CAD Drawing

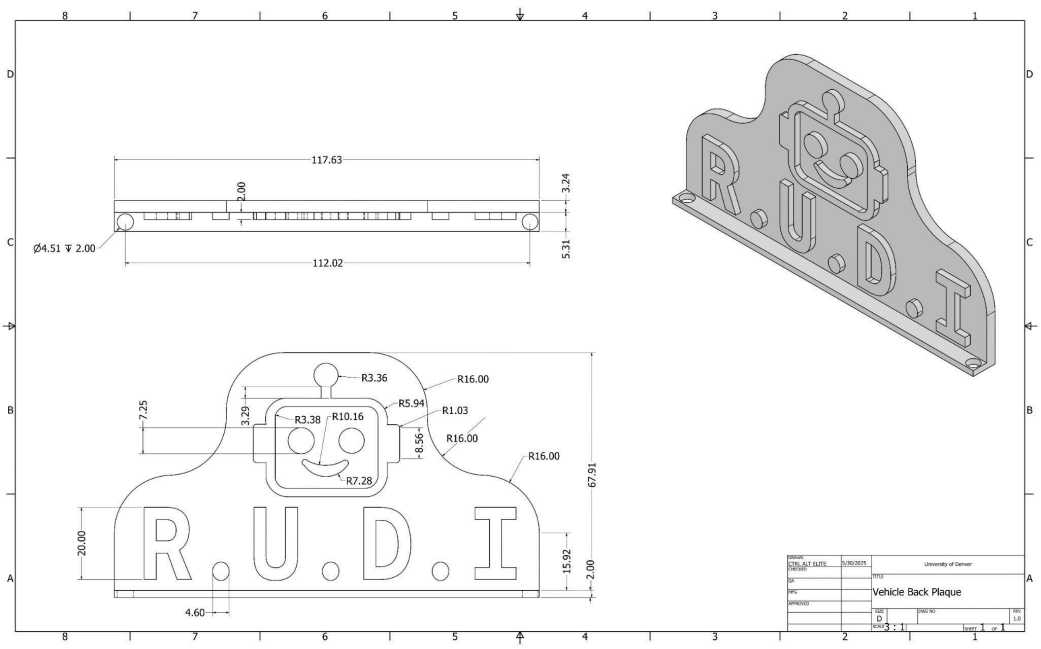

Figure 38: Back Plate CAD Drawing

Appendix C

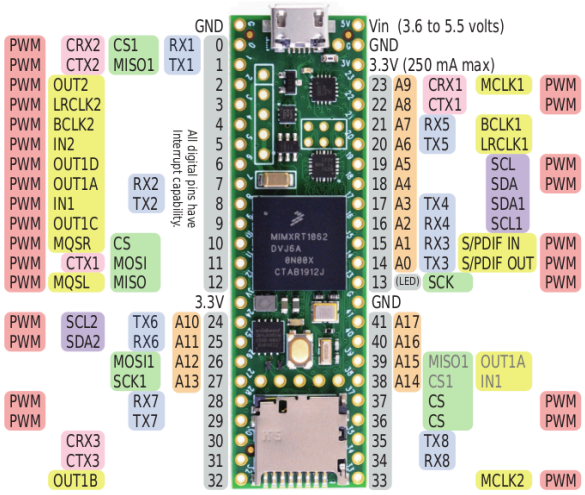

Figure 39: Teensy Microcontroller Pin Diagram

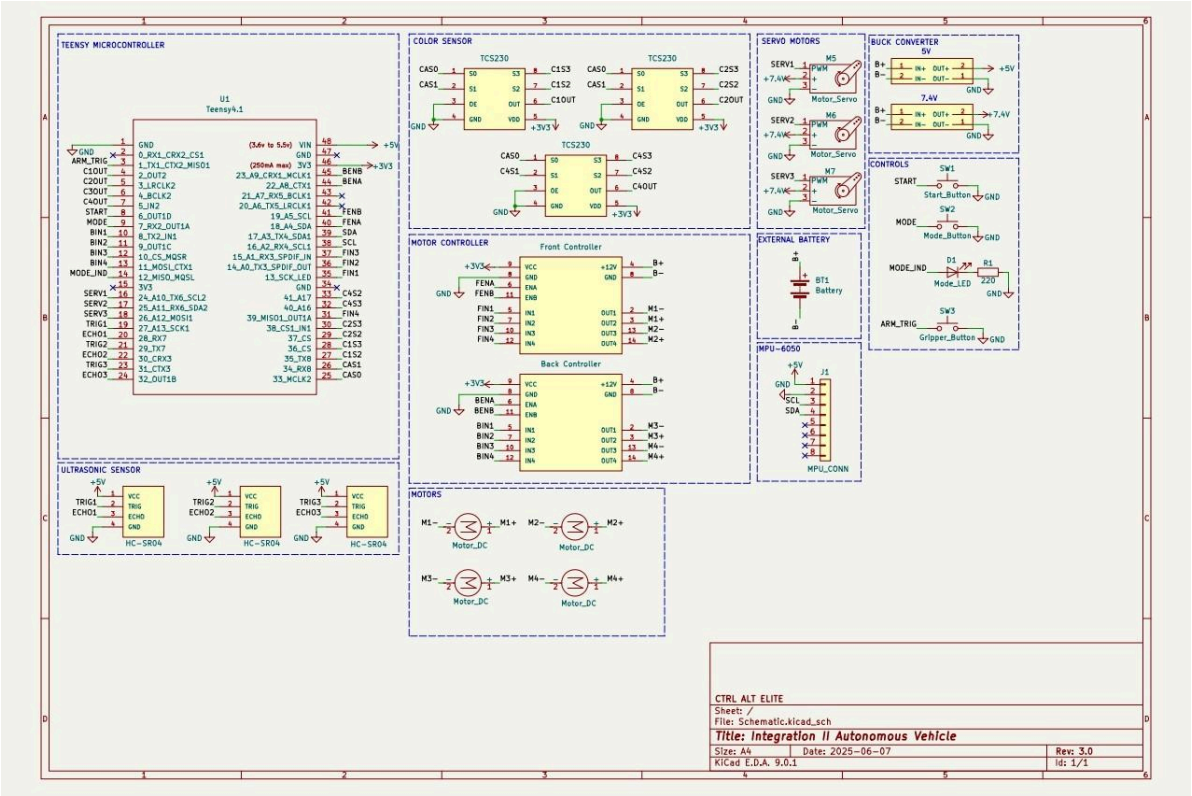

Figure 40: Vehicle Schematic

Table 8: Wiring Table

Sensor Number	Signal	Sensor	Teensy
Motor Controllers // Red			
Front // Yellow	ENB	Yellow	Blue
	IN3	Green	Green
	IN4	Blue	GreenT
	3V3	Orange	Red
	ENA	Purple	Yellow
	IN1	Gray	White
	IN2	White	Orange
	GND	Black	Black
Back // Green	ENB	Orange	Orange
	IN3	Brown	BlueT
	IN4	White	Green
	3V3	Red	Red
	ENA	Grey	Yellow
	IN1	Purple	White
	IN2	Blue	Blue
	GND	Black	Black
Ultrasonic Sensors // Blue			
Front // Yellow	Ground	Black	Black
	Echo	Brown	White
	Trig	Red	Blue
	VCC	Orange	Orange
Right // Green	Ground	Brown	Black
	Echo	Red	Yellow
	Trig	Orange	Orange
	VCC	Yellow	Red
Left // Red	Ground	White	Black
	Echo	Gray	White
	Trig	Purple	Yellow
	VCC	Blue	Red
Color Sensors // Yellow			
Front // White	S3	Green	Green
	S2	Yellow	Yellow
	S1	Gray	Blue
	S0	White	White
	OE	Purple	Black
	OUT	Orange	Orange
	VCC	Red	Red
	GND	Blue	Black
Left // Red	S3	Black	Blue
	S2	White	White

Sensor Number	Signal	Sensor	Teensy
	S1	Red	Orange
	S0	Brown	Yellow
	OE	Orange	Black
	OUT	Gray	Green
	VCC	Purple	Red
	GND	Yellow	Black
Right // Blue	S3	Blue	White
	S2	Purple	Yellow
	S1	Red	Green
	S0	Brown	Blue
	OE	Green	Black
	OUT	White	Orange
	VCC	Yellow	Red
	GND	Blue	Black
Servos // Green			
Main // Yellow	VCC	Red	Red
	GND	Black	Black
	SIG	White	White
Secondary // Blue	VCC	Red	Red
	GND	Black	Black
	SIG	White	Blue
Gripper // White	VCC	Red	Red
	GND	Black	Black
	SIG	White	Green
Gyro // White			
Main // Red	VCC	Red	Red
	GND	Black	Black
	SDA	Blue	Orange
	SCL	Green	Yellow

Appendix D

Table 9: Budget Tracking

Item	**Details**	**Cost ($)**
3D printed Parts	Includes: Lower color sensor mounts x 3, Front sensor mount x 1, Main shelve x 1, Main servo arm spacer x 1, Servo horn spacer x 1, Motor controller washer x 11, Motor controller spacer x 8	2.21
Color Sensors	x 4 from AliExpress	14.68
3D Printed Parts	Includes: New Lower Color Sensor Mount and New Front Sensor Mount	0.63
MPU6050	x 1 from AliExpress	1.87
3D Printed Parts	Includes: MPU Sensor Mount and Back Name Plate	0.35
	Total Spent	19.74

Appendix E

```
#include "MotorControl.h"
#include "ColorControl.h"
#include "ServoControl.h"
#include "GyroControl.h"
#include "UltrasonicControl.h"

//--------------------------------Control
Macros--------------------------------//
#define MODE_BUTTON 6
#define START_BUTTON 7
#define GRIPPER_BUTTON 1
#define MODE_IND 12

//-------------------------------Global
Variables-------------------------------//
int gMode = 0;
int gBoxColor = -1;
int gBoxSize = 0;
int gDropOffStatus = 0;

//-------------------- -------State Machine
Enumerate----------------------------//

enum Vehicle_State {
  IDLE_STATE,
  MAIN_PEDESTAL_FIND_STATE,
  PICK_UP_STATE,
  BOX_IDENTIFICATION_STATE,
  FIND_RED_LINE_STATE,
  FIND_BLUE_LINE_STATE,
  FOLLOW_LINE_STATE,
  SMALL_PEDESTAL_FIND_STATE,
  LARGE_PEDESTAL_FIND_STATE,
  DROP_OFF_STATE,
  RETURN_TO_LINE_STATE,
  NAVIGATE_STATE,
  WALL_FOUND_STATE,
  FOLLOW_RIGHT_WALL_STATE,
  FOLLOW_LEFT_WALL_STATE,
  COURSE_EDGE_STATE,
```

```
  RIGHT_OPENING_STATE,
  LEFT_OPENING_STATE,
  FINISH_LINE_STATE
};

Vehicle_State currentState = IDLE_STATE;

void setup()
{
  Serial.begin(115200);

  // Sensor and Motor Initialization
  initMotors();
  initUltrasonics();
  initColorSensors();
  initServos();
  initGyro();

  // Controls Set Up
  pinMode(MODE_BUTTON, INPUT_PULLUP);
  pinMode(START_BUTTON, INPUT_PULLUP);
  pinMode(GRIPPER_BUTTON, INPUT_PULLUP);
  pinMode(MODE_IND, OUTPUT);

}

void loop()
{
  switch (currentState) { // State Machine for Robot
  //--------------- IDLE_STATE ---------------//
  case IDLE_STATE:
  {
    // Vehicle is stopped, waiting for start button to be pressed, moves to
    // MAIN_PEDESTAL_STATE or NAVIGATE_STATE depending on state of mode
button.

    digitalWrite(MODE_IND, gMode); // Show current mode selection on LED
    stowModeEmpty(); // Move arm to default position

    // Toggle mode variable when mode button is pressed
    if (digitalRead(MODE_BUTTON) == 0)
    {
      gMode = !gMode;
```

```
      delay(250); // Delay for debouncing
    }

    // Start robot in the chosen mode
    if (digitalRead(START_BUTTON) == 0)
    {
      if (gMode == 1)
      {
        currentState = MAIN_PEDESTAL_FIND_STATE;
      }
      else
      {
        currentState = NAVIGATE_STATE;
      }
    }
    break;
  }

  //--------------- MAIN_PEDESTAL_FIND_STATE ---------------//
  case MAIN_PEDESTAL_FIND_STATE:
  {
    // Vehicle moves towards pedestal, reading front ultrasonic sensor.
Once front
    // ultrasonic sensor is below a certain threshold, move to
PICK_UP_STATE.

    moveFWD(70);
    int distance = readUltrasonicDistance(TRIG1, ECHO1); // Read values
from front ultrasonic sensor

    if (distance <= 21) // Check if vehicle is closer than 21 cm to the
pedestal
    {
      int distance = readUltrasonicDistance(TRIG1, ECHO1); // Checks
distance again for verification
      if (distance <= 21)
      {
        stop();
        currentState = PICK_UP_STATE; // Move to PICK_UP_STATE
      }
    }
    break;
  }
```

```
  //--------------- PICK_UP_STATE ---------------//
  case PICK_UP_STATE:
  {
    // Vehicle is to pick up box. Close to large box size. If button on
gripper is triggered,
    // the system moves to BOX_IDENTIFICATION_STATE. If the button is not
triggered, the gripper
    // closes to the small box size. The system then movves to
BOX_IDENTIFICATION_STATE. The size
    // of the box is saved in a global variable called gBoxSize. 1 for
large box, and 2 for small box.

    // Attempt to pick up large box
    moveServosTogether(96, 85, 138);
    delay(1000);
    pickUpModeLarge();
    delay(2000);

    // Check if the button on the gripper has been pressed
    if (digitalRead(GRIPPER_BUTTON) == 0)
    {
      stowModeLarge();
      delay(2000);
      gBoxSize = 1;
      currentState = BOX_IDENTIFICATION_STATE;
    }

    // Try and pick up small box if large box does not trigger gripper
button
    else
    {
      delay(1000);
      pickUpModeSmall();
      delay(500);
      stowModeSmall();
      delay(2000);
      gBoxSize = 2;
      currentState = BOX_IDENTIFICATION_STATE;
    }

    break;
  }
```

```
  //--------------- BOX_IDENTIFICATION_STATE ---------------//
  case BOX_IDENTIFICATION_STATE:
  {
    // Vehicle moves backwards until green line is found. The vehicle than
moves the servos
    // to place the box in front of the color sensor. The box color is read
and if the box is
    // red, the system goes to FIND_RED_LINE_STATE. If it is blue, it goes
to FIND_BLUE_LINE_STATE
    // the box color is saved as a global variable called gBoxColor.
gBoxColor is red if the box is
    // red and 2 if the box is blue.

    moveBWD(70);

    // Keep moving backward until one of the color sensors detects the
green line
    if ((classifyColor(2) == 4) || (classifyColor(3) == 4))
    {
      stop();

      if (gBoxSize == 1) // Large
      {
        colorReadModeLarge();
        delay(2000);

        if (classifyColor(1) == 3) // Red
        {
          gBoxColor = 3;

          currentState = FIND_RED_LINE_STATE;
        }
        else if (classifyColor(1) == 2) // Blue
        {
          gBoxColor = 2;

          currentState = FIND_BLUE_LINE_STATE;
        }

        stowModeLarge();

      }
```

```
      else // Small
      {
        colorReadModeSmall();
        delay(2000);

        if (classifyColor(1) == 3) // Red
        {
          gBoxColor = 3;

          currentState = FIND_RED_LINE_STATE;
        }
        else if (classifyColor(1) == 2) // Blue
        {
          gBoxColor = 2;

          currentState = FIND_BLUE_LINE_STATE;
        }

        stowModeSmall();

      }

    }

    break;
  }

  //--------------- FIND_RED_LINE_STATE ---------------//
  case FIND_RED_LINE_STATE:
  {
      // Vehicle moves foward to the center of rotation of the vehicle. It
then uses the
      // accelerometer to turn 90 degrees to the right. It then moves
foward until the red
      // line is found. It then turns right again 90 degrees and moves to
FOLLOW_LINE_STATE

      moveFWD(70);
      delay(1000);
      stop();
      gyroTurn("RIGHT", 93);
```

```
      while (true)
      {
        int color = classifyColor(3);
        Serial.print("Sensor 4 color: ");
        Serial.println(color);

        if (color == 3) break;

      moveFWD(70);
      delay(50); // Give time to read the color properly
      }

      delay(1500);

      gyroTurn("RIGHT", 90);

      currentState = FOLLOW_LINE_STATE;

      break;
  }

  //--------------- FIND_BLUE_LINE_STATE ---------------//
  case FIND_BLUE_LINE_STATE:
  {
    // Vehicle moves foward to the center of rotation of the vehicle. It
then uses the
    // accelerometer to turn 90 degrees to the left. It then moves foward
until the blue
    // line is found. It then turns left again 90 degrees and moves to
FOLLOW_LINE_STATE

    moveFWD(70);
    delay(1000);
    stop();
    gyroTurn("LEFT", 85);

    while(classifyColor(2) != 2)
    {
      moveFWD(70);
    }

    delay(1500);
```

```
    gyroTurn("LEFT", 88);

    currentState = FOLLOW_LINE_STATE;

    break;
  }

  //--------------- FOLLOW_LINE_STATE ---------------//
  case FOLLOW_LINE_STATE:
  {
    // The vehicle will follow the line until the second green line is
found. Then, depending on the state of
    // gBoxSize, the system moves to SMALL_DEDESTAL_FIND_STATE, or
LARGE_PEDESTAL_FIND_STATE. This state is also
    // used to return the vehicle to the first green line where the
procedure is complete. The state will differentiate
    // by keeping track of the gDropOffStatus global variable. If the
variable is 0, the drop off is not complete
    // and 1 after the drop off. The variable becomes 3 once the vehicle
crosses the green line closest to the
    // drop off point.

    moveFWD(70);

    int leftColor = classifyColor(2);
    int rightColor = classifyColor(3);

    // Detects if the left or right color sensor detects the same color of
the box
    if ((rightColor == gBoxColor) || (leftColor == gBoxColor))
    {
      stop();

      // This is true at the cross when the vehicle needs to make a
decision on what pedestal to go to
      if ((rightColor == gBoxColor) && (leftColor == gBoxColor))
      {
        if (gBoxSize == 1) // Large
        {
          currentState = LARGE_PEDESTAL_FIND_STATE;
        }
        if (gBoxSize == 2) // Small
        {
```

```
          currentState = SMALL_PEDESTAL_FIND_STATE;
        }
      }

      // The vehicle will make 5 degree corrective turns to stay following
the line
      else if (rightColor == gBoxColor)
      {
        gyroTurn("RIGHT", 5);
      }

      else if (leftColor == gBoxColor)
      {
        gyroTurn("LEFT", 5);
      }

    }

    // This is true if either of the left or right color sensors detects
the green line
    else if ((rightColor == 4) || (leftColor == 4))
    {
      if (gDropOffStatus == 2)
      {
        stop();
        currentState = IDLE_STATE;
      }

      if (gDropOffStatus == 1)
      {
        gDropOffStatus++;
        delay(1000); // Delay is to allow the vehicle to cross the green
line before reading sensors again
      }
    }

    break;
  }

  //--------- SMALL_PEDESTAL_FIND_STATE ---------//
  case SMALL_PEDESTAL_FIND_STATE:
  {
    // The vehicle will move foward until the split is found, it then will
```

```
turn right and
    // follow the line until the pedestal is found by using the front
ultrasonic sensor. The
    // system then moves to DROP_OFF_STATE

    moveFWD(70);
    delay(1500);
    gyroTurn("RIGHT", 87);

    int finalStraight = 0;
    int dropOff = 0;

    while(!finalStraight)
    {
      int leftColor = classifyColor(2);

      moveFWD(70);

      if (leftColor == gBoxColor)
      {
        delay(1300);
        gyroTurn("LEFT", 90);
        finalStraight = 1;
      }

    }

    while(!dropOff)
    {
      moveFWD(70);
      int distance = readUltrasonicDistance(TRIG1, ECHO1);

      if (distance <= 21)
      {
        int distance = readUltrasonicDistance(TRIG1, ECHO1);
        if (distance <= 21)
        {
          stop();
          dropOff = 1;
          currentState = DROP_OFF_STATE;
        }
      }
```

```
    }



    break;
  }

  //--------- LARGE_PEDESTAL_FIND_STATE ---------//
  case LARGE_PEDESTAL_FIND_STATE:
  {
    // The vehicle will move foward until the split is found, it then will
turn left and
    // follow the line until the pedestal is found by using the front
ultrasonic sensor. The
    // system then moves to DROP_OFF_STATE

    moveFWD(70);
    delay(1500);
    gyroTurn("LEFT", 88);

    int finalStraight = 0;
    int dropOff = 0;

    while(!finalStraight)
    {
      int rightColor = classifyColor(3);

      moveFWD(70);

      if (rightColor == gBoxColor)
      {
        delay(1500);
        gyroTurn("RIGHT", 88);
        finalStraight = 1;
      }

    }

    while(!dropOff)
    {
      moveFWD(70);
      int distance = readUltrasonicDistance(TRIG1, ECHO1);
```

```
      if (distance <= 21)
      {
        int distance = readUltrasonicDistance(TRIG1, ECHO1);
        if (distance <= 21)
        {
          stop();
          dropOff = 1;
          currentState = DROP_OFF_STATE;
        }
      }

    }

    break;
  }

  //--------------- DROP_OFF_STATE ---------------//
  case DROP_OFF_STATE:
  {
    // The vehicle will drop the box off onto the pedestal. Then move to
RETURN_TO_LINE_STATE

    if (gBoxSize == 1) // Large
    {
      pickUpModeLarge();
      delay(200);
      moveServosTogether(93, 85, 138);
      delay(200);
    }
    if (gBoxSize == 2) // Small
    {
      pickUpModeSmall();
      delay(200);
      moveServosTogether(93, 85, 138);
      delay(200);
    }

    stowModeEmpty();
    gDropOffStatus = 1;


    currentState = RETURN_TO_LINE_STATE;
```

```
    break;
  }

  //----------- RETURN_TO_LINE_STATE -----------//
  case RETURN_TO_LINE_STATE:
  {
    // The vehicle will move backwards until the first return line is
found. It will then move to
    //the center of rotation and turn to start following the line again.

    int crossFound = 0;
    int lineFound = 0;

    moveBWD(70);
    delay(500);

    while (!crossFound)
    {
      int leftColor = classifyColor(2);
      int rightColor = classifyColor(3);

      if ((leftColor == gBoxColor) || (rightColor == gBoxColor))
      {
        stop();
        moveFWD(70);
        delay(1500);
        crossFound = 1;
      }

    }

    if (gBoxSize == 1)
    {
      gyroTurn("RIGHT", 87);

    }

    if (gBoxSize == 2)
    {
      gyroTurn("LEFT", 87);
    }
```

```
    moveFWD(70);

    while(!lineFound)
    {
      int leftColor = classifyColor(2);
      int rightColor = classifyColor(3);

      if ((leftColor == gBoxColor) || (rightColor == gBoxColor))
      {
        if (gBoxSize == 1)
        {
          delay(1500);
          gyroTurn("RIGHT", 87);

        }

        if (gBoxSize == 2)
        {
          delay(1500);
          gyroTurn("LEFT", 87);
        }

        lineFound = 1;
        stop();
        currentState = FOLLOW_LINE_STATE;

      }

    }

    break;
  }
  //--------------- NAVIGATE_STATE ---------------//
  case NAVIGATE_STATE:
  {
    // Move forward and check for walls or the finish line by using the
front ultrasonic sensor
    // and the color sensors.

    moveFWD(90);
    // Read front distance once
    int distanceFront = readUltrasonicDistance(TRIG1, ECHO1);
```

```
    // Check for wall ahead
    if (distanceFront <= 15) {
      stop();
      delay(100);
      // Confirm with a second reading to reduce false triggers
      distanceFront = readUltrasonicDistance(TRIG1, ECHO1);
      if (distanceFront <= 15) {
        Serial.println("Wall detected");
        currentState = WALL_FOUND_STATE;
        break;
      }
    }

    // Check finish line via side color sensors
    int color2 = classifyColor(2);
    int color4 = classifyColor(3);

    if (color2 == 4 || color4 == 4) {
      stop();
      currentState = FINISH_LINE_STATE;
      break;
    }

    // Brief delay to avoid sensor saturation
    delay(50);
    break;
  }

  //--------------- WALL_FOUND_STATE ---------------//
  case WALL_FOUND_STATE:
  {
    // The vehicle will stop and turn left then currentState will move to
NAVIGATION_STATE

    stop();
    delay(500);
    gyroTurn("LEFT", 87);

    currentState = FOLLOW_RIGHT_WALL_STATE;
    break;
  }
```

```
  //--------------- FOLLOW_RIGHT_WALL_STATE ---------------//
  case FOLLOW_RIGHT_WALL_STATE:
  {
    // The vehicle moves foward until an opening on the right side is
sensed by the
    // right ultrasonic sensor, or the boundry is detected by the color
sensors. If an
    // opening is found, the vehicle moves to RIGHT_OPENING_STATE. If the
boundery is
    // found, the system moves to COURSE_EDGE_STATE.

    moveFWD(90);

    int distanceRight = readUltrasonicDistance(TRIG3, ECHO3);

    if (distanceRight > 30)
    {
      distanceRight = readUltrasonicDistance(TRIG3, ECHO3);
      if (distanceRight > 30)
      {
        currentState = RIGHT_OPENING_STATE;
      }
    }

    else if ((classifyColor(2) == 1) && (classifyColor(3) == 1))
    {
      stop();
      delay(500);
      moveFWD(70);
      delay(150);
      if ((classifyColor(2) == 1) && (classifyColor(3) == 1))
      {
        currentState = COURSE_EDGE_STATE;
      }
    }
    break;
  }

  //--------------- FOLLOW_left_WALL_STATE ---------------//
  case FOLLOW_LEFT_WALL_STATE:
  {
    // The vehicle moves foward until an opening on the left side is sensed
by the
```

```
    // left ultrasonic sensor, or the boundry is detected by the color
sensors. If an
    // opening is found, the vehicle moves to LEFT_OPENING_STATE. If the
boundery is
    // found, the system moves to COURSE_EDGE_STATE.

    moveFWD(90);
    int distanceLeft = readUltrasonicDistance(TRIG2, ECHO2);

    if (distanceLeft > 30)
    {
      distanceLeft = readUltrasonicDistance(TRIG3, ECHO3);
      if (distanceLeft > 30)
      {
        currentState = LEFT_OPENING_STATE;
      }
    }

    else if ((classifyColor(2) == 1) && (classifyColor(3) == 1))
    {
      if ((classifyColor(2) == 1) && (classifyColor(3) == 1))
      {
        currentState = COURSE_EDGE_STATE;
      }
    }
    break;
  }

  //--------------- COURSE_EDGE_STATE ---------------//
  case COURSE_EDGE_STATE:
  {
    // The vehicle will stop at the white line, turn around, then the
system will check for a wall
    // using the side ultrasonic sensors and move to either
FOLLOW_RIGHT_WALL_STATE or
    // FOLLOW_LEFT_WALL_STATE.

    stop();
    delay(1000);
    moveBWD(70);
    delay(2000);
    stop();
```

```
    gyroTurn("LEFT", 87);
    delay(1000);
    gyroTurn("LEFT", 87);

    if (readUltrasonicDistance(TRIG2, ECHO2) < 30)
    {
      currentState = FOLLOW_LEFT_WALL_STATE;
    }

    else
    {
      currentState = FOLLOW_RIGHT_WALL_STATE;
    }

    break;
  }

  //--------------- RIGHT_OPENING_STATE ---------------//
  case RIGHT_OPENING_STATE:
  {
    // The vehicle will move foward so that the center of rotation is
centered at the opening,
    // then the vehicle will do a right turn and the vehicle will move to
NAVIGATE_STATE.

    moveFWD(90);
    delay(1700);

    gyroTurn("RIGHT", 87);

    currentState = NAVIGATE_STATE;

    break;
  }

  //--------------- LEFT_OPENING_STATE ---------------//
  case LEFT_OPENING_STATE:
  {
    // The vehicle will move foward so that the center of rotation is
centered at the opening,
    // then the vehicle will do a left turn and the system will move to
NAVIGATE_STATE.
```

```
    moveFWD(90);
    delay(1700);

    gyroTurn("LEFT", 87);

    currentState = NAVIGATE_STATE;

    break;
  }

  //---------------FINISH_LINE_STATE ---------------//
  case FINISH_LINE_STATE:
  {
    // The vehicle has found the finish line of the obstacle course, and
will stop moving and move to IDLE_STATE

    stop();
    delay(1000);
    waveMode();
    delay(200);
    stowModeEmpty();

    currentState = IDLE_STATE;

    break;
  }
}
}
```

Figure 41: Main loop implementing the robot's state machine for task coordination.

```
#ifndef ULTRASONICCONTROL_H
#define ULTRASONICCONTROL_H

//--------------------------Ultrasonic Sensor Macros--------------------
#define TRIG1 27
#define ECHO1 28

#define TRIG2 29
#define ECHO2 30
```

```
#define TRIG3 31
#define ECHO3 32

// -------- Function to read distance from an ultrasonic sensor --------
void initUltrasonics();                 // Initializes ultrasonic sensor
pins
long readUltrasonicDistance(int trigPin, int echoPin);

#endif
```

Figure 42: Function declarations and constants for Ultrasonic Sensor control modules

```
#include "UltrasonicControl.h"
#include <Arduino.h>

void initUltrasonics() {
  pinMode(TRIG1, OUTPUT);
  pinMode(ECHO1, INPUT);
  pinMode(TRIG2, OUTPUT);
  pinMode(ECHO2, INPUT);
  pinMode(TRIG3, OUTPUT);
  pinMode(ECHO3, INPUT);
}

// Generic function to read ultrasonic distance
long readUltrasonicDistance(int trigPin, int echoPin)
{
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);

  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);

  long duration = pulseIn(echoPin, HIGH);
  long distance = duration * 0.0343 / 2;  // convert time to distance (cm)

  delay(10);

  return distance;
```

```
}
```

Figure 43: Ultrasonic distance sensing and initialization

```
#ifndef SERVOCONTROL_H
#define SERVOCONTROL_H

#include <Servo.h>

// -------- Servo Pin Definitions --------
#define Servo1Pin 24
#define Servo2Pin 25
#define Servo3Pin 26

// -------- Servo Objects --------
extern Servo Servo1;
extern Servo Servo2;
extern Servo Servo3;

// -------- Functions --------
void initServos();
void moveServosTogether(int target1, int target2, int target3, int delayMs
= 15);
void stowModeEmpty();
void stowModeLarge();
void stowModeSmall();
void extendMode();
void pickUpModeLarge();
void pickUpModeSmall();
void colorReadModeLarge();
void colorReadModeSmall();
void waveMode();

#endif
```

Figure 44: Function declarations and constants for Servo control modules

```
#include "ServoControl.h"
```

```
#include <Arduino.h>
#include <math.h>

// Declare Servo objects for controlling three servos
Servo Servo1, Servo2, Servo3;

// Initializes the servo motors by attaching them to the defined pins
// and setting Servo3 to an initial position
void initServos() {
  Servo1.attach(Servo1Pin);  // Attach Servo1 to its control pin
  Servo2.attach(Servo2Pin);  // Attach Servo2 to its control pin
  Servo3.attach(Servo3Pin);  // Attach Servo3 to its control pin

  Servo3.write(138);         // Set initial position for Servo3
  delay(500);                // Allow time for movement
}

// Moves all three servos simultaneously from current positions to target
positions
// by interpolating intermediate steps with a specified delay between each
step
void moveServosTogether(int target1, int target2, int target3, int delayMs)
{
  int pos1 = Servo1.read(); // Get current position of Servo1
  int pos2 = Servo2.read(); // Get current position of Servo2
  int pos3 = Servo3.read(); // Get current position of Servo3

  // Calculate the maximum number of steps needed based on the largest
movement
  int steps = max(max(abs(target1 - pos1), abs(target2 - pos2)),
abs(target3 - pos3));
  if (steps == 0) return; // Exit if no movement needed

  // Calculate the incremental step values for each servo
  float step1 = (float)(target1 - pos1) / steps;
  float step2 = (float)(target2 - pos2) / steps;
  float step3 = (float)(target3 - pos3) / steps;

  // Incrementally move each servo to its target position
  for (int i = 0; i <= steps; i++) {
    Servo1.write(round(pos1 + step1 * i));
    Servo2.write(round(pos2 + step2 * i));
    Servo3.write(round(pos3 + step3 * i));
```

```
    delay(delayMs); // Delay between each step for smooth motion
  }
}

// Moves arm to stowed position after releasing or before gripping a box
// Configuration for when the gripper is empty
void stowModeEmpty() {
  moveServosTogether(50, 41, 138);
}

// Configuration for stowing after gripping a large box
void stowModeLarge() {
  moveServosTogether(50, 41, 112);
}

// Configuration for stowing after gripping a small box
void stowModeSmall() {
  moveServosTogether(50, 41, 103);
}

// Fully extends the arm to reach out
void extendMode() {
  moveServosTogether(80, 85, 103);
}

// Moves servos into position to grip a large box
void pickUpModeLarge() {
  moveServosTogether(93, 85, 112);
}

// Moves servos into position to grip a small box
void pickUpModeSmall() {
  moveServosTogether(93, 85, 103);
}

// Positions arm to allow front color sensor to read color of large box
void colorReadModeLarge() {
  moveServosTogether(82, 16, 112);
}

// Positions arm to allow front color sensor to read color of small box
void colorReadModeSmall() {
  moveServosTogether(82, 16, 103);
```

```
}

// Performs a waving animation by quickly oscillating Servo2
void waveMode()
{
  moveServosTogether(50, 46, 138); // Wave position 1
  delay(100);
  moveServosTogether(50, 32, 138); // Wave position 2
  delay(100);
  moveServosTogether(50, 46, 138); // Repeat
  delay(100);
  moveServosTogether(50, 32, 138);
  delay(100);
}
```

Figure 45: Servo control functions for arm positioning and box handling

```
#ifndef MOTORCONTROL_H
#define MOTORCONTROL_H

// -------- Motor Pin Definitions --------
#define FENA 18
#define FENB 19
#define FIN1 13
#define FIN2 14
#define FIN3 15
#define FIN4 39

#define BENA 22
#define BENB 23
#define BIN1 8
#define BIN2 9
#define BIN3 10
#define BIN4 11

// -------- Global PWM Variables --------
extern volatile int PWMval;
extern volatile int PWMval2;

// -------- Motor Control Functions --------
void initMotors();
```

```
void moveFWD(int speed);
void moveBWD(int speed);
void moveLeft(int speed);
void moveRight(int speed);
void stop();

#endif
```

Figure 46: Function declarations and constants for Motor control modules

```
#include "MotorControl.h"
#include <Arduino.h>

// Global PWM values for motor speed control
volatile int PWMval = 0;   // Main PWM duty cycle value
volatile int PWMval2 = 0;  // Secondary PWM value used for slight
directional bias

// Initializes all motor-related pins and stops motors initially
void initMotors()
{
  // Enable pins for front and back motors
  pinMode(FENA, OUTPUT);
  pinMode(FENB, OUTPUT);
  pinMode(BENA, OUTPUT);
  pinMode(BENB, OUTPUT);

  // Direction control pins for front motors
  pinMode(FIN1, OUTPUT);
  pinMode(FIN2, OUTPUT);
  pinMode(FIN3, OUTPUT);
  pinMode(FIN4, OUTPUT);

  // Direction control pins for back motors
  pinMode(BIN1, OUTPUT);
  pinMode(BIN2, OUTPUT);
  pinMode(BIN3, OUTPUT);
  pinMode(BIN4, OUTPUT);

  stop();  // Ensure all motors are stopped on startup
}
```

```
// Stops all motors by setting PWM to 0 and disabling all direction control
pins
void stop() {
  analogWrite(FENA, 0);
  analogWrite(FENB, 0);
  analogWrite(BENA, 0);
  analogWrite(BENB, 0);

  digitalWrite(FIN1, LOW);
  digitalWrite(FIN2, LOW);
  digitalWrite(FIN3, LOW);
  digitalWrite(FIN4, LOW);

  digitalWrite(BIN1, LOW);
  digitalWrite(BIN2, LOW);
  digitalWrite(BIN3, LOW);
  digitalWrite(BIN4, LOW);
}

// Moves robot forward at a given speed (0-100%)
void moveFWD(int speed) {
  PWMval = 255 * speed / 100;  // Convert % speed to 0-255 PWM range

  // Set motor enable pins to speed value
  analogWrite(FENA, PWMval);
  analogWrite(FENB, PWMval);
  analogWrite(BENA, PWMval);
  analogWrite(BENB, PWMval);

  // Set direction pins for forward movement
  digitalWrite(FIN1, HIGH);
  digitalWrite(FIN2, LOW);
  digitalWrite(FIN3, LOW);
  digitalWrite(FIN4, HIGH);

  digitalWrite(BIN1, HIGH);
  digitalWrite(BIN2, LOW);
  digitalWrite(BIN3, LOW);
  digitalWrite(BIN4, HIGH);
}

// Moves robot backward at a given speed (0-100%)
void moveBWD(int speed) {
```

```
  PWMval = 255 * speed / 100;

  analogWrite(FENA, PWMval);
  analogWrite(FENB, PWMval);
  analogWrite(BENA, PWMval);
  analogWrite(BENB, PWMval);

  // Set direction pins for reverse movement
  digitalWrite(FIN1, LOW);
  digitalWrite(FIN2, HIGH);
  digitalWrite(FIN3, HIGH);
  digitalWrite(FIN4, LOW);

  digitalWrite(BIN1, LOW);
  digitalWrite(BIN2, HIGH);
  digitalWrite(BIN3, HIGH);
  digitalWrite(BIN4, LOW);
}

// Turns the robot right by biasing speed and setting opposite wheel
directions
void moveRight(int speed) {
  PWMval = 255 * speed / 100;
  PWMval2 = 255 * (speed + 10) / 100;  // Slightly faster on one set for
turning

  analogWrite(FENA, PWMval2);
  analogWrite(FENB, PWMval);
  analogWrite(BENA, PWMval2);
  analogWrite(BENB, PWMval);

  // Set direction pins for a right turn
  digitalWrite(FIN1, HIGH);
  digitalWrite(FIN2, LOW);
  digitalWrite(FIN3, HIGH);
  digitalWrite(FIN4, LOW);

  digitalWrite(BIN1, LOW);
  digitalWrite(BIN2, HIGH);
  digitalWrite(BIN3, LOW);
  digitalWrite(BIN4, HIGH);
}
```

```
// Turns the robot left by biasing speed and setting opposite wheel
directions
void moveLeft(int speed) {
  PWMval = 255 * speed / 100;
  PWMval2 = 255 * (speed + 10) / 100;

  analogWrite(FENA, PWMval);
  analogWrite(FENB, PWMval2);
  analogWrite(BENA, PWMval);
  analogWrite(BENB, PWMval2);

  // Set direction pins for a left turn
  digitalWrite(FIN1, LOW);
  digitalWrite(FIN2, HIGH);
  digitalWrite(FIN3, LOW);
  digitalWrite(FIN4, HIGH);

  digitalWrite(BIN1, HIGH);
  digitalWrite(BIN2, LOW);
  digitalWrite(BIN3, HIGH);
  digitalWrite(BIN4, LOW);
}
```

Figure 47: Motor control functions for directional movement

```
#ifndef GYROCONTROL_H
#define GYROCONTROL_H

#include <Adafruit_MPU6050.h>
#include <Adafruit_Sensor.h>

// -------- Gyro Object and Variables --------
extern Adafruit_MPU6050 mpu;

extern float angleZ;
extern float gyroZ_bias;
extern unsigned long lastTime;

// -------- Functions --------
void initGyro();  // Initializes MPU6050 and performs calibration
void calibrateGyroZ();
```

```
void resetGyro();
void updateGyroAngle();
void gyroTurn(String direction, float targetDeg, int speed = 70);

#endif
```

Figure 48: Function declarations and constants for Gyro control modules

```
#include "GyroControl.h"
#include <Wire.h>                    // For I2C communication
#include <Adafruit_MPU6050.h>       // MPU6050 sensor library
#include <Adafruit_Sensor.h>        // Required for sensor event objects
#include <Arduino.h>
#include "MotorControl.h"           // For motor movement functions

// Create MPU6050 object
Adafruit_MPU6050 mpu;

// Variables to track gyro rotation angle and bias
float angleZ = 0.0;                 // Integrated Z-axis angle (in degrees)
float gyroZ_bias = 0.0;            // Bias to correct for gyro drift
unsigned long lastTime = 0;        // Time of last gyro update

// Initializes the gyroscope (MPU6050) and calibrates Z-axis gyro
void initGyro() {
  Wire1.begin();                   // Start I2C on Wire1

  // Try to connect to the MPU6050 sensor
  if (!mpu.begin(MPU6050_I2CADDR_DEFAULT, &Wire1)) {
    Serial.println("MPU6050 not detected on Wire1!");
    while (1) delay(10);          // Halt if not detected
  }

  Serial.println("MPU6050 connected.");

  // Configure sensor ranges and filtering
  mpu.setAccelerometerRange(MPU6050_RANGE_8_G);      // ±8g range
  mpu.setGyroRange(MPU6050_RANGE_250_DEG);           // ±250°/s range
  mpu.setFilterBandwidth(MPU6050_BAND_21_HZ);        // Noise filtering
  delay(1000);                                        // Wait for sensor to
stabilize
```

```
  calibrateGyroZ();  // Measure average gyro offset
}

// Calibrates Z-axis gyro by averaging multiple samples while stationary
void calibrateGyroZ() {
  float sum = 0;
  sensors_event_t a, g, temp;

  // Take 100 samples of Z-axis gyro data
  for (int i = 0; i < 100; i++) {
    mpu.getEvent(&a, &g, &temp);
    sum += g.gyro.z;  // Z-axis angular velocity in rad/s
    delay(5);         // Small delay between samples
  }

  gyroZ_bias = sum / 100.0;  // Store average bias
  Serial.print("Gyro Z bias: ");
  Serial.println(gyroZ_bias, 6);  // Print with 6 decimal places
}

// Resets the integrated angle and timing
void resetGyro() {
  angleZ = 0.0;
  lastTime = millis();
}

// Updates the integrated Z angle by reading gyro and computing angular
displacement
void updateGyroAngle() {
  sensors_event_t a, g, temp;
  mpu.getEvent(&a, &g, &temp);  // Get latest sensor data

  unsigned long now = millis();        // Current time
  float dt = (now - lastTime) / 1000.0; // Convert time delta to seconds
  lastTime = now;

  // Convert gyro reading from rad/s to deg/s and subtract bias
  float degPerSec = (g.gyro.z - gyroZ_bias) * (180.0 / PI);

  angleZ += degPerSec * dt;  // Integrate to get angle
}
```

```
// Performs a turn using the gyroscope until reaching a target degree angle
// dir: "LEFT" or "RIGHT", targetDeg: degrees to turn, speed: motor speed
(0-100)
void gyroTurn(String dir, float targetDeg, int speed) {
  resetGyro();                          // Clear previous angle and start
time
  unsigned long startTime = millis();   // For timeout safety

  // Continue turning until target angle reached or timeout
  while (abs(angleZ) < targetDeg && millis() - startTime < 5000) {
    updateGyroAngle();  // Update angle reading

    if (dir == "LEFT") moveLeft(speed);     // Turn left
    else if (dir == "RIGHT") moveRight(speed); // Turn right

    delay(5);  // Short delay to allow time for motors to act and sensors
to update
  }

  stop();       // Stop all motors once target reached or timeout
  delay(300);   // Allow robot to settle after stopping
}
```

Figure 49: Gyroscope initialization, calibration, and turning logic

```
#ifndef COLORCONTROL_H
#define COLORCONTROL_H

// -------- Shared Frequency Scaling Pins --------
#define S0 33
#define S1 34

// -------- Color Sensor Pin Definitions --------
#define S2_1 35
#define S3_1 36
#define OUT_1 2

#define S2_2 37
#define S3_2 38
#define OUT_2 3
```

```
#define S2_4 41
#define S3_4 40
#define OUT_4 5

// -------- Color Codes --------
// 0 = Black, 1 = White, 2 = Blue, 3 = Red, 4 = Green, -1 = Unknown

// -------- Struct for Pin Mapping --------
struct ColorSensorPins {
  int s2;
  int s3;
  int out;
};

// -------- Functions --------
void initColorSensors();
int classifyColor(int sensorNumber);
int readColorComponent(int s2, int s3, int out, bool s2State, bool
s3State);

#endif
```

Figure 50: Function declarations and constants for Color control modules

```
#include "ColorControl.h"
#include <Arduino.h>

// Array of structs holding the S2, S3, and OUT pins for each sensor
ColorSensorPins sensorPins[] = {
  {0, 0, 0},                    // Dummy 0-index (so indexing starts at 1)
  {S2_1, S3_1, OUT_1},          // Sensor 1 pin configuration
  {S2_2, S3_2, OUT_2},          // Sensor 2 pin configuration
  {S2_4, S3_4, OUT_4}           // Sensor 4 pin configuration
};

// Initializes color sensors and sets up control and output pins
void initColorSensors() {
  pinMode(S0, OUTPUT);
  pinMode(S1, OUTPUT);

  // Set frequency scaling for TCS230/TCS3200 sensors to 100%
```

```
  digitalWrite(S0, HIGH);
  digitalWrite(S1, LOW);  // LOW for 100% scaling

  // Configure sensor 1 pins
  pinMode(S2_1, OUTPUT);
  pinMode(S3_1, OUTPUT);
  pinMode(OUT_1, INPUT);

  // Configure sensor 2 pins
  pinMode(S2_2, OUTPUT);
  pinMode(S3_2, OUTPUT);
  pinMode(OUT_2, INPUT);

  // Configure sensor 4 pins
  pinMode(S2_4, OUTPUT);
  pinMode(S3_4, OUTPUT);
  pinMode(OUT_4, INPUT);
}

// Utility function to check if a value is within a tolerance of a target
bool inRange(int val, int target, int tol) {
  return abs(val - target) <= tol;
}

// Reads the pulse value for a specific color component from a sensor
// s2 and s3: color filter selection pins
// out: signal pin for the frequency-based output
// s2State, s3State: logic level for color selection (based on datasheet)
int readColorComponent(int s2, int s3, int out, bool s2State, bool s3State)
{
  digitalWrite(s2, s2State);
  digitalWrite(s3, s3State);
  delay(10);  // Wait for filter settings to stabilize
  return pulseIn(out, LOW, 10000);  // Measure duration of LOW pulse
(timeout = 10ms)
}

// Classifies a color reading based on pre-defined thresholds
// sensorIndex = 1 (Sensor 1), 2 (Sensor 2), 3 (Sensor 4)
// Returns: 2 = Blue, 3 = Red, 4 = Green, -1 = Unknown
int classifyColor(int sensorIndex) {
  // Check if index is valid
  if (sensorIndex < 1 || sensorIndex > 3) return -1;
```

```
  // Get pin config for the selected sensor
  ColorSensorPins p = sensorPins[sensorIndex];

  // Read red, green, and blue components by setting filter selection
  int r = readColorComponent(p.s2, p.s3, p.out, LOW, LOW);    // Red
  int g = readColorComponent(p.s2, p.s3, p.out, HIGH, HIGH);  // Green
  int b = readColorComponent(p.s2, p.s3, p.out, LOW, HIGH);   // Blue

  switch (sensorIndex) {

    // ------------------------------------
    // Sensor 1 - Detects RED and BLUE only
    // ------------------------------------
    case 1:
      // RED: R low, G & B higher
      if (r < 200 && g > 100 && b > 90) return 3;

      // BLUE: R high, G moderate, B lower
      if (r > 250 && g > 100 && b > 70 && r > g && r > b) return 2;
      break;

    // ------------------------------------
    // Sensor 2 - RGB classification
    // ------------------------------------
    case 2:
      // RED: low R, high G/B
      if (r >= 60 && r <= 100 && g >= 170 && g <= 280 && b >= 140 && b <=
240) return 3;

      // GREEN: all components lower
      if (r >= 70 && r <= 120 && g >= 60 && g <= 100 && b >= 90 && b <=
140) return 4;

      // BLUE: R mid-high, G/B lower
      if (r >= 180 && r <= 320 && g >= 90 && g <= 160 && b >= 60 && b <=
100) return 2;

      break;

    // ------------------------------------
    // Sensor 4 - RGB classification
    // ------------------------------------
```

```
    case 3:
      // RED: R medium, G/B high
      if (r >= 110 && r <= 170 && g >= 500 && g <= 750 && b >= 370 && b <=
540) return 3;

      // GREEN: R mid, G low-mid, B high
      if (r >= 140 && r <= 210 && g >= 110 && g <= 150 && b >= 170 && b <=
260) return 4;

      // BLUE: high R, mid G, mid-low B
      if (r >= 650 && r <= 900 && g >= 220 && g <= 340 && b >= 110 && b <=
180) return 2;

      break;
  }

  return -1;  // Return -1 if no valid match
}
```

Figure 51: Color sensor setup and RGB classification logic

References

ServoCity. *ServoCity*, https://www.servocity.com/. Accessed 1 June 2025.

AliExpress. "DC 6V 12V Micro Linear Actuator Motor Stroke 30mm 50mm 100mm 150mm 200mm 250mm 300mm Electric Mini Push Rod 150N 50N 100N." *AliExpress*, https://www.aliexpress.us/item/3256807324908583.html. Accessed 1 June 2025.

AliExpress. "TCRT5000 Infrared Reflective Sensor Module." *AliExpress*, https://www.aliexpress.us/item/3256807103847839.html. Accessed 1 June 2025.

MaxBotix. "How Ultrasonic Sensors Work." *MaxBotix*, https://maxbotix.com/blogs/blog/how-ultrasonic-sensors-work. Accessed 2 June 2025.

ROHM. "Color Sensor – Operating Principle." *Electronics Basics*, ROHM Semiconductor, https://www.rohm.com/electronics-basics/sensors/sensor_what7#:~:text=Color%20Sensor%20%2D%20Operating%20Principle,light%20component%20will%20be%20red.