EG2310

Fundamentals of Systems Design

G2 Report

Group 12

Raymand Tey Jia Cheng A0258990L

Hassan Bin Azhar A0265149X

Dwivedi Arshita A0266808R

Dileep Pillai Vaibhav A0266793L

Zhu Fangze A0232367B

Table of Contents

1. Problem Definition

We are tasked to create a food delivery system for a restaurant that can automatically dispense and deliver a can of soda to a table. The restaurant will have at least 6 tables. The system shall consist of a Dispenser as well as a Delivery bot.

The Dispenser shall store at least one can of soda that is manually inserted by the user after which the user enters the table number to which the Delivery bot should deliver the can. Upon confirmation that the Delivery bot has docked into the Dispenser, the Dispenser should load the can into the Delivery bot. The Delivery bot should not move while the can is being loaded. Moreover, the Dispenser should be able to take delivery orders while the Delivery bot is working on its previous delivery.

After successful loading, the Delivery bot shall navigate through the restaurant to the designated table while avoiding collision with other tables and walls. At the designated table, the Delivery bot shall wait until the user picks up the can of soda after which the Delivery bot shall return to the Dispenser to dock itself.

2. Literature Review

2.1 Dispenser's Storage System

2.1.1 Two-Tier Storage System

The Two-Tier Storage System utilises gravity to pack cans at the bottom, allowing for multiple cans to be stored in a compact space. However, one limitation of this system is the lack of control over the release of the cans. To address this limitation, we could modify the Two-Tier Storage System by incorporating a Can Release System consisting of two gates. This system will allow for the controlled release of cans, enabling the Delivery bot to efficiently retrieve and transport them one at a time.

2.1.2 Single Can Storage System

The Single Can Storage System also utilises gravity to roll a single can onto the storage compartment of the Delivery bot. This system only works with one can at a time and hence uses only a single gate. This system comprises a short, slightly declined ramp with a Can Release System towards the end of this ramp.

2.1.3 Decision and Rationale

Although the Two-Tier System allows for greater scalability as it is possible to store multiple cans, it is more complex to implement as it operates on multiple gates working synchronously to control the movement of cans onto the storage of the Delivery bot. Thus, we have settled on the use of a Single Can Storage System as it only uses a single gate which reduces costs and complexity whilst meeting the stakeholder requirements.

2.2 Can Release System

2.2.1 Flaps System

The Flaps System is a mechanism that employs flaps or doors to control the release of soda can. This system features a horizontal flap located towards the end of the ramp of the storage system and it is operated via a servo to release the soda can to the Delivery bot when the command is given.

2.2.2 Barrier System

The Barrier System releases the can via a ‘wall’ that is pushed up and down by the rack and pinion set. The pinion would be attached to a DC motor. Similar to the flap system, it is located towards the end of the ramp of the storage system.

2.2.3 Decision and Rationale

We have decided to use a Flap System as the components that make up said system are far simpler than a Barrier System and therefore less prone to mechanical failure. To illustrate, the Barrier System’s gears may become misaligned if the can strikes the barrier with a large enough force. Furthermore, the gears need to be lubricated from time to time.

2.3 Delivery Bot Storage System (Container)

2.3.1 Rhombus Shaped Storage Design

There would be an open cuboid installed on the front side of the Delivery bot onto which the soda can would be placed horizontally. It will be installed on an extended base of the Delivery bot to keep the centre of gravity as low as possible. Additionally, it will possess a slope on the front end to ensure a smooth descent of the soda can from the Dispenser to the Delivery bot. Space would be provided in the cuboid along the length of the soda can to ensure a comfortable grip to boost user convenience and give allowance for any potential errors in docking procedure.

2.3.2 Decision and Rationale

We chose this design as it is simple and easy to build. Additionally, it makes it easier for the Dispenser to release the soda can into this design than if it was, for example, into one which would store the soda can vertically.

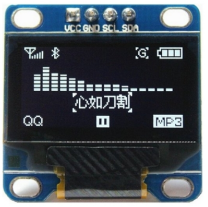
2.4 Dispensers' Interface System

2.4.1 1602 Character LCD03 ESP32 Display Module with Buttons

The interface system would consist of an LCD Display Module onto which prompts like "Please choose the table number" or "Please confirm order" would be displayed to the user. The user can then interact and choose options using the 4 buttons that would be placed below this display. The 4 buttons would be Cancel, Previous, Next, and Enter in that particular order from left to right.

2.4.2 Arduino OLED 4 pin I2C with Buttons (SSD1306)

Similar to the LCD Display Module, the Arduino OLED 4 pin I2C with Front Panel Integration is a compact display with a 4 pin communication device with front panel integration. It does not have backlight and uses OLED technology making it more power efficient. It is able to display dynamic text and is easy to code when used in conjunction with an Arduino board.



2.4.3 Decision and Rationale

The Arduino OLED with buttons was chosen as the interface for the Dispenser because it is more power efficient and user-friendly. The keyboard allows for straightforward input of table numbers and the OLED display provides real-time confirmation and visual feedback. Furthermore, there is more documentation available online as well as past projects of similar nature, making it easier to troubleshoot.

2.5 Mapping System

2.5.1 Cartographer SLAM

Cartographer is a library for real-time simultaneous localization and mapping (SLAM) in 2D and 3D. It is implemented as a set of ROS packages and uses a variety of sensor data such as LIDAR, Inertial Measurement Unit (IMU), and odometry to generate 2D and 3D maps of the environment. Cartographer uses advanced optimization and filtering techniques, such as submap-based loop closure detection and pose graph optimization, to improve the accuracy and robustness of the map. It also supports real-time operation and can be easily integrated with other systems.

2.5.2 Gmapping

Gmapping is a ROS package that uses laser rangefinder and odometry data to generate 2D occupancy grid maps of the environment. It is based on the GridSLAM algorithm, which uses a Rao-Blackwellized particle filter (RBPF) to track the bot's pose and simultaneously build a map of the environment. The algorithm is called "GMapping" because it uses a grid-based representation of the map. The maps generated by Gmapping are 2D grid maps. It is

relatively simple to use and understand, but it may not be as accurate or robust as other SLAM algorithms.

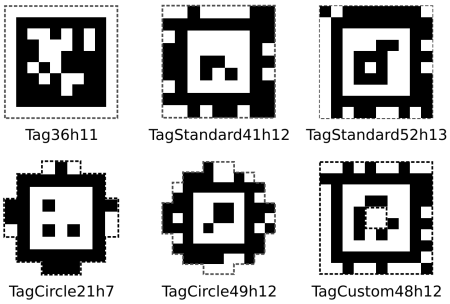
2.5.3 Decision and Rationale

Cartographer is considered a more advanced package for SLAM compared to Gmapping as it uses advanced optimization techniques, can handle a variety of sensor configurations, and is more flexible. It is also more accurate and robust than Gmapping. Furthermore, Cartographer is already implemented as a launch file for turtlebot3, hence, it has less development cost. Therefore, we decided to use Cartographer SLAM as it allows us to streamline the process and minimise the required effort.

2.6 Docking System

2.6.1 AprilTag

AprilTag utilises visual fiducial markers that allow camera calibration and thus in our project, alignment to the Dispenser. The markers can be made simply from a normal printer. Using the AprilTag detection software, accurate position and orientation of the tags relative to the camera can be calculated and pathing to the station will ensue. They are similar to QR codes as they are both 2D barcodes.



2.6.2 Line Following

An alternative for our AprilTag is to use line following for our docking system. The turtlebot will travel to a fixed location and scan for lines made with electrical tape on the floor. When it reaches the last waypoint, it will start scanning for lines on the floor using a pi camera and openCV. The bot will then use object detection to follow the line to reach the docking station and stop when it reaches the end of the line.

2.6.3 Decision and Rationale

Through our research, we discovered that there is more support for AprilTag [1] and its implementation via ROS for our specific project. For that reason, we have excluded Aruco markers despite Aruco being functionally similar. Additionally, the data that is encoded in the AprilTag markers is smaller than QR codes which results in lower processing power and can be detected at a further range [2]. Hence, we have also excluded QR codes from this system.

2.7 Loading and Unloading System

2.7.1 Limit Switch

A Limit Switch is a sensor that can detect when an object comes into contact with it. We would place it at the bottom of the container. When the soda can rolls onto the Delivery bot, the switch would be triggered. This would send a HIGH signal to the RPi indicating that the soda can has been received by the Delivery bot. When the can is taken by the user at the end of delivery, the signal changes back to LOW which would signal the RPi that the soda can is delivered.

2.7.2 Pressure Sensor

Pressure sensors are devices that are used to measure the force exerted on an object. A Pressure Sensor would be placed at the bottom of the Container to sense pressure when the soda can drops into it. The increasing spike in pressure from the weight of the soda can will let the RPi know that the soda can is in place. Similarly, the decreasing spike in pressure when the soda can is taken by the user will let the RPi know that the soda can is delivered.

2.7.3 Decision and Rationale

We have decided to go with the Limit Switch on the Delivery bot as it is cheaper to acquire and likely to give a lower chance of failure as it can adjust the pressure needed to trigger the switch. Furthermore, the exact measurements from the pressure sensor are unnecessary and require more computational power whereas the Limit Switch only checks whether the can is present.

2.8 Inter-Device Communication Protocols

2.8.1 Message Queuing Telemetry Transport (MQTT)

MQTT is a standard messaging protocol for IoT which relies on the publish and/or subscribe model to initiate communication between machines. MQTT implements the publish/subscribe model by defining clients and brokers. A client is any device that publishes or receives data through MQTT over a network. Conversely, a broker acts like a server that handles messages from clients and coordinates the flow of messages, ensuring that all clients that listen for a particular topic receive their intended message.

2.8.2 Decision and Rationale

MQTT is chosen as the communication protocol to facilitate data transfer between the Dispenser and the Delivery as this protocol allows for bi-directional communication. Furthermore, MQTT implementation requires minimal resources and hence it can be installed on small microcontrollers. It also requires a minimal amount of code to install and execute, which consumes little power during operations. Additionally, there are various

documentations available online for its implementation on various devices, such as RPi, Arduino and ESP32.

3. Concept of Operations

3.1 Overview of Phases

The entire operation has been divided into three distinct phases: the Interface phase, the Loading phase, and the Navigation phase.

The Interface phase is where the user inputs the table number via the keypad into the Dispenser. The Dispenser then communicates that location to the Delivery bot.

The Loading phase begins with the manual loading of the can onto the Dispenser and ends when the Delivery bot has received it. This phase includes both the Dispenser and Delivery bot working together.

The Navigation phase is where the Delivery bot navigates autonomously to the correct table while avoiding obstacles. The can is then removed from the Delivery bot, and the Delivery bot automatically goes back to the Dispenser to collect the next can and restart the process.

3.2 Overview of Systems

1. Mechanical
    i. Motion system (TurtleBot)
    ii. Can release system
2. Electrical
    i. Control system (RPi)
    ii. Control system (ESP32)
3. Software
    i. Interface system
    ii. Navigation system

3.3 Phase 1 (Interface Phase)

Objective of Phase: To allow the user to input the destination table's number.

Description of Phase: The Arduino OLED with Integrated Keyboard shall be turned on when the ESP32 is powered through micro-USB cable connected to a laptop. The keypad contains 12 buttons containing the numeric characters from 0-9 and an asterisk and hash key. The display shall prompt the user to select the table number, 1-6, which the user may input with the corresponding keys and use the hash key to confirm selection or asterisks to remove selection. If the delivery bot is docked at the Dispenser, a confirmation message, "Can is being delivered", should be displayed and . Else, the interface should display a "Waiting for Delivery bot" message until the delivery bot is detected.

3.4 Phase 2 (Loading Phase)

Objective of Phase: To facilitate the loading of the can onto the Container.

Description of Phase: In this phase, the user shall load the can into the Dispenser's Storage System. Upon completion of the Interface phase, the ESP32 in the Dispenser will transmit a signal to the servo to raise the barrier of the Can Release System when the table number has been inputted and the delivery bot is docked. This will result in the can rolling down the ramp into the Container. The weight of the can triggers the limit switch, leading to the RPi transmitting a signal to the ESP32 to lower the flaps of the Can Release System. Finally, the Delivery bot will start undocking and move on to the navigation phase.

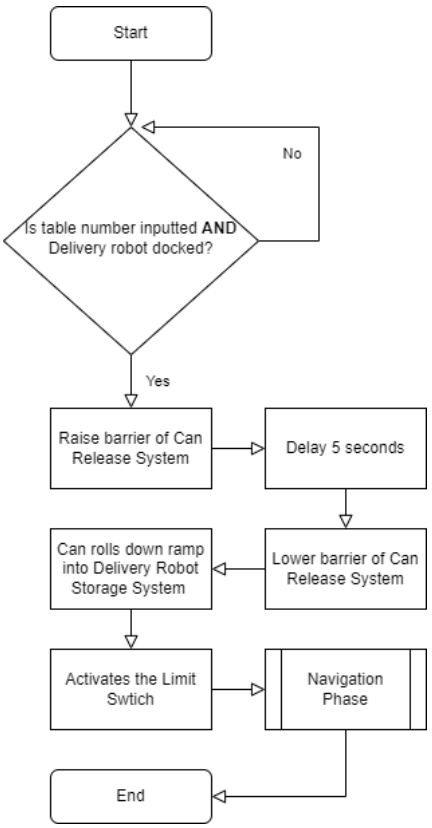


Fig.1: Flowchart representation of Loading Phase

3.5 Phase 3 (Navigation Phase)

Objective of Phase: To accomplish autonomous navigation to the designated location and successfully return to the docking station after delivery.

Description of Phase: We will assign an area with coordinates for each table of which the delivery bot will have to travel to to drop off the can. As the delivery bot travels, it will continue to reference the coordinates at regular intervals and its angular and linear velocity is updated to move towards the table. Upon reaching the table, the delivery bot will stop and wait until the can is removed by the customer. After removal of the can, the delivery bot shall travel back to the dispenser using the same path as before in reverse order. Upon nearing the dispenser, the delivery bot will turn on the pi camera to locate the AprilTag to dock accurately.

4. Post Design Review Preliminary Design

Following our Preliminary Design Review, the **only major change** needed was to redesign the container. However, there were a few other minor changes made as well.

4.1 Major Change - Container Design

Initially, the five plates of the container had a square-ended butt joint to hold them in place together. In order to use acrylic glue to ensure strong and firmly held perpendicularly placed plates, we decided to increase surface area by cutting the container plates to form interlocking joints (dovetail joints). Additionally, to accommodate the change in limit switch, the dimension of the slot provided for it had to be changed.

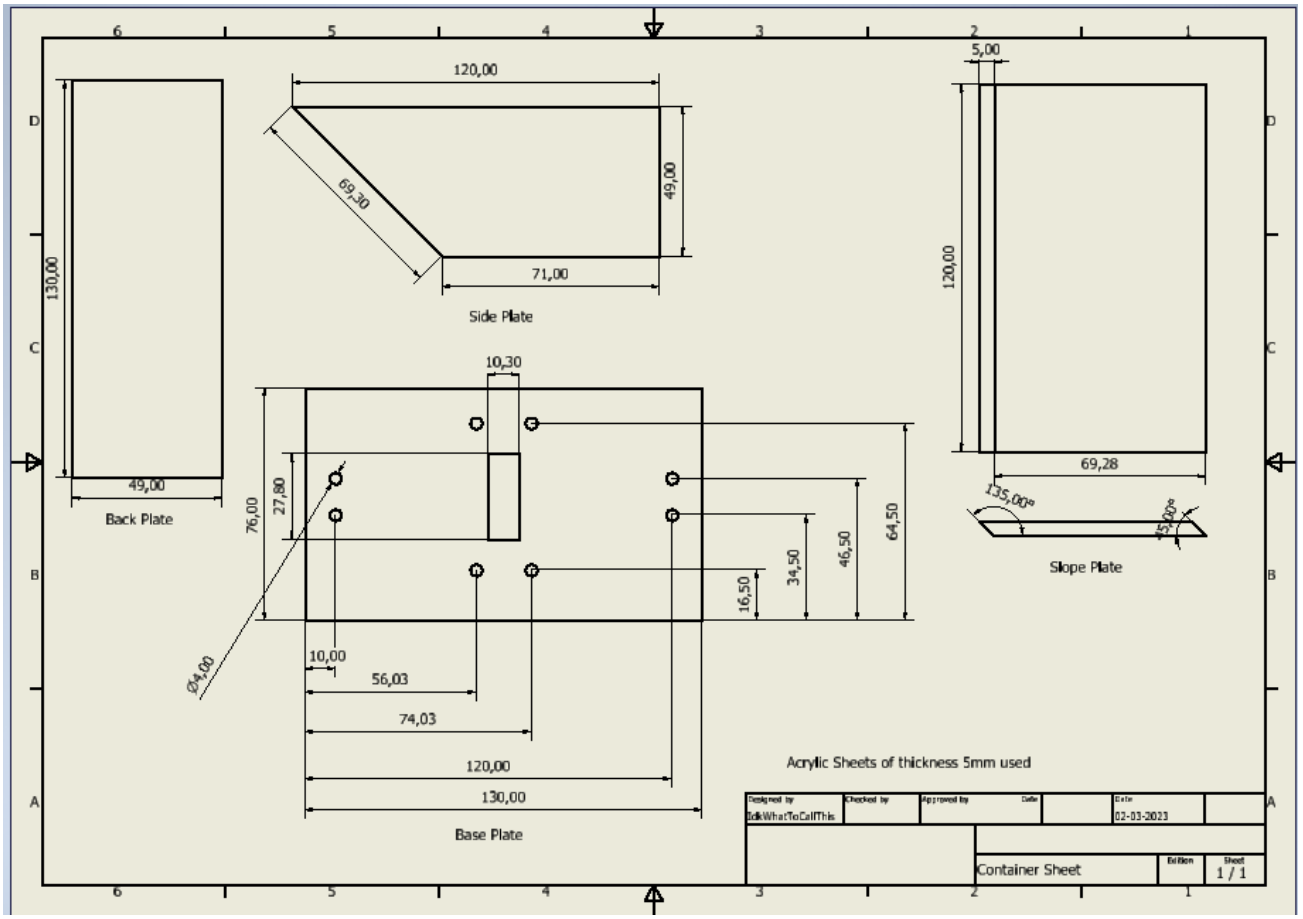


Fig. 2: Initial Container Design's Engineering Drawing

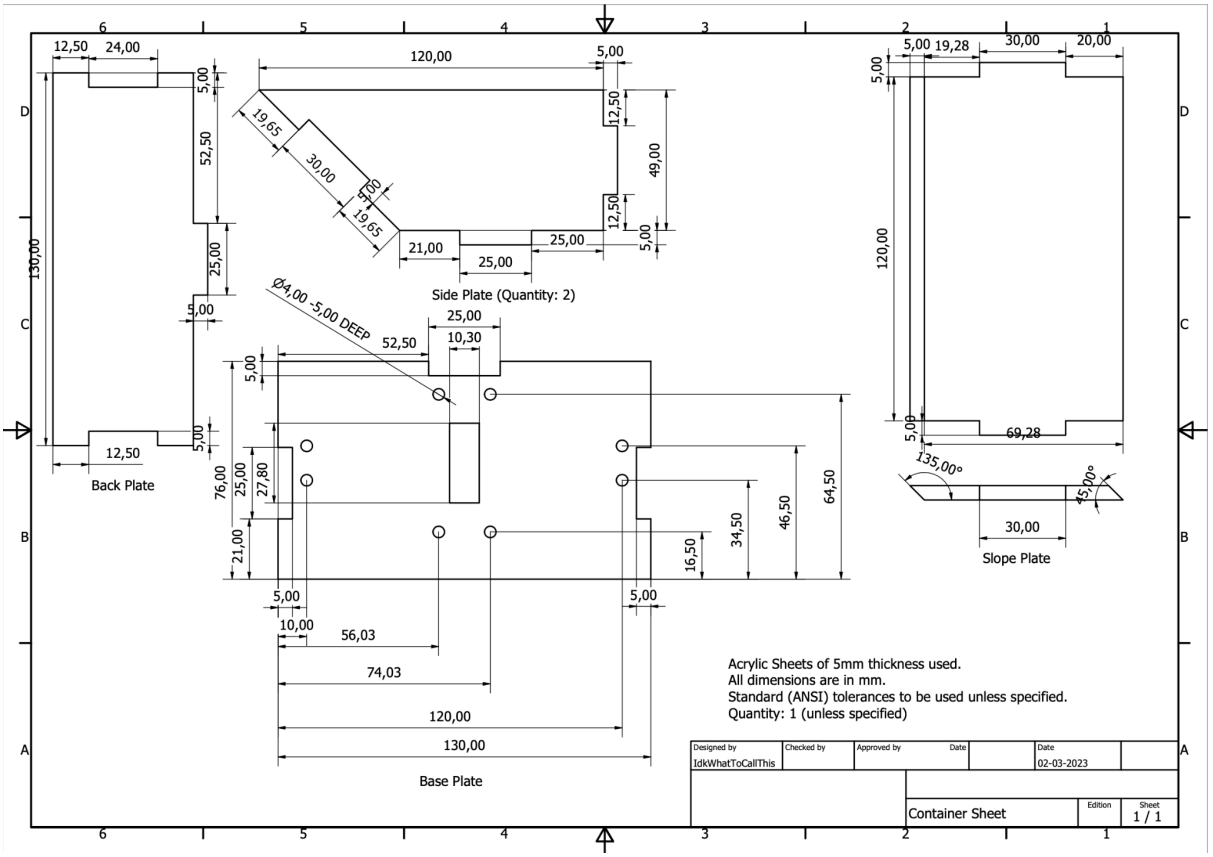


Fig. 3: Revised Container Design's Engineering Drawing

4.2 Other Changes

4.2.1 Limit Switch

We changed the proposed model (V-155-1A6) of the limit switch to the model D2F-01L. Since the model V-155-1A6 was not available in the Singapore market and that it would take too long to have it delivered from elsewhere, the team chose to use the D2F-01L model from the Electronics lab. The maximum operating force of the limit switch (1.47 N) is lower than the weight of the can (3.43 N), hence we decided to place a sponge on top of the limit switch to cushion the impact.

4.2.2 Servo

In order to calculate the torque needed by the servo, we had to account for the weight of the servo arm itself as well as the component of the can's force that was acting on the servo arm offering friction to the movement. Post analysis, we decided to use the sine component of the weight of the can in the calculations as it was the force acting perpendicular to the servo arm.

The required torque was increased to 0.1211 Nm. The MG90S servo has a stall torque of 2.2 Nm which at first glance seemed like sufficient power but as discussed in M1, a design target of 20%-50% of the stall torque is needed for delivery of maximum power, i.e. for optimal performance.

MG995 offered a higher stall torque which was a better design choice. Thus, in order to reduce risk, we decided to use MG995.

Refer to Annex D for servo torque calculations.

4.2.3 4x3 Keypad

We opted for a 4 x 3 Keypad instead of using 4 buttons for the dispenser interface system as it was available in the Electronics lab. This eliminates the hassle of soldering the 4 buttons as well as decreasing the complexity of programming needed to select a key. The user selects a table 1 to 6 by pressing on the corresponding number on the keypad. Asterisk (*) is programmed to undo any input given before the order is sent and Hashtag (#) to confirm the order and add it to the queue.

5. Testing

5.1 Methods of Testing

In this section, we will provide a detailed overview of the testing process that we conducted during the prototyping phase of our dispenser and delivery bot. We started with individual component testing, progressed to individual system testing and finally integrated systems testing. Furthermore, we calibrated our system to optimise the dispenser and delivery bot.

Our testing process was conducted in a sequential manner, following a specific order of tests as indicated by the arrows in our testing plan. This is because each subsequent test could only be performed after the necessary prerequisite tests were completed.

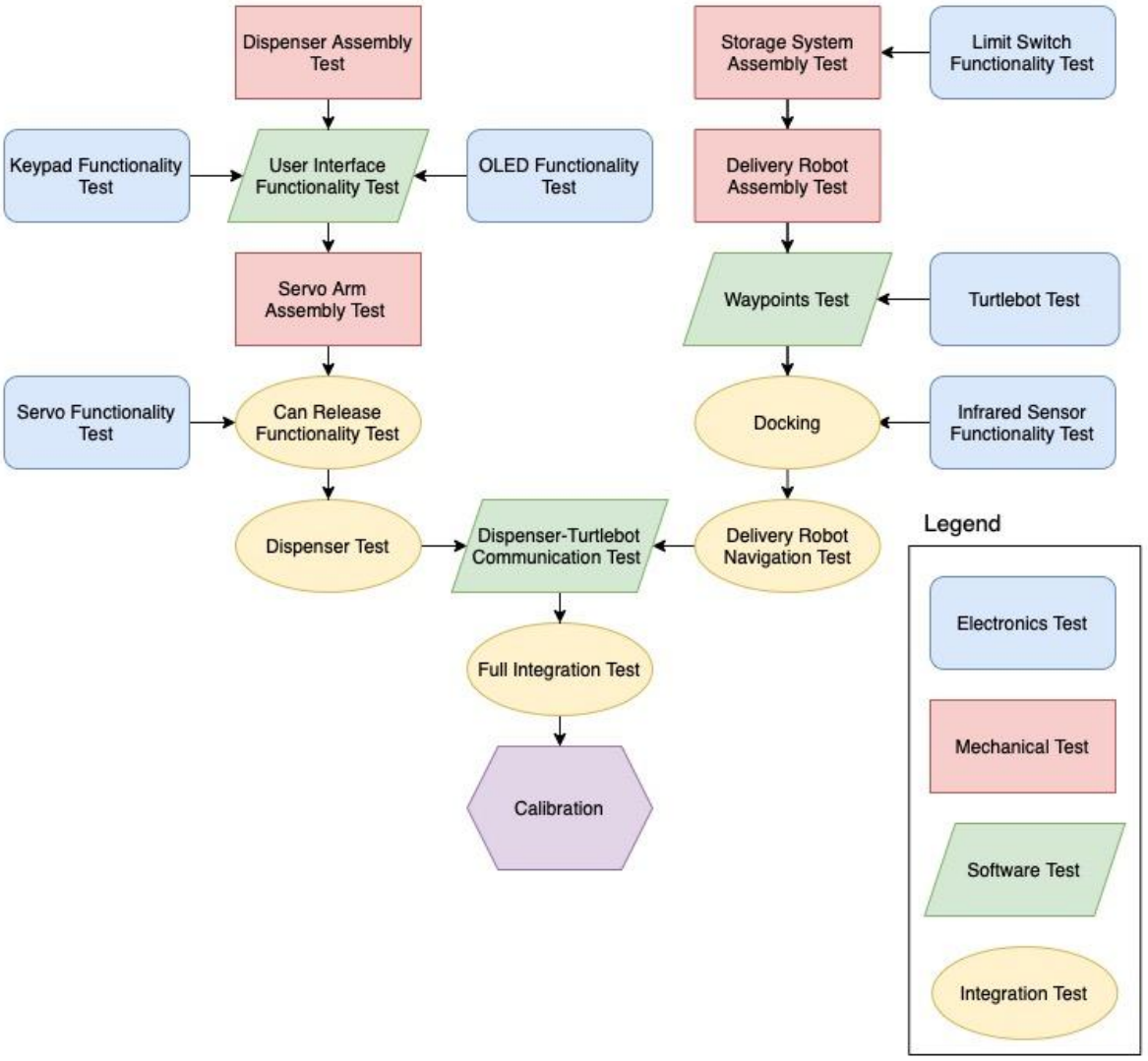


Fig. 4: Methods of Testing

5.2 Test Results

In this section, we will provide a comprehensive overview of the results from our testing plan and outline the necessary changes we made to address any issues or improve upon our prototype. We conducted multiple rounds of testing to ensure that all problems were rectified and the desired outcome was achieved.

Test Procedure	Test Results	Solutions
Software		
MQTT mosquitto broker is running on Raspberry Pi	Output of `systemctl status mosquitto` on terminal is "Active"	-
MQTT mosquitto broker is able to run concurrently with other processes, i.e rosbu	While we were able to run several processes concurrently, such as rosbu and rslam, we found that whilst Active, the MQTT broker severely lagged the RPi. As a result, unnecessary time was spent waiting for the RPi to respond to new commands.	We uninstalled the broker on the RPi and reinstalled it on the main computer
MQTT client (ESP32, RPi) is able to connect to the broker	We have found that the time duration allocated for the client to connect to the broker was far too little, causing the client to fruitlessly attempt to reconnect forever.	We have increased the time allocated for the ESP32 to connect to the broker from 10s to 30s
MQTT client is able to receive data from other clients	Callback functions were successfully triggered.	-
MQTT client is able to publish data to other clients	Correct callback functions were triggered on the client that is supposed to receive the message	-
WayPoints storage test	The coordinates and orientation can be seen to be updated in the json file (wayPointsData.json)	-

Navigation test	Turtlebot can navigate from the Docking point to the tables and from the tables back to the Docking point.	-
Docking test	Turtlebot can detect the line on the floor and start docking. It can correct itself to align with the line and thus with the Dispenser bot.	-
Electrical		
Keypad Functionality Test	Serial Monitor displays the key pressed.	-
OLED Functionality Test	OLED displays a series of different animations.	-
User Interface Functionality Test	OLED correctly reflects the user's action, correct table number, asterisk to cancel input before order is sent and hashtag to send order when the delivery bot is docked.	-
Servo Functionality Test	Servo turns to preset angle.	-
Can Release Functionality Test	Servo is able to turn to preset angle without stalling when a can is loaded.	-
Limit Switch Test	Limit Switch functions as intended, with ‘False’ echoed when Limit Switch is not pressed and ‘True’ echoed when the switch is pressed.	-
Infrared Sensor Test	Infrared Sensors work as intended, with ‘s’ echoed when there is a black line below both left and right IR sensors, and ‘f’ echoed when there is no black line below either sensor. ‘r’ or ‘l’ is echoed when there is a black line below the left or right IR sensor respectively.	-

Pi Camera Test	Pi Camera was not detected despite being connected to the camera port on the Rpi. Furthermore, we could not find the appropriate software file to configure the camera in Ubuntu.	After spending about two weeks attempting to configure the camera, we decided to abandon using the Pi camera and AprilTag for docking due to time constraints. We resorted to using our alternative docking method, Line Following, using IR sensors instead.
Turtlebot test	Turtlebot could be controlled by the computer and a LiDAR map is generated. However, the map crashes sometimes.	Restart rosbu and rslam if the map crashes
Mechanical		
Can is rolled down the sloped plate of the dispenser	Can rolls down the slope without the acrylic rails hindering or interfering its movement	-
Can is rolled down the slope while servo arm is in the neutral position	Servo arm is able to withstand the force of the can acting on it	-
Servo arm is manually pushed up and down	Servo arm is able to go up and down without getting obstructed by the flap that supports it	-
The delivery bot's container is placed in positions that are slightly deviated from the ideal docking position and a can is allowed to roll down the sloped plate of the dispenser to account for the random errors in the docking process	The can rolls into the container and the limit switch is triggered, echoing "True" value.	-

System Integration		
Dispenser Integration Test	The Dispenser is able to store a can. The user is able to key in table number on the keypad and the corresponding number is displayed on the OLED prior to confirmation. The servo is able to lift the servo arm when a can is loaded, allowing the can to roll down onto the container of the Delivery bot.	-
Delivery Integration Test	The Delivery bot is able to store a can whilst navigating accurately to designated tables and stop once reached within 15 cm of it. Upon which, it will remain there until the can is removed from the container. The Delivery bot is able to navigate back to the Dispenser. The Delivery bot is able to dock accurately at the Dispenser.	-
Full Integration Test	The Delivery bot is able to receive a can from the Dispenser and deliver it to the 6 tables.	-

6. Calibration

Once our Dispenser and Delivery bot achieved all their individual and their collective deliverables, we then started refining our implementation to handle corner cases and bugs to improve our rate of success. Furthermore, we also calibrated our implementation with the aim to complete the objective in the shortest time possible

No.	Issue	Cause	Solution(s)
1	A Brownout detector on the ESP32 was triggered sometimes when the servo rotates to raise the barrier	The servo was drawing too much power as it instantly changed from 0 degrees to 90 degrees	Added a small delay of 10 microseconds between intervals of one degree
2	The limit switch is not triggered even when the can is present about 60% of the time	The sponge that we added to the container to cushion the impact of the weight of the can on the limit switch was too effective and the orientation of the can within the container also affected whether it would trigger the switch	We added a secondary sponge that would restrict the final position of the can within the container to remain directly above the switch.
3	The Delivery bot does not always dock accurately with the dispenser	The length of the docking line as well as the thickness of the line was not sufficient enough to allow the Delivery bot to correct itself, before docking at the Dispenser bot.	We increased the thickness of the line to almost (around 2mm less than the exact distance) match the distance between the IR sensors on the Delivery bot. We also increased the length of the line.

4	The Delivery bot stops when approaching the docking line perpendicularly.	When the Delivery bot approaches the black line at a 90-degree angle, both IR sensors detect it. However, our current docking algorithm interprets this as a signal to stop, which should only occur when the bot is docked with the Dispenser bot.	We introduced a checking algorithm to make sure that the bot is indeed at the docking position (near the dispenser). When the bot detects the first 'stop' signal, it sets a count variable to 1. The bot then moves forward for a set amount of time (enough to overcome the thickness of the docking line) after which it checks the signal from the IR sensors again. If the signal is 'stop', it sets count to 2, which sets the docked flag to true, resulting in the Delivery bot stopping. Else, the docking function is called again which would result in the whole docking process to restart.
5	We were unable to complete the mission in time.	The linear speed while the Delivery bot moves straight as well as the angular speed while the bot rotates was too slow. Furthermore, the bot took too long to dock	We increased the linear speed (SPEED_CHANGE) as well as the rotation speed (ROTATE_CHANGE) which resulted in faster docking, rotation, and linear motion.
6	The Delivery bot does not always detect table 6 (unknown table)	The Delivery bot did not always stop at the same position from which it had to search for the table, this resulted in the angle to check the table to change, resulting in it not detecting the table occasionally.	We increased the angle it should check for the table from 44 degrees to 50 degrees. This allowed for error in position of the bot when it was searching for table 6.

7	The Delivery bot sometimes collides with table 6	We identified two main reasons for this: 1. The distance at which it should stop was too less, resulting in it colliding when the bot rotates. 2. The distance from the table was calculated from the LiDAR data's 0th index. However, while the bot moves towards the table, the table sometimes gets out of the bot's 0 degree view.	We increased the distance from the table (0.40m) at which the bot is supposed to stop in order to account for the bot's rotation. We also changed the algorithm to check for the distance at all angles, and to stop the bot when any one of the angles becomes less than the distance mentioned above.
8	The bot does not rotate accurately, especially after increasing the rotation speed in point 5.	The 'rotatebot' function does not account for momentum. Due to the increased rotation speed, the degree of error increased when turning large angles.	We introduced a while loop which keeps the bot turning until the angle relative to where it is supposed to be is less than the acceptable angle error.
9	Implementing point 8 resulted in the bot continuously turning left and right at some points when it is close to the goal point.	The speed of the rotation resulted in the bot turning more than the acceptable angle error everytime it tried to correct the error. This resulted in the bot continuously turning left and right.	The algorithm in point 8 was changed. In the while loop, we added an if statement which checked if the angle left to turn was less than a set angle. If this was true, then the rotation speed was adjusted to a slower speed in order to prevent overshooting.
10	Delivery Bot does not move straight to the waypoints.	When the bot moved long distances, it deviated from the direction it was supposed to move to. This results in the bot missing the waypoint.	We fixed this by adding a distance interval checking. When the interval is reached,It would check if the angle change is greater than the angle error. If this is true, the angle is corrected to make sure the bot is facing the waypoint it is supposed to go to.

11	Delivery Bot overshoots the waypoint occasionally.	The momentum of the bot causes it to move ahead of the waypoint. Once this happens, it just keeps moving ahead until the checking interval mentioned in point 10 is reached. However, by this time the bot would have crashed with the wall or the bucket.	We fixed this by first determining a speed reduction distance. Once the distance to the waypoint is less than this distance, the linear speed is reduced such that the bot doesn't overshoot the waypoint. We also increased the acceptable stopping distance from the waypoint to account for minor errors. Additionally, we added an if statement which checked if the distance to the waypoint is less than the speed reduction distance mentioned above. If it is, then the distance at which it checks for the angle mentioned in point 10 is reduced in order to prevent the bot from overshooting the waypoint if it is not facing the waypoint directly.

7. Final Design

7.1 CAD model

7.1.1 Dispenser

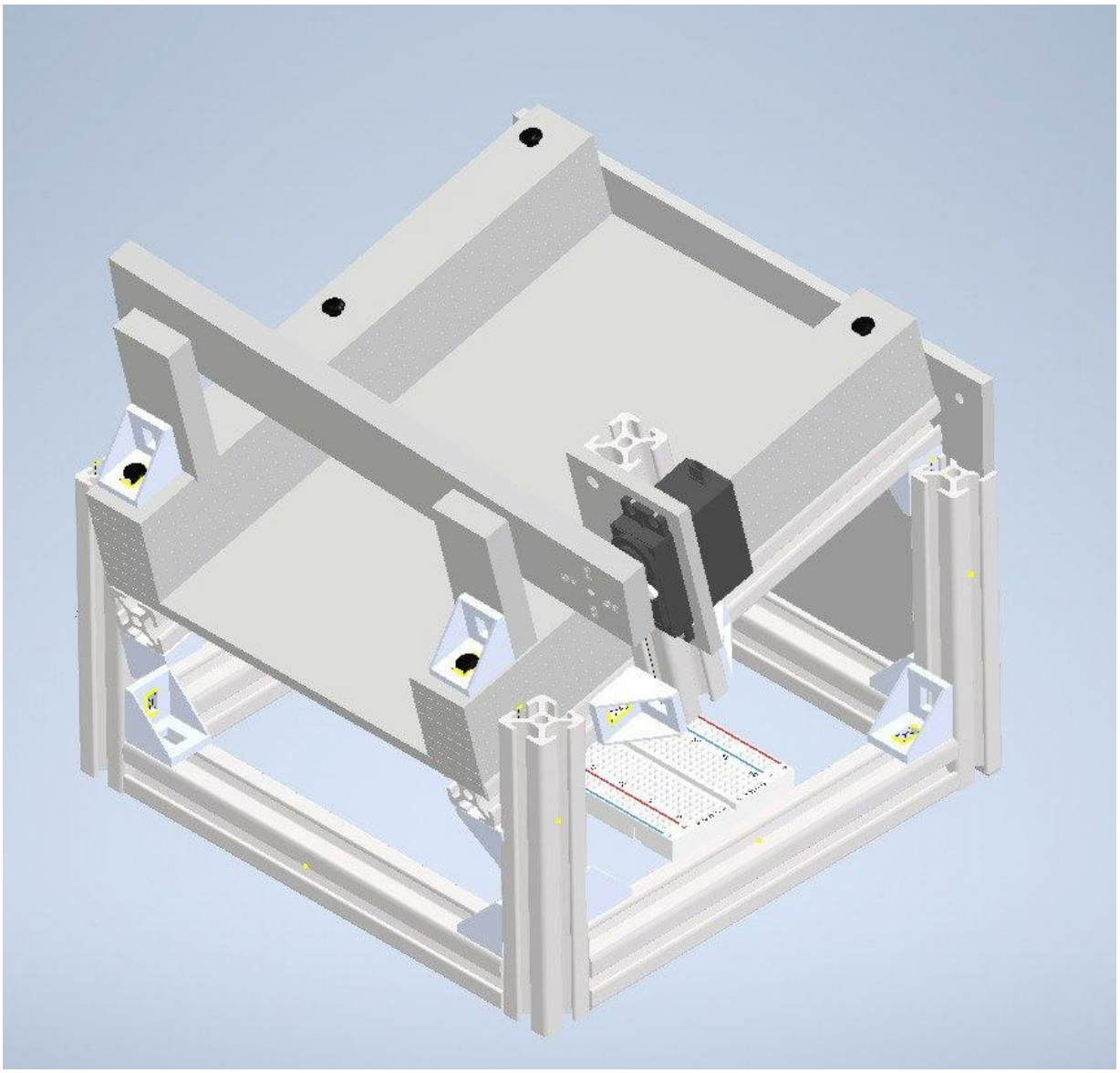


Fig. 5: Isometric View of the Dispenser

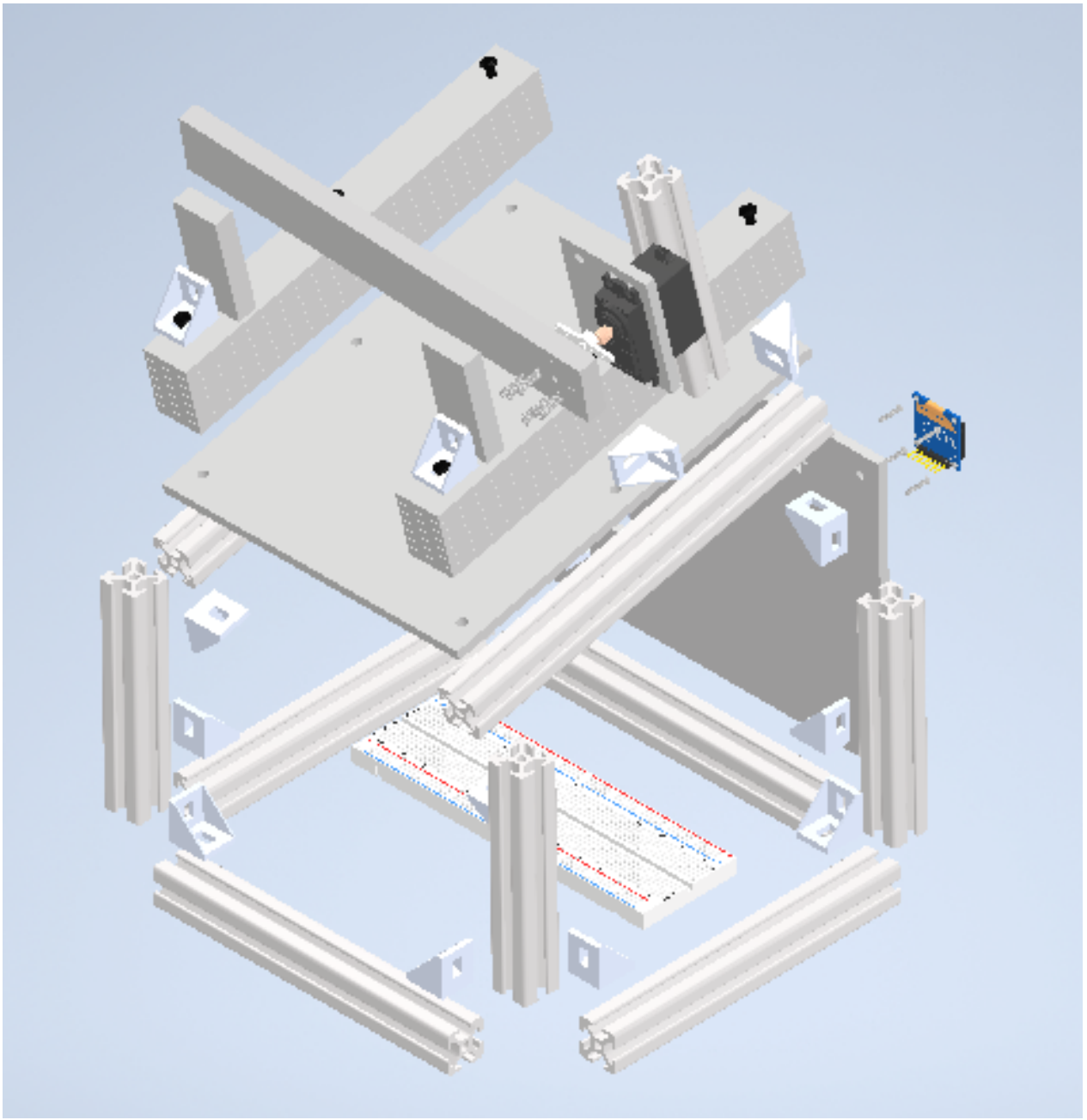


Fig. 6: Exploded View of Dispenser

7.2.2 Delivery Bot

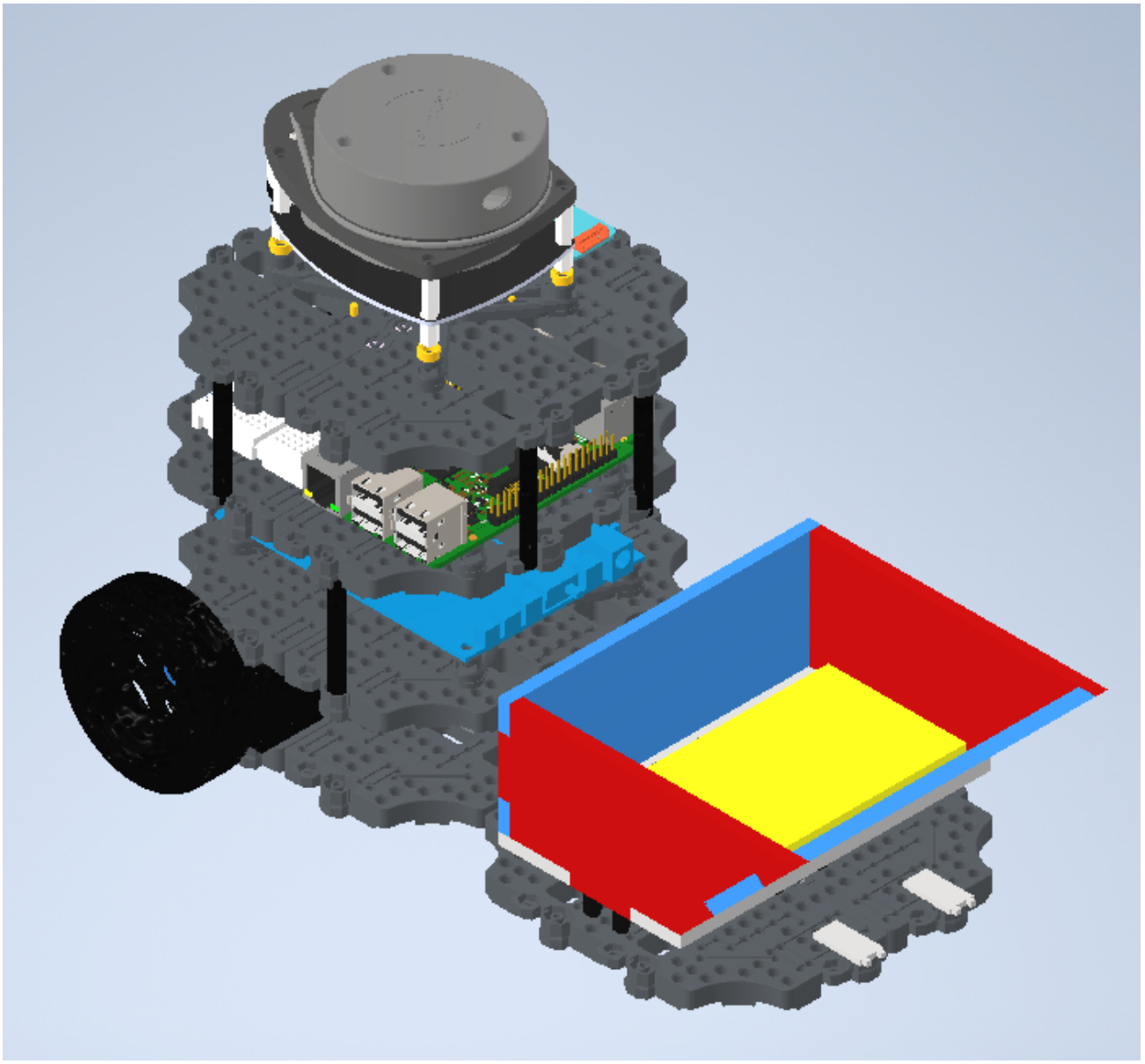


Fig. 7: Isometric View of the Delivery Bot

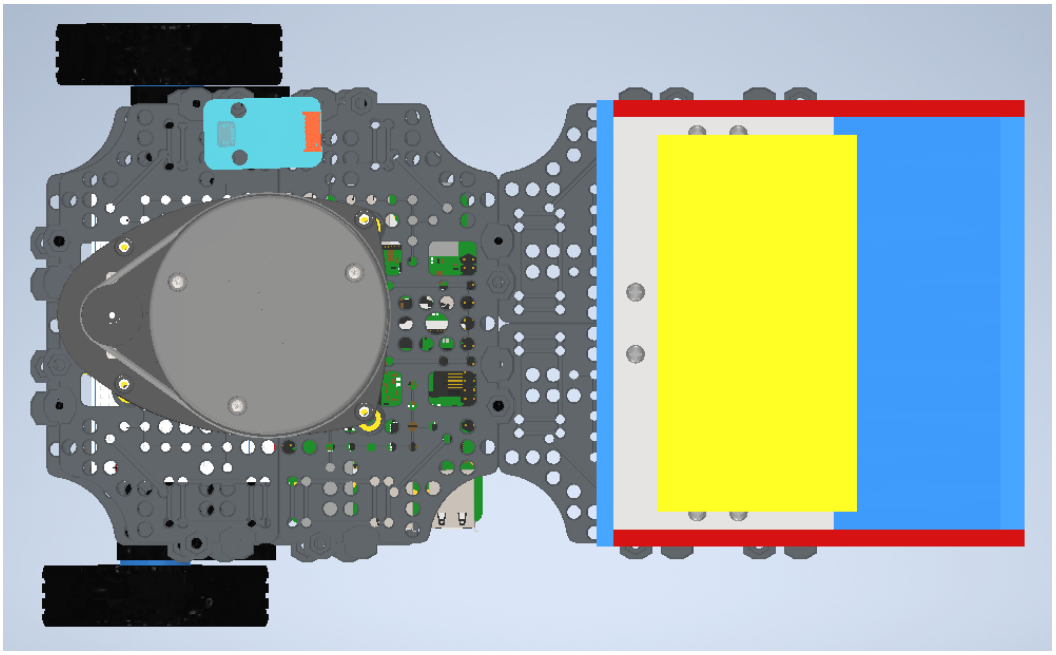


Fig. 8: Top View of Delivery Bot

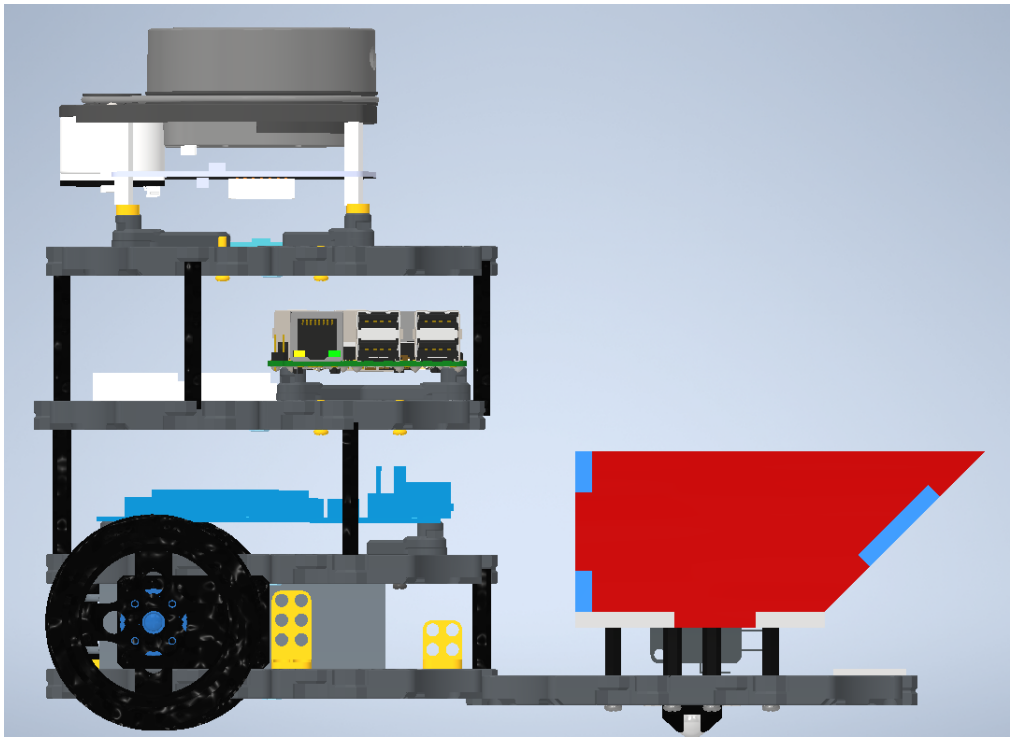


Fig. 9: Side View of Delivery Bot

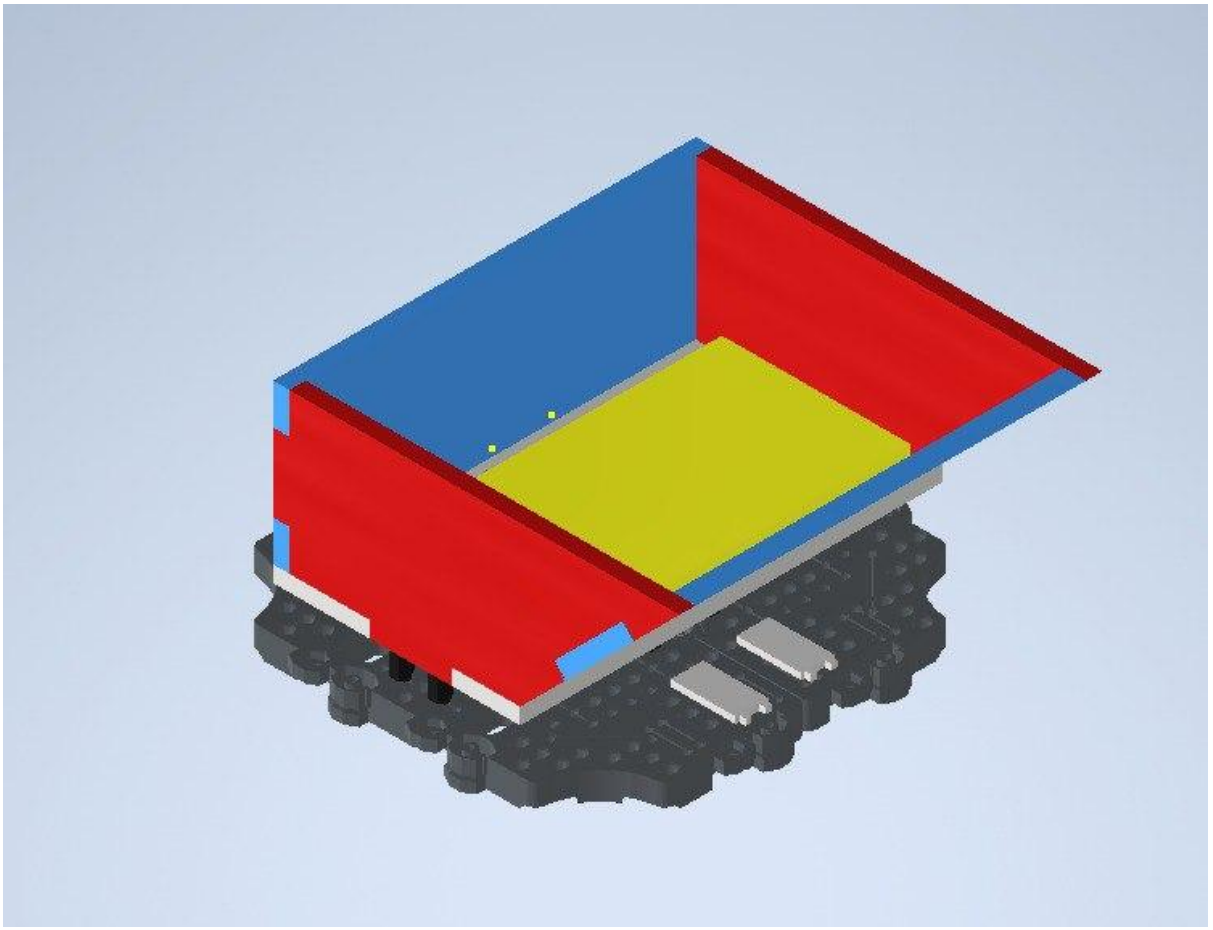


Fig. 10: Isometric View of the Container

7.2 Mechanical Overview

The mechanical systems consist of the Can Release system and the Delivery bot Storage system.

7.2.1 Can Release System

When the delivery bot is in the docked position and the user has entered a valid input, the Can Release system must initiate the lift of the servo arm to allow the can to roll into the Delivery bot Storage system. Additionally, while the delivery bot is completing a task, the Can Release system should be able to support the weight of the can during the wait.

Our Can Release system comprises several components of the Dispenser. The main components include the MG995 servo, the 221mm long acrylic servo arm, acrylic flaps that support the servo arm, and the acrylic sloped plate of the dispenser along with 35mm long stacked railings paving the path for the can. All the acrylic sheets are of 5mm thickness. Acrylic was the material opted for due to its ease of fabrication and cost-effective nature.

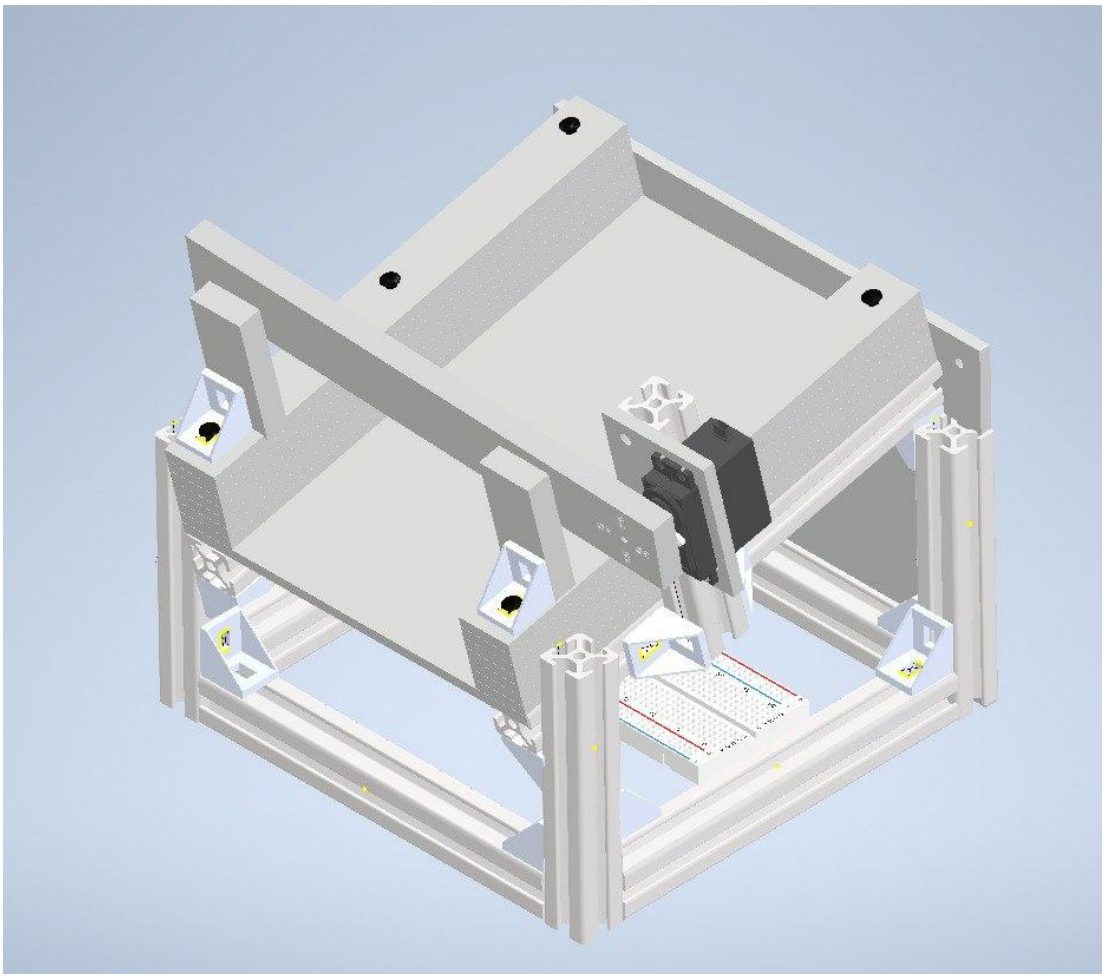


Fig. 11: Exploded View of the Delivery Bot Storage System

Implementation:

The Dispenser has a path of width 117mm for the can to roll down since the can itself is of length 115mm, with a 1mm clearance on both sides. The slope of the dispenser is 12º with respect to the ground, which is sufficient for the can to roll down, taking into account the coefficient of friction of acrylic. By placing the can on a slope, only a fraction (sine component) of the weight of the can acts on the servo arm. The flaps that support the servo arm offer an increase in surface area, therefore reducing pressure off the servo arm.

7.1.2 Delivery Bot Storage System

The can to be transported is placed in the Delivery bot Storage system (container) located on the Delivery bot. An extra waffle plate is added to the Delivery bot for it. The orientation is illustrated in Figure 7. By placing two castor wheels under this waffle plate, the Delivery bot does not have to hold the entire weight of the can. The limit switch on the can is used in conjunction with a sponge.

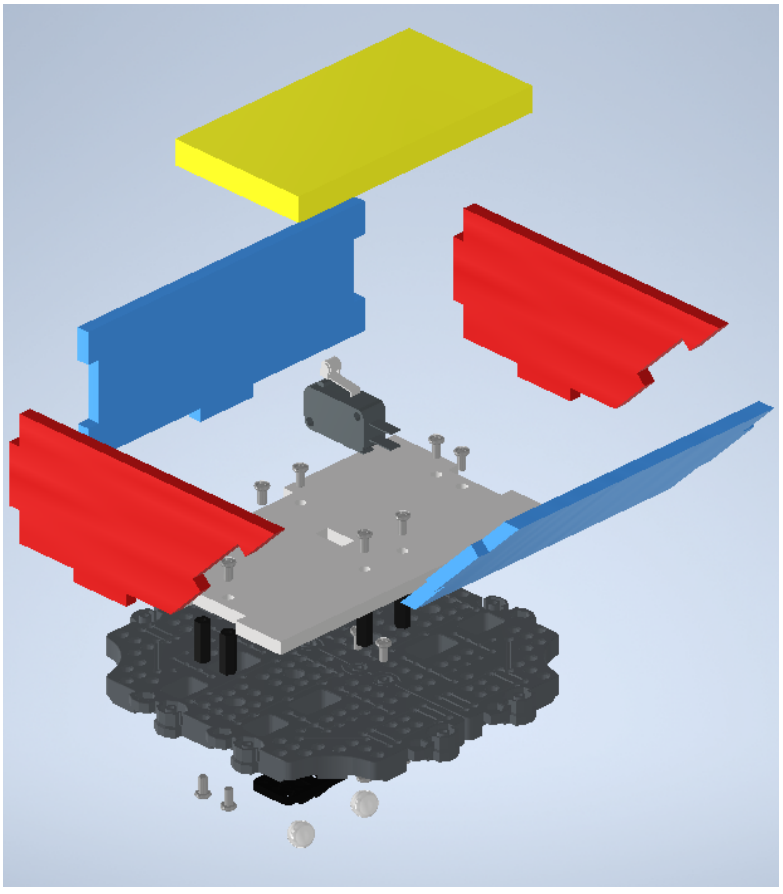


Fig. 12: Exploded View of the Delivery Bot Storage System

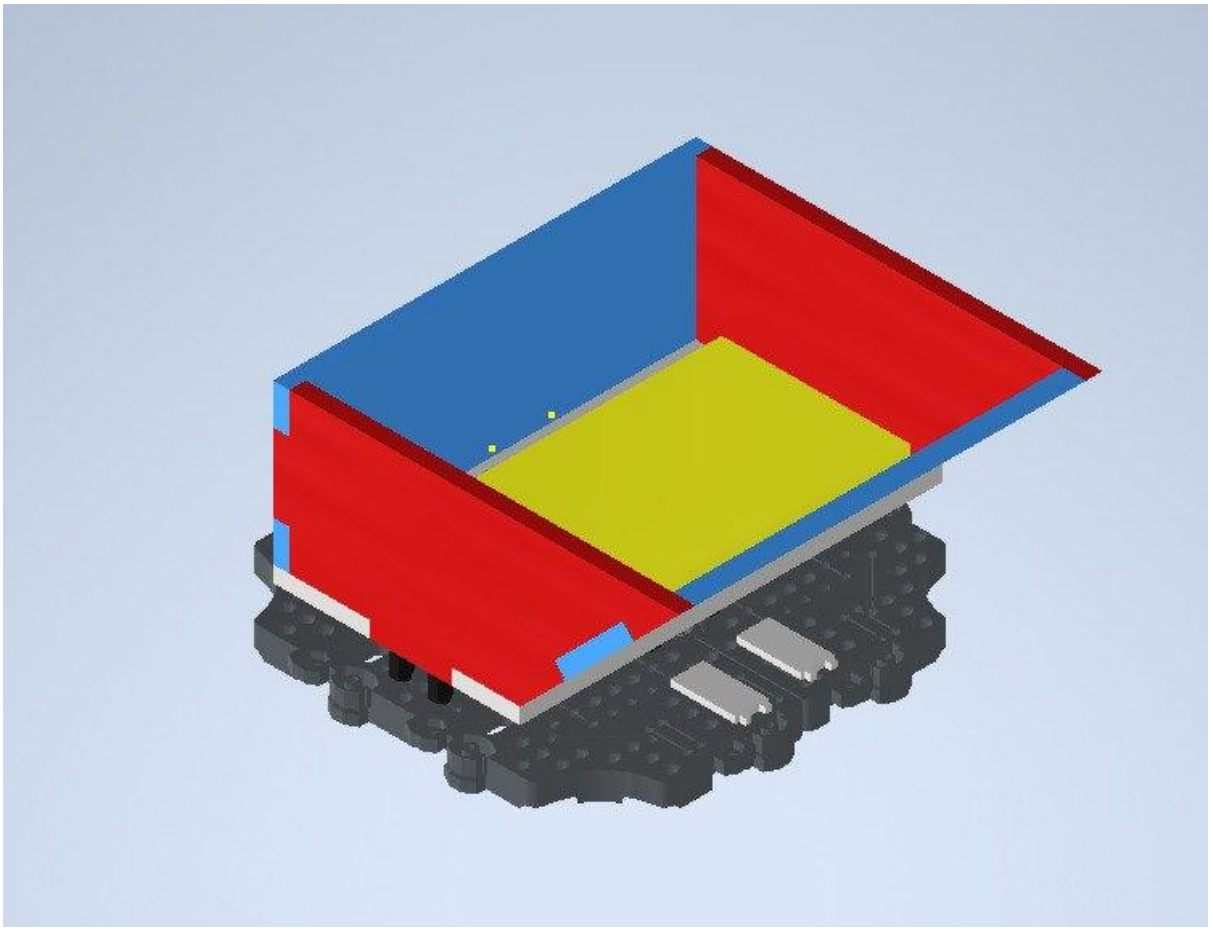


Fig. 13: Isometric view of the Delivery Bot Storage System

Implementation:

The maximum operating force of the limit switch (1.47N) is lesser than the weight exerted by the can on it (3.43N). In order for the proposed limit switch to be used, a sponge is placed on top of it as a means of cushioning. To match the elevation of the sloped plate of the dispenser which facilitates a smooth roll of the can into the container, the container is placed on standoffs of 8mm.

7.3 Electrical Overview

7.3.1 Functional Block Diagram

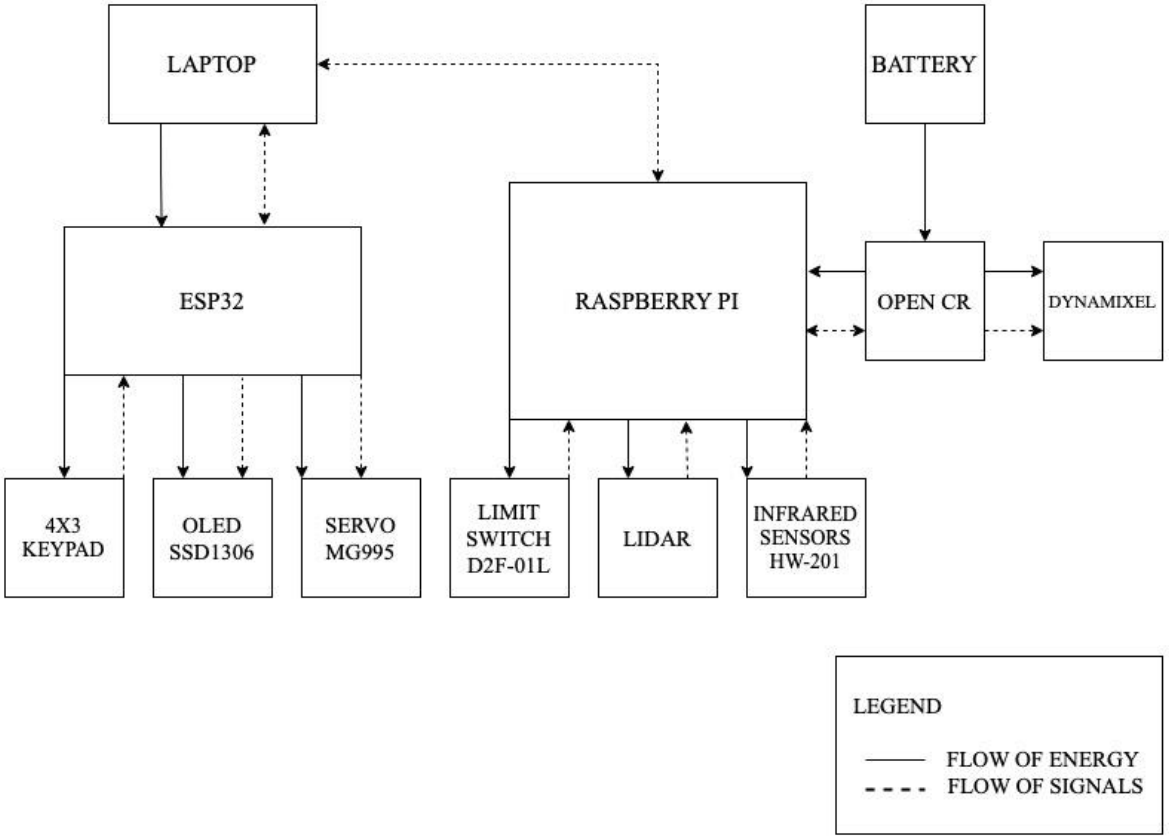


Fig. 14: Functional Block Diagram of Bot

7.3.2 Dispenser

The OLED displays a half circle on the screen implying waiting for connection to the server. A successfully connected message will be displayed when a successful connection is established. After the can is loaded, the user can then proceed to select a table from 1 to 6 via pressing the corresponding key on the keypad. Asterisk (*) to undo any input given before the order is sent and Hashtag (#) to confirm the order and add it to the queue. The servo arm will turn to release the can if the delivery bot is docked.

7.3.3 Delivery Bot

The Infrared sensors detect the black line and the Delivery bot moves forward and dock. The can then rolls down the slope into the container of the Delivery bot when the flap is open. The weight of the can triggers the limit switch, the Delivery bot will then navigate to the designated table. After the can is removed by the container, the limit switch will return a “false” value, the Delivery bot will then return to dock.

7.3.4 Electronic System Architecture

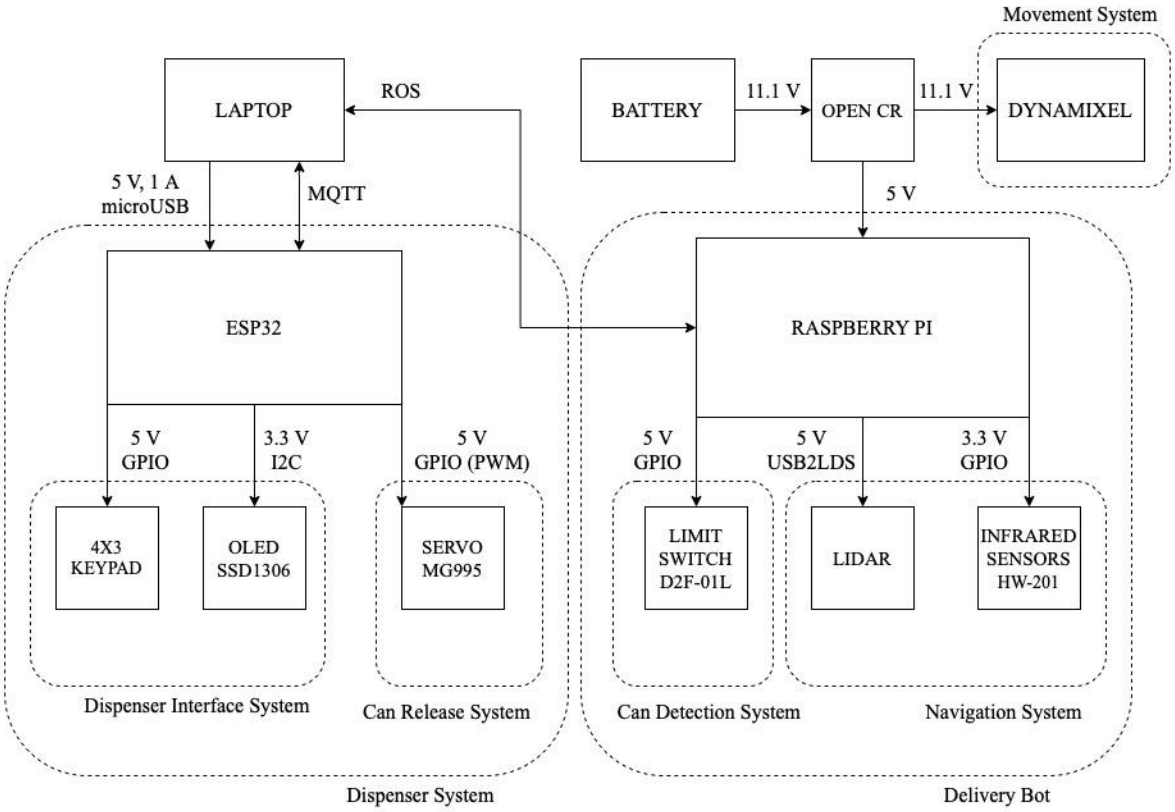


Fig. 15: Electronics Systems Architecture

7.3.4 Power Budgeting

Delivery Bot:

Component	Voltage/V	Current/A	Power Consumption/W
Delivery Bot (During operation)	12.3 V	0.70 A	8.61 W
Limit Switch D2F-01L	5 V	0.16 A	0.8 W
2 x Infrared Sensor HW-201	5 V	0.032 A	0.32 W

Battery Capacity = 19.98 Wh

Total Power Consumption = 8.61 + 0.8 + 0.032 = 9.73 W

Assuming Battery efficiency of 75%,

Battery Life = 0.75 x 19.98 / 9.73 = 1.54 h

The battery life of 1.54h is sufficient for our mission which lasts 26 minutes maximum.

Dispenser:

Component	Voltage/V	Current/A	Power Consumption/W
ESP32	3.3 V	0.24 A	0.792 W
4x3 Keypad	5 V	0.10 A	0.50 W
OLED SSD1306	3.3 V	0.020 A	0.066 W
Servo Motor, MG995 (During operation)	5 V	0.20 A	1.00 W

Maximum Power Consumption = 0.792 + 0.50 + 0.066 + 1.00 = 2.36 W

Choice of Power Supply: microUSB connected into a laptop

Power supplied = 5 V x 1 A = 5 W > 2.36 W

According to our calculations, we will need 2.36W of power to operate our Dispenser bot. As the bot is always stationary, we determined that a microUSB connected to a laptop would be sufficient as the power source. This connection can supply up to 5W of power, which meets and exceeds the power requirements of our bot. Furthermore, this provides us with an additional failsafe in case that the keypad or OLED fails.

7.4 Software Overview

7.4.1 Software Block Diagram

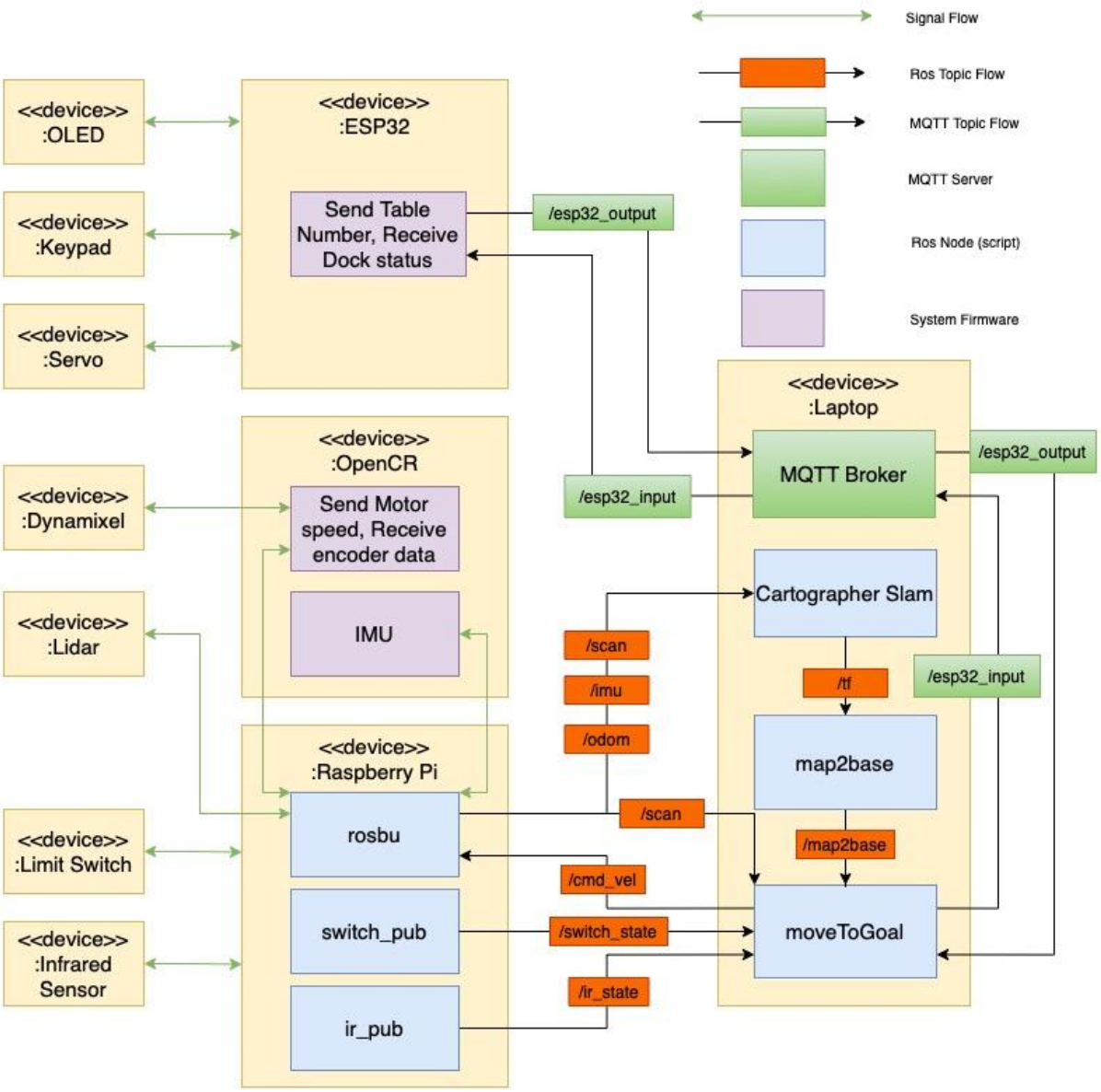


Fig. 16: Software Block Diagram

The diagram is a brief overview of the transfer of signals and/or data between devices through specific mediums (MQTT or ROS).

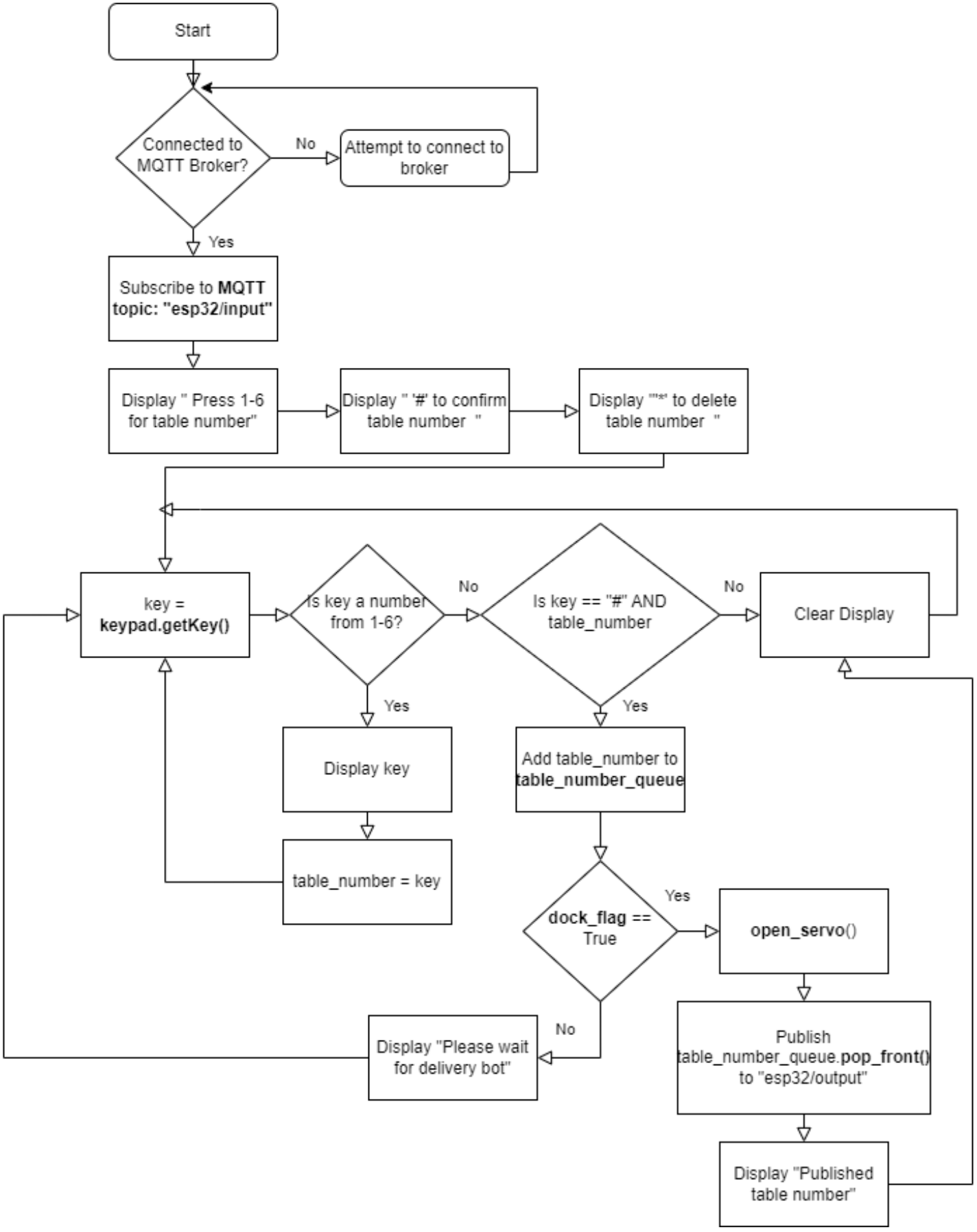
7.4.2 Dispenser

The main algorithm used within the Dispenser is to handle three deliverables:

- User Interface
- Can Release System
- Publishing table number via MQTT

The following diagram and table illustrates the logic of the algorithm

Variable / Function	Description	Initial Value
MQTT Topic: “esp32/input”	The Dispenser will subscribe to this topic on the MQTT broker to receive the dock status of the Delivery bot	-
keypad.getKey()	Returns the value of the key pressed on the keypad	None
table_number_queue	A queue that stores the confirmed table numbers	None
dock_flag	Boolean value to check whether the Delivery bot is docked. Its value will be updated by a callback function that subscribes to “esp32/input”. If a message of “1” is published by the Delivery bot, set dock_flag to True. If a message of “0” published by the Delivery bot, set dock_flag to False	True
open_servo()	Function call to raise barrier of Can Release System by rotating the barrier by 90°, wait 3 seconds and then lower the barrier	-
pop_front()	Returns the last value added to the queue	-


Start
Connected to MQTT Broker?
No
Attempt to connect to broker
Yes
Subscribe to MQTT topic: "esp32/input"
Display " Press 1-6 for table number"
Display " '#' to confirm table number "
Display ""*" to delete table number "
key = keypad.getKey()
Is key a number from 1-6?
No
Is key == "#" AND table_number
No
Clear Display
Yes
Display key
table_number = key
Yes
Add table_number to table_number_queue
dock_flag == True
Yes
open_servo()
No
Display "Please wait for delivery bot"
Publish table_number_queue.pop_front() to "esp32/output"
Display "Published table number"

7.4.3 Delivery Bot

For our main navigation algorithm, we decided on setting waypoints on the map that the Delivery bot would use to get to the tables. We chose this method as it works well for our mission statement. An alternative to this was to create our own path planning-algorithm which would require much greater knowledge which we did not possess. This would have also taken a lot more development time. For these reasons, we decided to go ahead with the waypoints algorithm.

The diagram below shows the general overall process for our navigation algorithm (moveToGoal.py). The individual functions as well as some important variables are described in the following tables.

Table of description of important variables

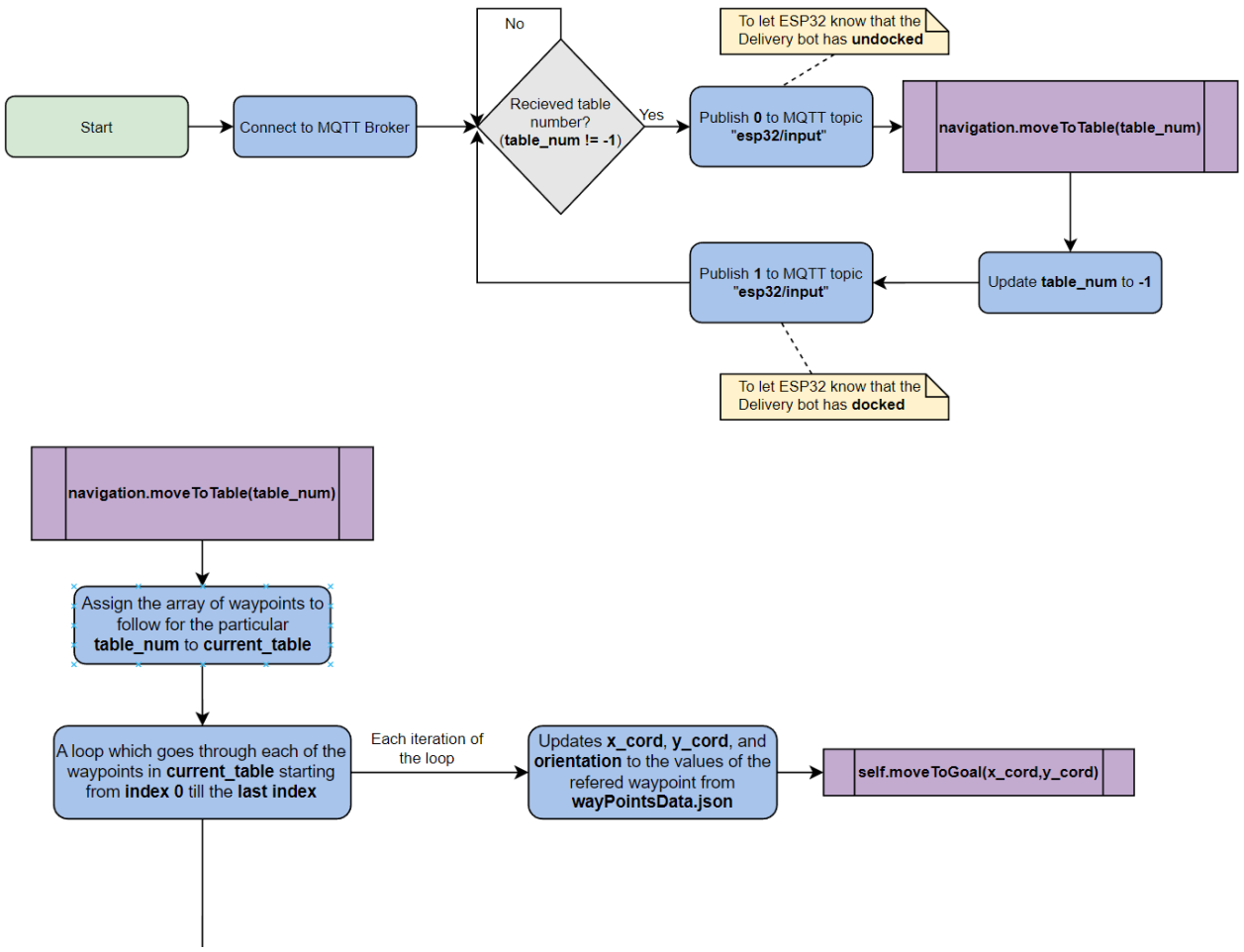
Variable	Description	Initial Value
table_num	To store the table number entered by the user and sent to the RPi by the ESP32 through the MQTT topic “esp32/output	-1
current_table	Stores the array of waypoint numbers for going to table_num table	-
x_cord	Stores the x-coordinate of the waypoint	-
y_cord	Stores the y-coordinate of the waypoint	-
orientation	Stores the orientation of the Delivery bot, when the waypoint was saved	-
angle_to_turn	Stores the angle to turn in degrees. Calculated by subtracting the current orientation (in degrees) from the angle to the waypoint (goal), in degrees	-
self.switch_state	Stores the state of the limit switch. switch_state: True = pressed, False = not pressed	False

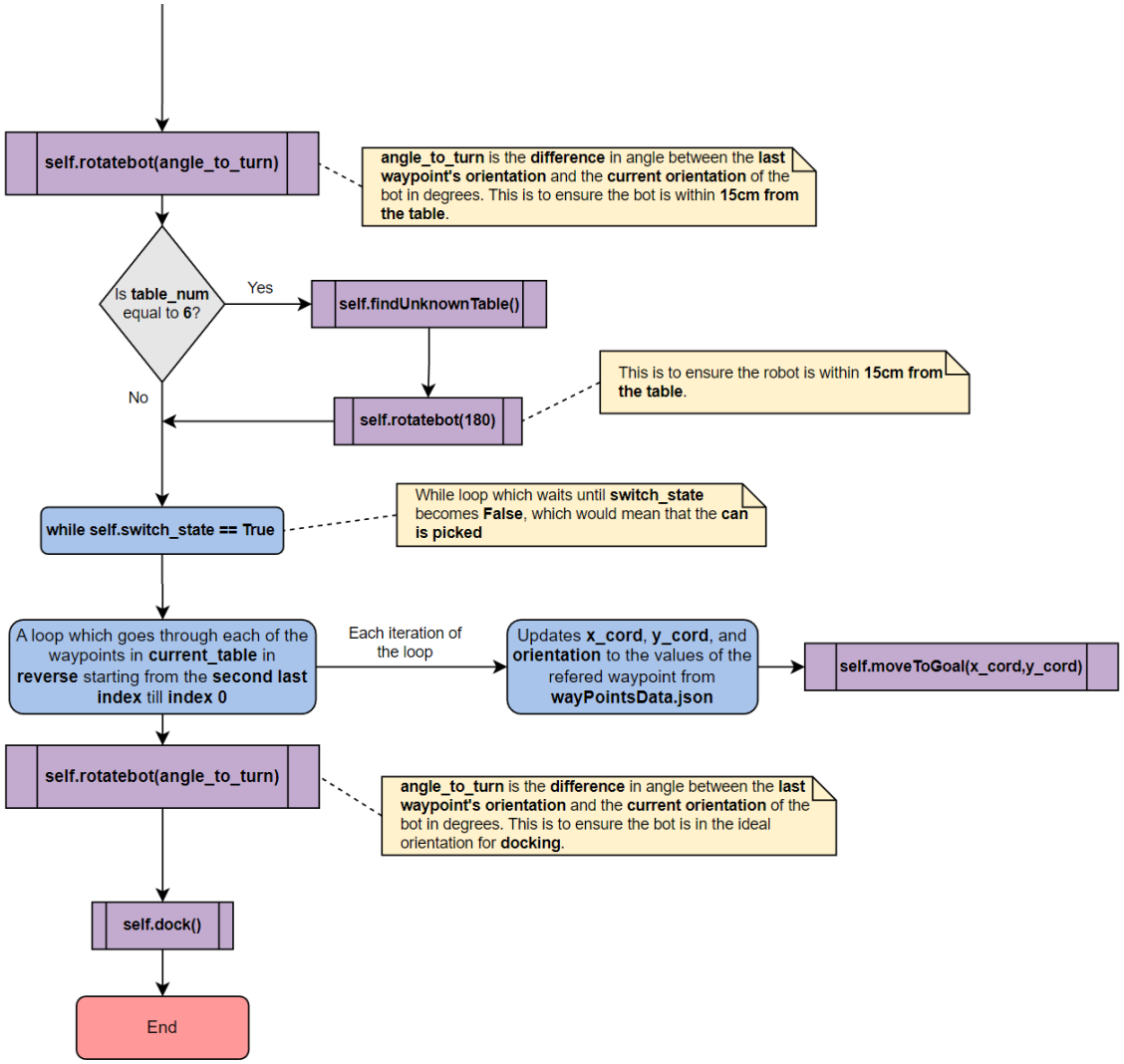
Table of description of various functions:

Function	Description
navigation.moveToTable()	This function takes a table number as input and uses pre-defined waypoints to navigate the bot to the table. It moves the bot to each point in the waypoint array of the specific table number. After reaching the last waypoint, it rotates the bot to match the saved orientation of the last waypoint.

	If the table number is 6, it finds the unknown table. Once it is at the table, it rotates the bot by 180 degrees to ensure that it is within 15 cm from the table. After reaching the table, it waits for the can to be picked, and once the can is picked, it moves the bot back to the docking point in reverse order of the waypoint array excluding the last waypoint in the array (as it is already at that waypoint). Once the bot is at the docking point, the function to dock the bot is called.
self.moveToGoal()	This function takes two parameters, the x-coordinate and the y-coordinate of the waypoint (goal) to which the bot is supposed to move towards. It creates a 'Point' object to store the goal position and a 'Twist' object to store the linear speed and angular speed values for the bot. It uses a while loop to rotate the bot towards the goal position and then moves the bot straight towards the goal position until it reaches the goal position. During the bot's motion, it checks the distance and angle between the current position and the goal position and adjusts the bot's motion accordingly. At the end of the function, it publishes the Twist message to control the bot's motion. There are several if statements to handle different scenarios, such as if the distance to the goal position is less than the 'SPEED_REDUCTION_DISTANCE', the bot's linear speed is reduced to prevent overshooting. If the bot deviates from its path, the angle to the goal position is checked and corrected. Overall, this function controls the bot's motion towards the waypoint.
self.rotatebot()	This function takes in one parameter, the angle to turn in degrees. The function then uses Twist messages to rotate the bot based on the parameter's value. This function was taken from the r2moverotate code.
self.findUnknownTable()	This function uses laser scan data to locate the unknown table and move the bot towards it. How it works is, in the scan_callback function, it checks for the index in the laser scan data array which has the lowest value within a range of indexes. It then calculates the angle to turn such that the bot faces this said index. This value is then passed onto

	the findUnknownTable function. The bot then turns this angle using the self.rotatebot function and then moves straight until it reaches the table. How it detects that it has reached the table is that the function uses a while loop which continuously updates the laser scan data and checks if any of the angles (indexes) has a value less than 0.40. If it does, it stops the bot by sending a Twist message.
self.dock()	This function uses the infrared sensors data to dock the bot with the Dispenser bot. The bot starts by rotating in place until it finds the docking line using its infrared sensors. It then follows the line until it reaches the end. The function uses a while loop to continuously check the bot's infrared sensor readings and moves the bot accordingly by publishing Twist messages. Once the bot reaches the pattern for stopping, which is a long and wide horizontal strip (refer to Annex E), the function publishes Twist messages to stop the Delivery bot.




self.rotatebot(angle_to_turn)
angle_to_turn is the difference in angle between the last waypoint's orientation and the current orientation of the bot in degrees. This is to ensure the bot is within 15cm from the table.
Is table_num equal to 6?
Yes
self.findUnknownTable()
No
self.rotatebot(180)
This is to ensure the robot is within 15cm from the table.
while self.switch_state == True
While loop which waits until switch_state becomes False, which would mean that the can is picked
A loop which goes through each of the waypoints in current_table in reverse starting from the second last index till index 0
Each iteration of the loop
Updates x_cord, y_cord, and orientation to the values of the refered waypoint from wayPointsData.json
self.moveToGoal(x_cord,y_cord)
self.rotatebot(angle_to_turn)
angle_to_turn is the difference in angle between the last waypoint's orientation and the current orientation of the bot in degrees. This is to ensure the bot is in the ideal orientation for docking.
self.dock()
End

8. Specifications

The Delivery bot uses the ROBOTIS TurtleBot 3 as the basis of construction. Refer to TurtleBot 3 Burger Specifications found on ROBOTIS e-Manual website [1]. The following listed item specifications add on to, or overwrite the respective item specifications listed on the ROBOTIS e-Manual website:

General Specifications(Delivery Bot)	
Item	**Specification**
Maximum Payload	1 Standard 330 ml can
Size in cm (LxWxH)	28.6 x 17.8 x 19.3
Software Version	Running on Ubuntu 20.04 and ROS2 Foxy with Raspberry Pi 3B+
Battery Type	LiPo Battery
Battery Capacity	19.98Wh
Expected operating time	1.54h
Weight	1063.1g
Center of Gravity	(4.4cm, 9.3cm, 6.4cm)
Infrared Sensor	HW-201
Limit switch	D2F-01L
General Specifications (Dispenser Bot)	
Item	**Specifications**
Maximum Payload	1 Standard 330 ml can
Size in cm (LxWxH)	28.6 x 17.8 x 19.3
Power Supply	5 V, 1 A microUSB connected to a laptop
Keypad	4x3 Keypad
OLED	SSD1306
Servo	MG995

9. Bill of Materials

The delivery bot uses the ROBOTIS turtlebot 3 as the basis of construction. Refer to TurtleBot 2 parts list found on ROBOTIS e-Manual Website [6]. The exhaustive bill of materials and developmental cost and cost per bot can be found in **Annex B**.

The following parts listed are removed from the base turtlebot parts list in the construction of our delivery bot.

Part Name	Quantity
Plate support M3 x 35mm	1

Note: The location of the ball caster from the original TurtleBot has been changed. Refer to the mechanical assembly for detailed breakdown.

10. Assembly Guide

Assemble the TurtleBot 3 Burger with reference to the TurtleBot 3 e-manual available on the Botis website. The changes made to the assembly have been described below

10.1 Mechanical Assembly

10.1.1 Layer 1:

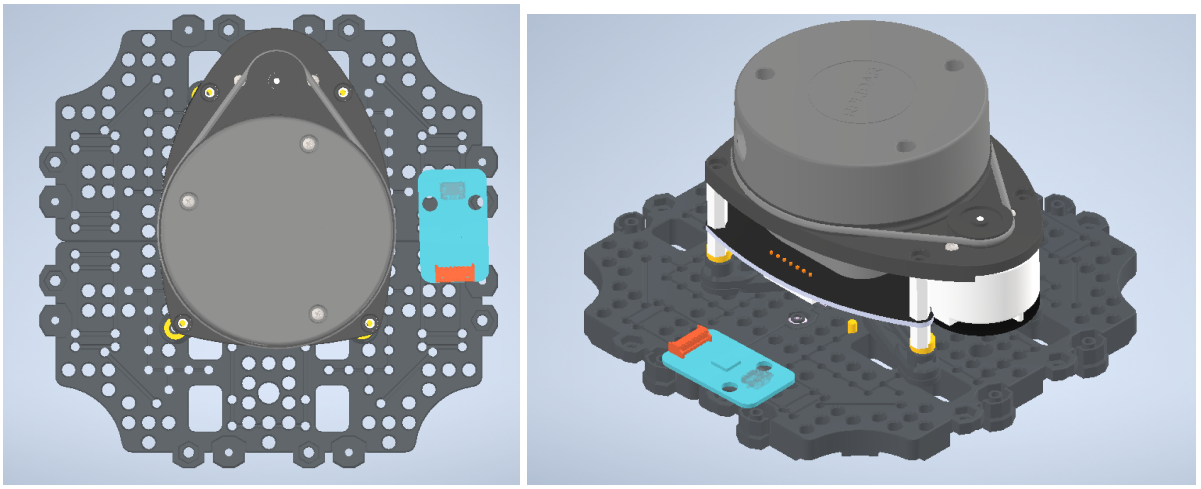


Fig. 17: Isometric View of Layer 1 (left), Top View of Layer 1 (right)

The USB2LDS's position is changed to the top plate and attached via the adapter plate as illustrated in figure 16.

10.1.2 Layer 2:

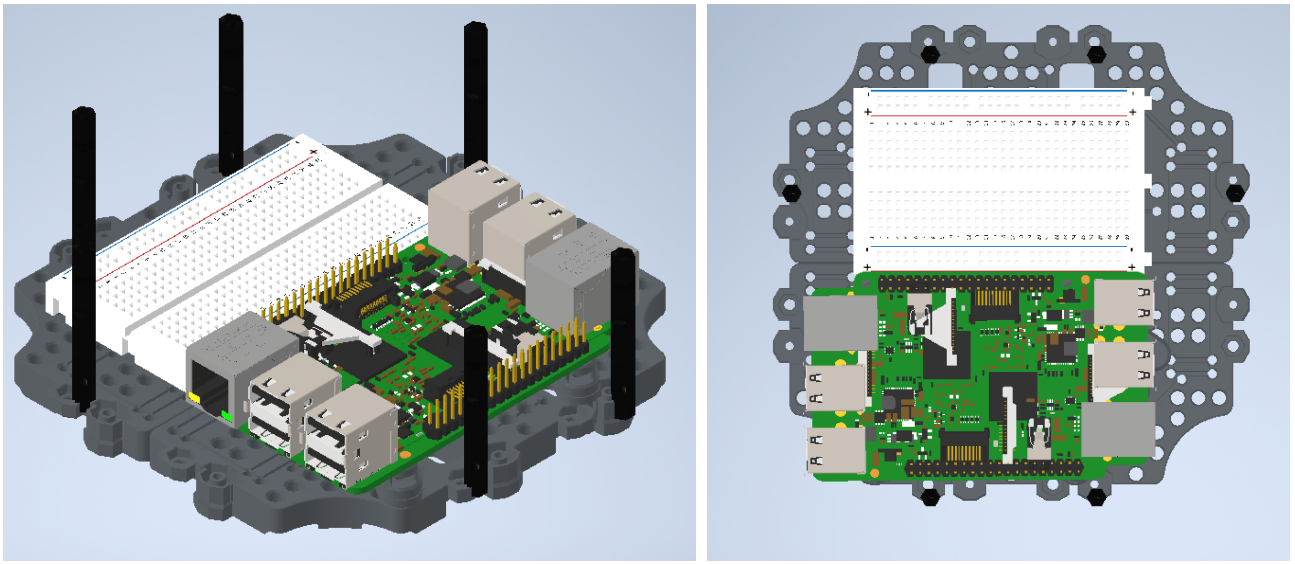


Fig. 18: Isometric View of Layer 2 (left), Top View of Layer 2 (right)

In order to do the wiring, a breadboard is added to this layer. As mentioned previously, there is also a change in the USB2LDS's position.

10.1.3 Layer 4:

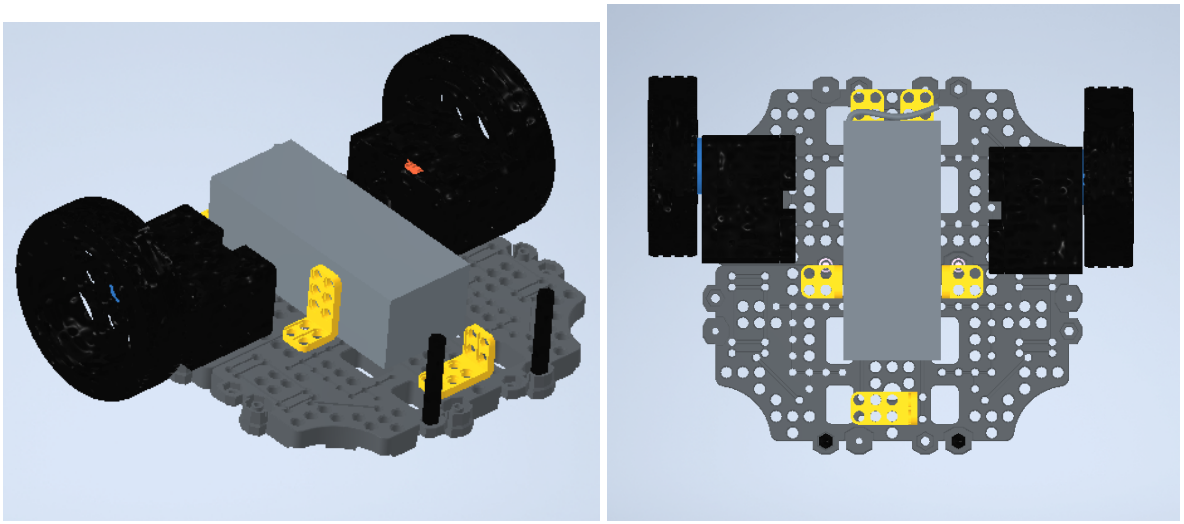


Fig. 19: Isometric View of Layer 4 (left), Top View of Layer 4 (right)

The castor ball is removed from the bottom. In order to accommodate the extra waffle plate added for the delivery bot storage system, 20 mm long standoffs are used in conjunction with 12 mm long standoffs to maintain height.

10.1.4 Container Assembly

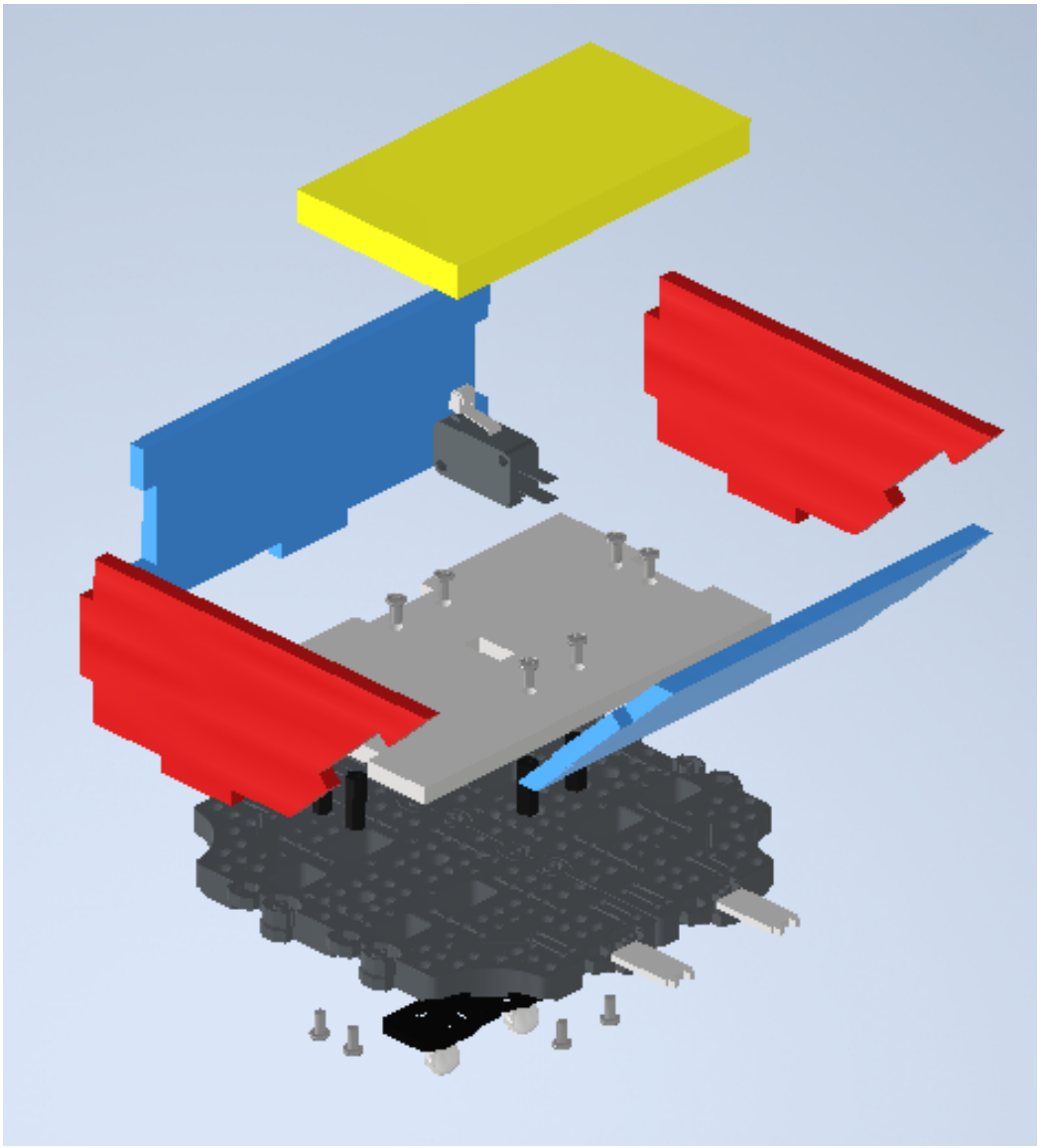


Fig. 20: Exploded View of the Delivery Bot Storage System

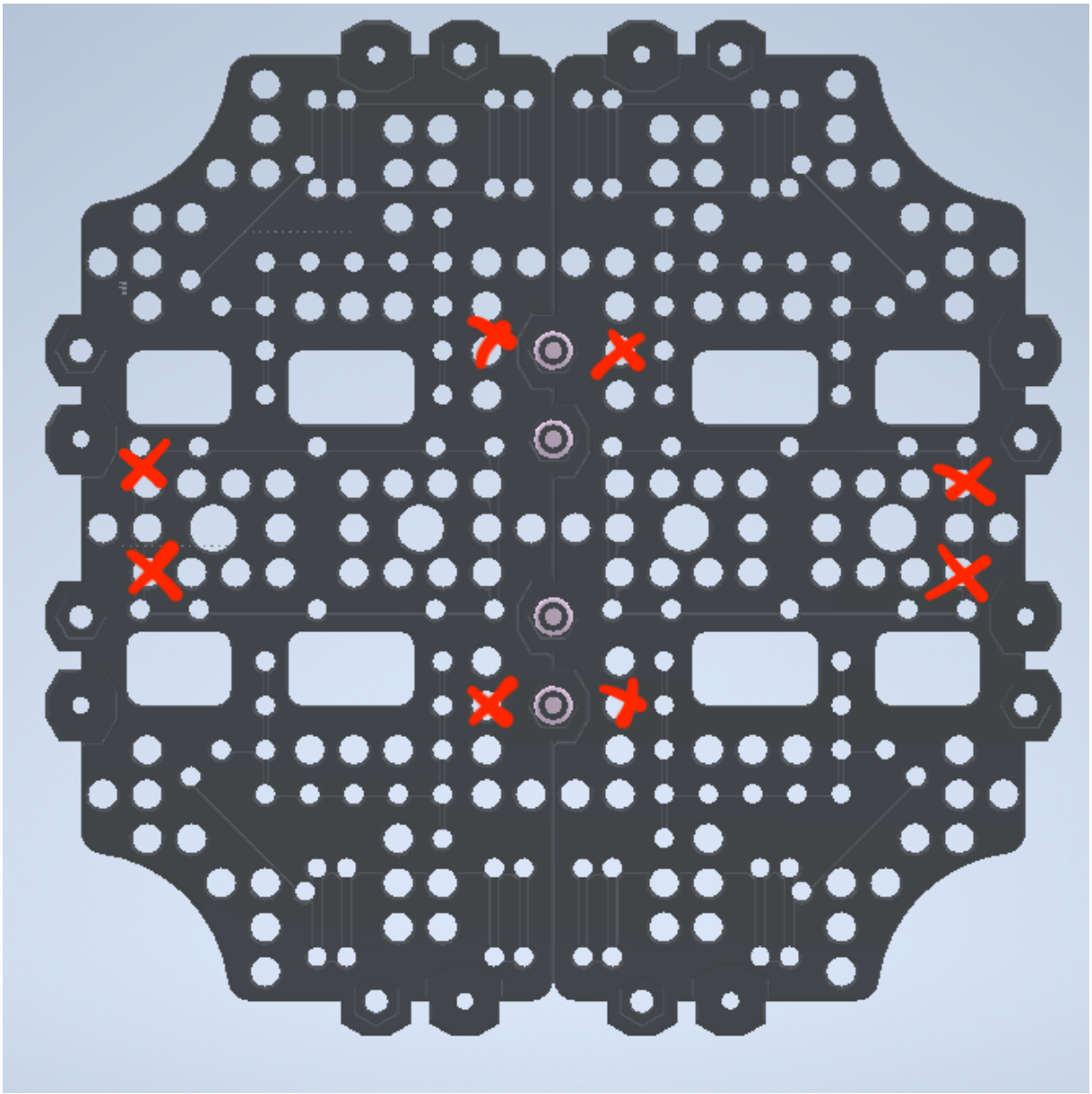


Fig. 21: Waffle Plate with 'X' markings for standoffs

1. On the underside of the waffle plate, 2 ball casters are attached using Plate Supports and PH_T2x6mm_K bolts in the orientation illustrated in figure X.
2. 8 standoffs of M3x8mm are placed on the waffle plate as per figure 19 and 20. This ensures alignment with the holes on the container.
3. The limit switch is placed in the slot of the bottom plate and fixed in position using hot glue adhesive and reinforced with electrical tape.
4. The bottom plate is placed over the standoffs and M3x8mm screws secure it in place.
5. Two plates are held together at a time in position and using a brush acrylic glue is to be put along the edge of the two plates.
6. Repeat step 4 for all the plates' assembly as shown in figure 19.

10.1.5 Dispenser Assembly

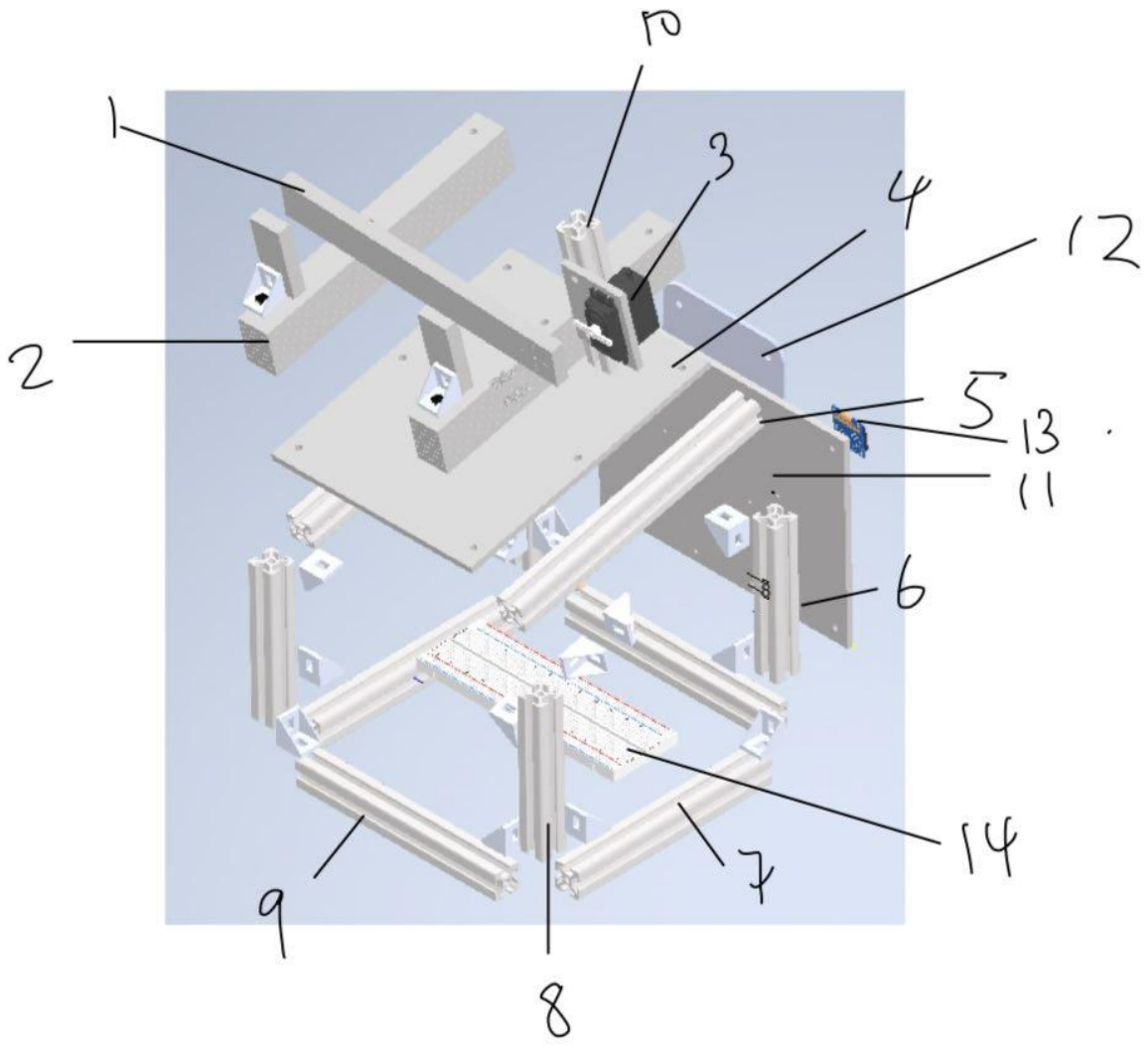


Fig. 22: Exploded view of the Dispenser Assembly

	Parts List	
Items	Quantity	Part Name
1	1	Servo Arm
2	14	Acrylic Railings
3	1	Servo MG995
4	1	Flap that supports servo arm
5	2	Aluminium Extrusion Bars (204mm)
6	2	Aluminium Extrusion Bars (150mm)
7	2	Aluminium Extrusion Bars (160mm)
8	2	Aluminium Extrusion Bars (165mm)
9	2	Aluminium Extrusion Bars (110mm)
10	1	Aluminium Extrusion Bars (112mm)
11	1	Back Board
12	1	Keypad
13	1	OLED Display
14	1	Breadboard

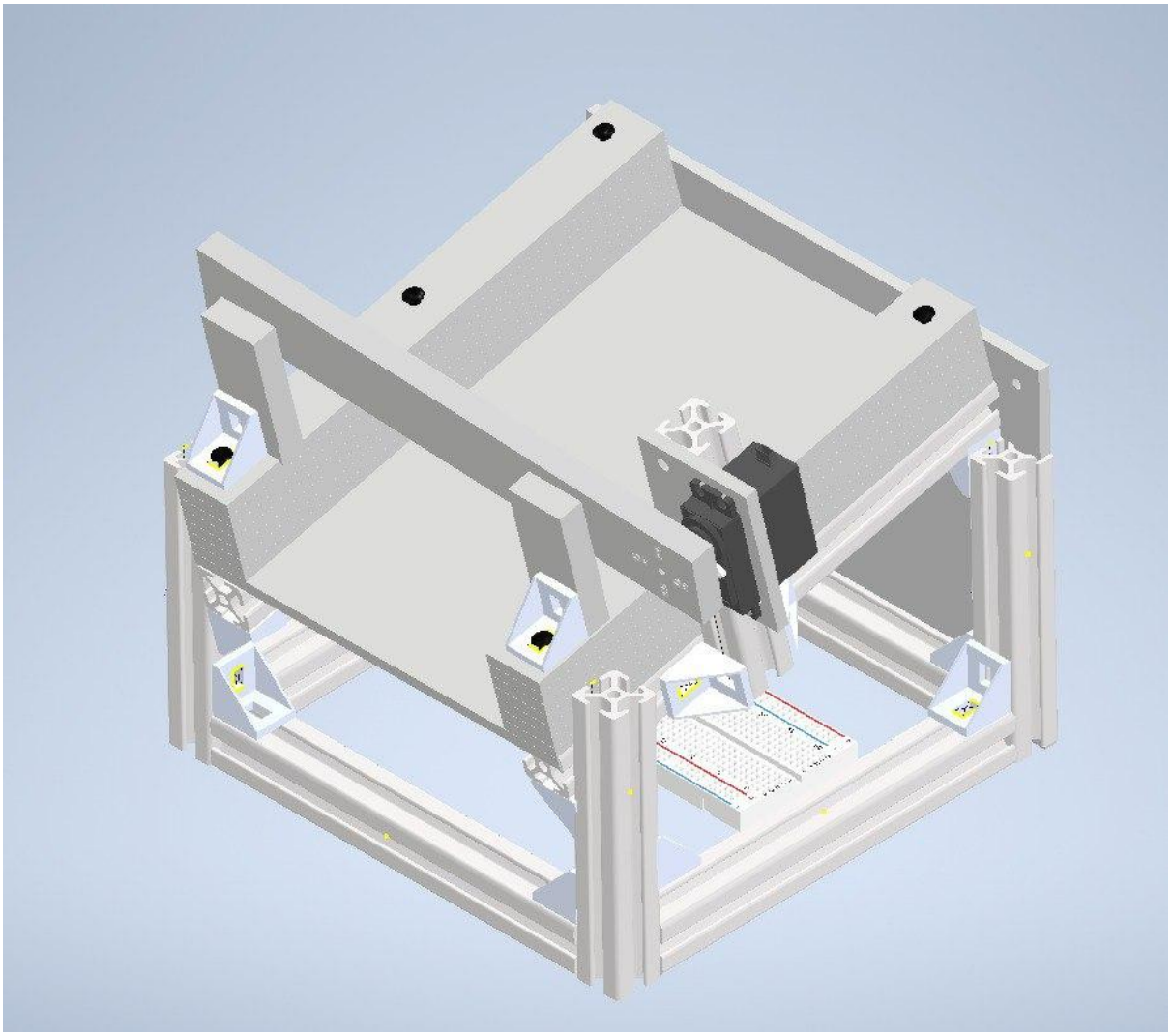


Fig. 23: Isometric view of the Dispenser

1. Form the base of the dispenser using the 20mm x 20mm aluminium extrusion bars. Connect the aluminium extrusion bars using M5 screws with M5 rhombus nuts.
2. Position and fasten the sloped extrusion bar at an angle of approximately 11º to 14º to the horizontal.
3. The dispenser back wall is fixed in position using M5 screws and M5 rhombus nuts.
4. In the slot provided for the keypad, insert the keypad and secure it using 4 PH_M2x12 screws.
5. Insert the OLED display into the left slot on the back wall and secure it using 4 M2x12 screws.
6. The sloped plate is positioned on top of the sloped extrusion bars. 7 acrylic railings are then stacked on top of the slope plate such that the holes are aligned. The 4 upper holes on the slope are fixed in position with M5x40 screws.
7. For the bottom 2 holes, place brackets on top of them. Then screw M5x40 screws through it.
8. On one side of the flap that supports the servo arm, make countersunk depressions 14mm from the bottom and 9mm form the right along the slots. Flat head M5 screws

are then used to hold the flap that supports the servo arm and the acrylic brackets together.

9. The ‘flaps that support servo arm’ are placed vertically along the acrylic bracket installed in step 7.
10. The 112mm extrusion bar is placed perpendicular to the sloped plate at a distance of 40 mm from the flap that supports the servo arm.
11. Place the servo-extrusion connector such that its holes are aligned with the 112mm extrusion bar. Refer to figure X for orientation. The centre of the servo extrusion connector’s top hole is 20mm above the acrylic railing. Then, use M5 screws with M5 rhombus nuts to hold this set up in place.
12. Place the MG995 Servo in the slot of the servo extrusion connector. The holes on the servo are aligned with the holes on the servo extrusion connector and the M2x20mm screws are put through it.
13. The servo horn is fitted onto the MG995 servo.
14. Servo arm is attached to the servo horn using M2 x12mm screws.

10.2 Electrical Setup

10.2.1 Dispenser

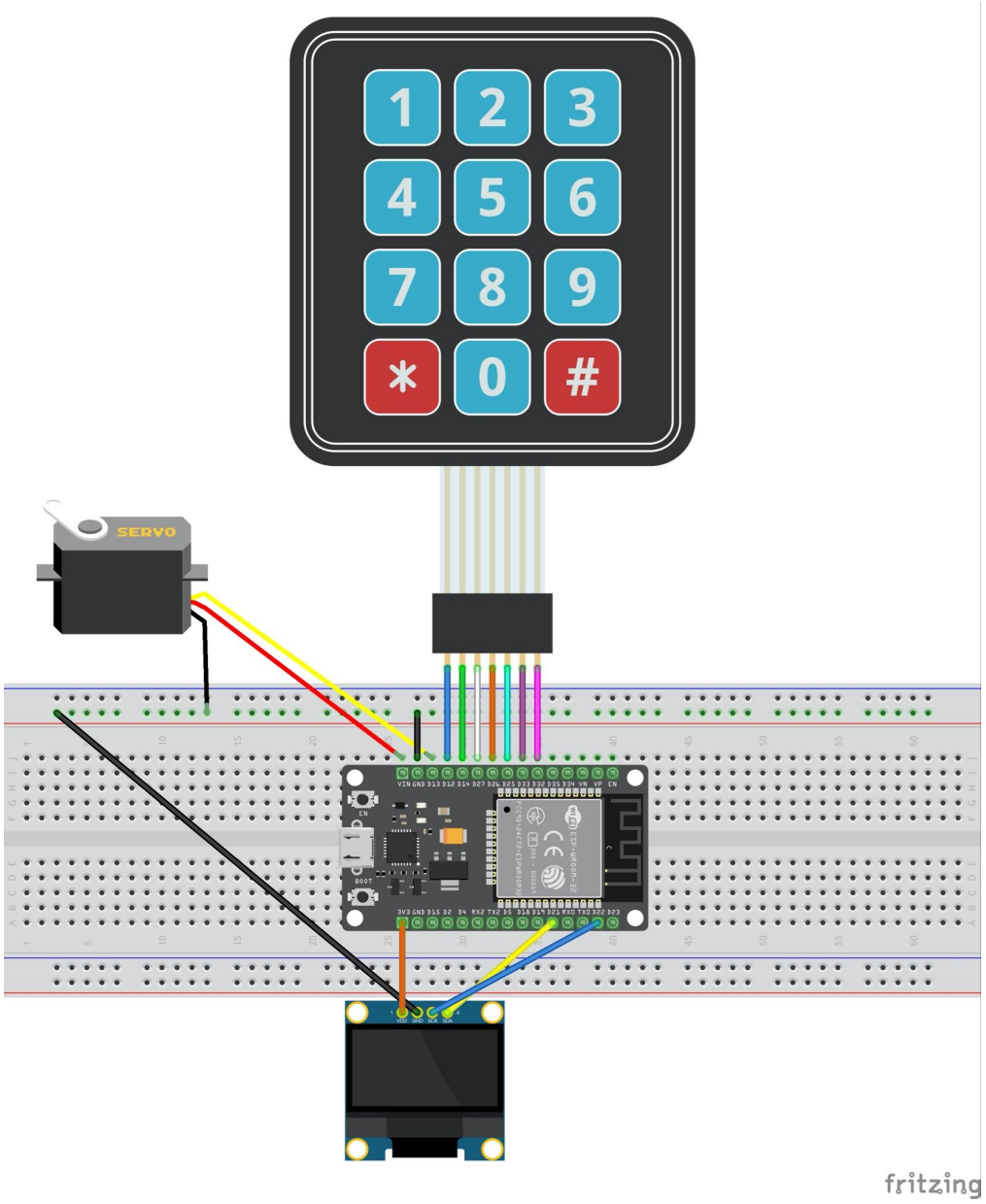


Fig. 24: Electrical Assembly for Dispenser

Pinouts:

Keypad	OLED (I2C)	Servo
Row 1 - 4: GPIO 33, 12, 14, 26 Column 1 - 3: GPIO 25, 32, 27	Vin: 3.3 V SDA: GPIO 21 SCL: GPIO 22 GND: Ground	Signal: GPIO 13 VCC: 5 V GND: Ground

10.2.2 Delivery Bot

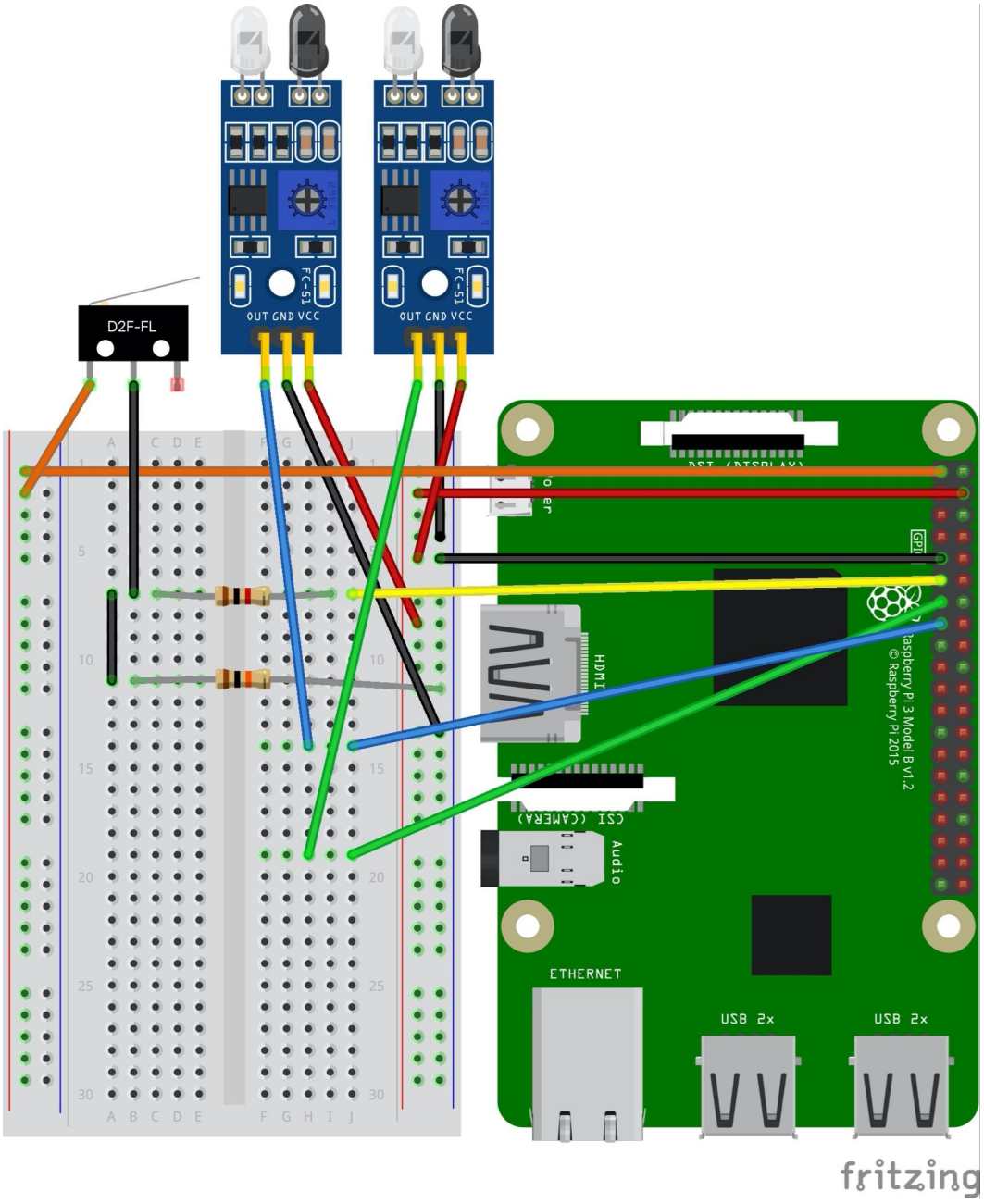
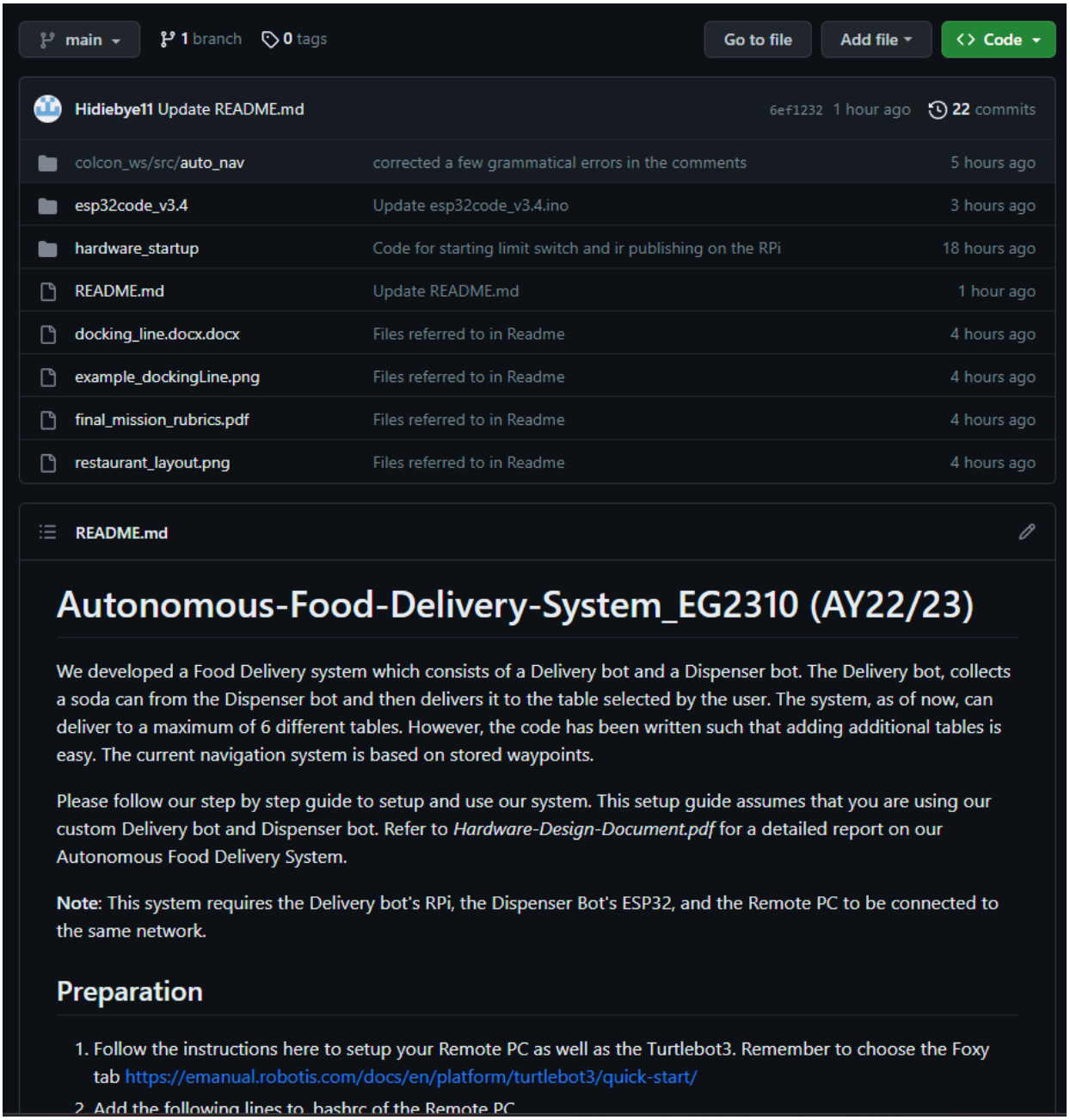


Fig. 24: Electrical Assembly for Delivery Bot

Pinouts:

Limit Switch (Normally Open)	Infrared Sensors
Vin: 3.3 V Current-limiting Resistor: 1 kΩ Pull-down Resistor: 10 kΩ INPUT: GPIO 17	VCC: 5 V OUTPUT: GPIO 27 (Left), GPIO 22 (Right) GND: Ground

10.3 Software Setup



Visit our Github repository's [7] README file for the step by step guide on how to set up and run the software for our Autonomous Food Delivery System. A description of the parameters used for the navigation script is included at the end of the README file.

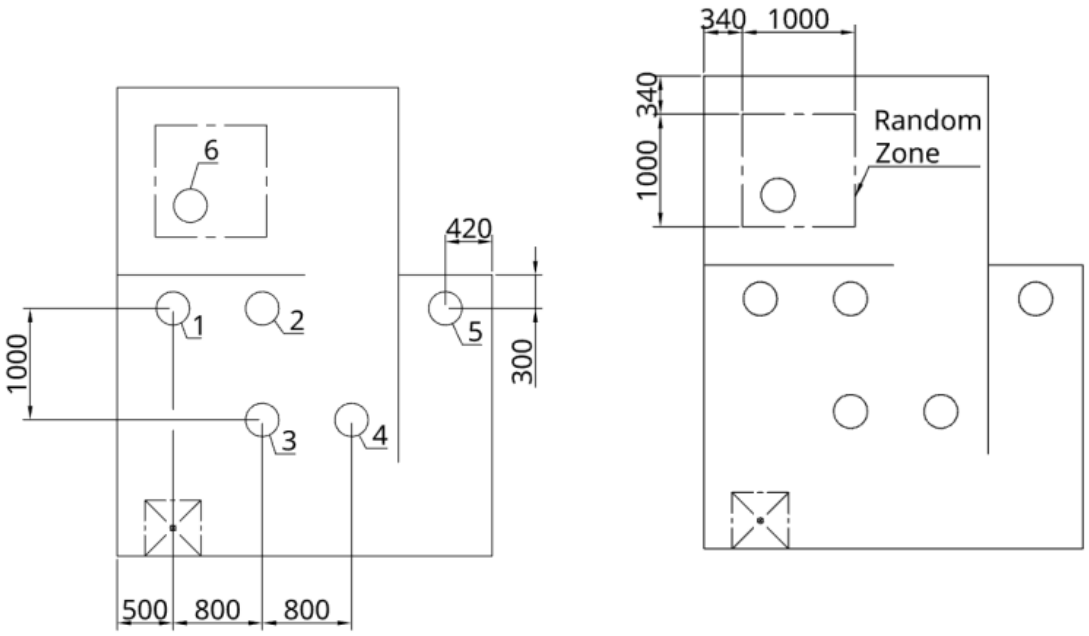
11.Conclusion

A final graded test was conducted on 11th April 2023 in NUS Engineering Block E2A Level 3, Studio 4.

The configuration of the tables within a designated area as set by the teaching staff is shown below:

The objective of this run was to ensure:

1. Navigate to all 6 tables, stopping approximately 15 cm near the table.
2. Complete setup and objective 1 within 25 mins in the fastest time possible.
3. Avoid crashing into walls and tables (and Jared).



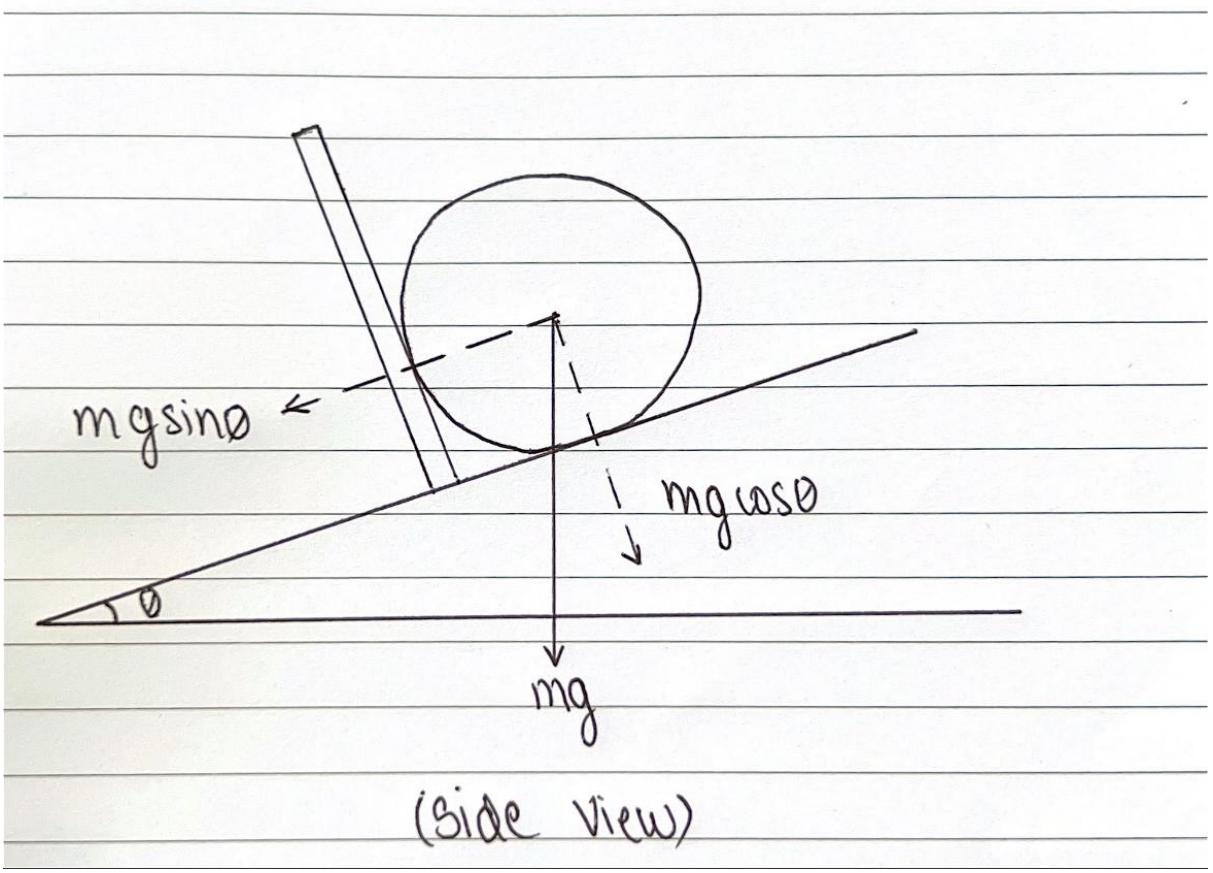
The only issue we encountered in an otherwise perfect run was colliding with table 4 when returning to docking position from table 6 as the Delivery bot overshot a waypoint and did not correct itself in time.

The results of our graded run have proven that our Delivery and Dispenser bot were able to clear all objectives set by the problem statement.

However, we were unable to achieve the fastest run and to this end we could further optimise the linear and angular velocity used during navigation to reduce travelling time to and fro tables whilst not compromising the accuracy of navigation.

Appendix

Annex A (Calculations and Analysis)



As shown in the figure illustrated above, there will be a can of mass ‘m’ placed on an inclined surface with an angle of incline of θ (theta). After prototyping and testing, an angle will be decided for the incline based on material of the plane. For now, we shall refer to the angle as θ. The weight of the can will act horizontally downwards. If we split it into the components along the axes of the inclined plane, we observe that a force of mg cos θ acts normally on the surface while the other component mg sin θ acts on the barrier but at an angle. Thus, by using this orientation of the barrier, we minimise the force being exerted on it and reinforce structural integrity.

Annex B (Bill of Materials)

No.	Part	Quantity	Unit Cost (in $SGD)	Total Cost (in $SGD)
Turtlebot3 (Mechanical)				
1	Waffle Plate	8	Provided	Provided
2	Wheel	2		
3	Tire	2		
4	M2.5 Nuts (0.45P)	20		
5	M3 Nuts	16		
6	Spacer	4		
7	Rivet (Short)	14		
8	Rivet (Long)	2		
9	Plate Support M3x35	3		
10	Plate Support M3x45	10		
11	Adaptor Plate	1		
12	Adaptor Bracket	5		
13	PCB Support	12		
14	PH_M2x4mm K	8		
15	PH_M2x6mm K	4		
16	PH_M2.5x8mm K	16		
17	PH_T 2.6x12mm K	16		
18	PH_M 2.5x16mm K	4		
19	PH_M 3x8mm K	44		
20	Rear Ball Caster (w/ Ball)	0		
21	Breadboard	1		
Turtlebot3 (Electrical)				
21	Li-Po battery	1	Provided	Provided
22	Li-Po battery charger	1		
23	USB Cable	2		
24	Dynamixel to OpenCR Cable	2		
25	Raspberry Pi 3 Power Cable	1		
26	Li-Po Battery Extension Cable	1		

27	SMPS	1		
28	AC-Cord	1		
29	Prototyping Board	1		
30	Dynamixel XL 430	2		
Turtlebot3 (for Software)				
	OpenCR 1.0	1	Provided	Provided
32	Raspberry Pi 3	1		
33	360 Laser Distance Sensor LDS-01	1		
34	USB2LDS	1		
35	Push Button	1		
Dispenser				
1	ESP32-DevKit V1	1	0	0
2	Aluminium Extrusion bars	1.690 m	0	0
5	20x20 Corner Bracket	16	0	0
6	Micro-USB power source	1	0	0
7	4x3 keypad	2	0	0
8	OLED SSD1306	1	0	0
9	Servo Motor MG995	2	0	0
10	Limit Switch D2F-01L	2	0	0
11	Infrared Sensor HW-201	2	0	0
12	M5 Rhombus Nuts	40	0.50	20
13	M5 x 8 Allen Socket Head Screws	34	0	0
14	M2 x 32mm Phillips (Cross) Pan Head	4	0	0
15	M3 x 8 mm Allen Flat Head Screw	1	0	0
16	M5 x16 mm Allen Countersunk Screws	2	0	0
17	M5 x 40 mm Allen Countersunk Screws	6	0	0
18	M5 Hex Nuts	2	0	0
19	M2 x 12mm Phillips (Cross) Pan Head screws	8	0	0
20	M2 Hex Nuts	8	0	0
21	M2 x 8mm Phillips (Cross) Pan Head Screws	4	0	0
22	Breadboard	1	0	0

Delivery Bot Storage System				
1	Waffle Plate	1	0	0
2	Caster Ball	2	0	0
3	Plate Support M3x15	8	0	0
4	Sponge	2	0	0
5	M3 Hex Nuts	12	0	0
6	M3 x 8mm Phillips (Cross) Screws	10	0	0
7	M2 x 6mm Phillips (Cross) Screws	8	0	0
8	Infrared Sensor HW-201	2	0	0
Fabrication				
1	Sawing Extrusion Bars	1 hour	$SGD30/hour	30
2	Laser Cutting Acrylic Sheet	1 hour	$SGD 0/hour	0

Annex C (References)

Using the APA referencing style, the citations would be formatted as follows:

[1] Adlink-ROS. (2023). apriltag_docking. [Online]. Available at: https://github.com/Adlink-ROS/apriltag_docking [Accessed 25 January 2023].

[2] Olson, E. (n.d.). AprilTag: A robust and flexible visual fiducial system. [Online]. Available at: https://april.eecs.umich.edu/software/apriltag [Accessed 25 January 2023].

[3] Adlink-ROS. (2023). neuronbot2. [Online]. Available at: https://github.com/Adlink-ROS/neuronbot2 [Accessed 25 January 2023].

[4] Adlink-ROS. (2023). BT_ros2. [Online]. Available at: https://github.com/Adlink-ROS/BT_ros2 [Accessed 25 January 2023].

[5] MQTT.org. (n.d.). MQTT - The Standard for IoT Messaging. [Online]. Available at: https://mqtt.org/ [Accessed 21 April 2023].

[6] ROBOTIS. (2022). TurtleBot3 Specifications. ROBOTIS e-Manual. [Online]. Available at: https://emanual.botis.com/docs/en/platform/turtlebot3/features/#specifications [Accessed 21 April 2023].

[7] Vibes-863. (2023). Autonomous-Food-Delivery-System_EG2310. [Online]. Available at: https://github.com/vibes-863/Autonomous-Food-Delivery-System_EG2310.git [Accessed 21 April 2023].

Annex D (Servo torque calculations)

Variables

Mass of can (m1)= 350g

Mass of barrier (m2) = 1.19g/cm3 x 22.1cm x 5cm x 0.5cm = 65.7g

Length of barrier = 22.1 cm = 0.221 m

Length of CG of barrier (d) = 11.05 cm = 0.1105 m

Calculation - Torque

Torque of can = μ x m1 x g sin θ x d = 0.61 x 0.35 x 9.81 sin 11.31º x 0.1105 = 0.04534 Nm

Torque of barrier = m2 x g x d = 0.07 x 9.8 x 0.1105 = 0.0758 Nm

Total torque = 0.1211 Nm

Calculation -Power

To turn 90 degrees in 1 second,

Angular velocity = 90/180 x π = 1.57 rad/s

Total Power required = Total torque x Angular velocity = 0.1211 x 1.57 = 0.1759W

Annex E (Line used for line following in docking)

Archive

1 Problem Definition

We are tasked to create a food delivery and dispenser bot system that can automatically dispense and deliver a single can of soda. The can of soda is to be delivered to one of the 6 tables while avoiding obstacles. The dispenser bot should be able to hold at least one can and dispense it automatically, communicating the destination instructions to the delivery bot. The delivery bot should be able to receive the can and navigate to the assigned destination table, wait for manual removal of the can, and then return to the dispenser bot to receive the next can.

2.1.1 Two-tier storage

The two-tier storage system as shown at the side is able to keep multiple cans and uses gravity to pack the cans at the bottom. We will be modifying this system by adding a Gatekeeping System to allow the cans to be fed to the delivery bot one at a time.

2.3 Docking System

2.3.1 Spring Lock System

The spring lock system, which is positioned directly under the dispenser, utilises two spring-loaded pins to safely interrupt the motion of the bot. The pins align and secure the bot in place for loading when the bot comes into contact with them. The gate on the dispenser then opens, allowing a can to roll onto the holder of the Turtlebot for delivery. This method, however, not only involves a sudden stop of the bot's motion, which can potentially cause damage to the bot's hardware, but it also requires a significant amount of time, taking up to one minute to complete the entire process.

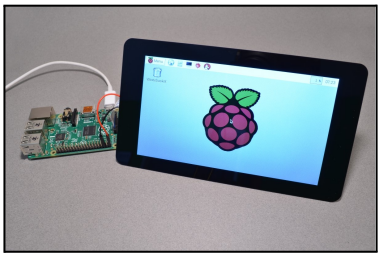
2.3.2 Magnetic Lock System

Initially, magnets were proposed to be incorporated into the dispenser to align the metal strips on the bot and facilitate the unloading of the payload. Although this seemed advantageous due to its cost effective nature, it was decided that the use of magnets in close proximity to electronics could potentially cause disruptions, therefore this feature was removed.

2.3.1 Identifier System (QR Codes/NFC Chips)

Identifiers such as QR codes and NFC will be placed on the ground strategically and a sensor on the delivery bot will align itself to the identifier. This alignment will thus bring the bot to alignment with the dispenser and enable it to receive the soda cans safely. The system will use an on-board camera, infrared sensors and passive base stations. The camera will be responsible for image processing which would give the relative position of the delivery bot with respect to the dock while the infrared sensors will be primarily used for obstacle detection. However, the car must have the computational power to process the sensor data, calculate the position of the QR code and plan the path to follow.

2.4.1 Touchscreen



By utilising a Raspberry Pi's touchscreen display, the dispenser's user interface is made more user-friendly, allowing users to quickly and easily select the destination table number with just a single touch. This enhances the overall user experience, making the dispenser more efficient, and allows for easy editing in case of restaurant expansion, such as adding or editing table numbers.

- Wireless charger adapter / EMI idea (additional)

2.4.3 Decision and Rationale

The LCD Display Module with Integrated Keyboard was chosen as it is a cost-effective and a user-friendly interface for the Dispenser bot. The use of a keyboard is a straightforward and intuitive method for users to input the table number, and the LCD display provides real-time confirmation and visual feedback.

Another option would be to use individual buttons for each table along with LED lights to indicate the selected table. This option is comparatively cheaper than our chosen system, however it is less user friendly and difficult to edit or update (like adding more tables during restaurant expansion). Alternatively, we could have used the Raspberry Pi's touchscreen display as it would have been even more user-friendly and easier to edit and update. However, the touchscreen display is comparatively very expensive and would also be harder to code. Furthermore, the touchscreen display would require us to use a Raspberry Pi for the Dispenser bot as more processing power is needed whereas our chosen system would only require an Arduino Uno hence being more cost efficient.

1.1. Mapping System

2.5.1 Cartographer SLAM

Cartographers help a bot navigate by creating a map of its environment and simultaneously determining its location within that map. The bot then uses this map and its location to plan and execute its movement. A bot using this is usually utilising a set of sensors such as a LIDAR, IMU and cameras to gather data about its surroundings. The bot should also ideally be able to use the map to avoid fixed obstacles and update the map in real time. Due to the multiple sensors that the cartographer SLAM can use to gather data about its environment, it is able to generate an accurate map of its environment.

2.5.2 Adaptive Monte Carlo Localisation

This method takes help of data collected by sensors like LIDAR, IMU, Odometry and Camera to estimate the position and orientation of a bot. The algorithm uses the Monte Carlo technique which involves sampling to generate a set of possible poses for the bot. Each pose is evaluated based on how well it matches the sensor data and the map of the environment. The algorithm then uses a technique called resampling to select the most likely poses. AMCL algorithm is a method of localising a bot using sensor data, map of the environment, and probabilistic techniques. It generates a set of possible poses, evaluates them and resamples to select the most likely one.

2.5.3 Decision and Rationale

Given that TurtleBot 3 is equipped with a built-in LIDAR sensor and that Cartographer is a widely recognized and extensively documented library for creating maps and determining a bot's location, we have chosen to implement Cartographer SLAM in our project. This decision allows us to streamline the process and minimise the required effort.

3.4 Phase 4 (Drop Off Phase)
Objective of Phase: To recognise when the can is removed by the customer manually and to navigate back to the start point, dispenser, automatically. The delivery bot will then initiate docking with the dispenser. Upon successful docking the delivery bot will restart from phase 1 as it waits for the next order.

Description of Phase: The delivery bot will stop at a specified region near the destination table, whereupon it will wait till the user removes the can from the storage compartment. This will uplift the end stop switch of the delivery bot which will run a Python script that utilises SLAM to navigate back to the dispenser whose coordinates will be predefined

3.1 Overview of Phases
We have divided the entire operation into 3 phases namely the Interface phase, Loading phase and Navigation phase. The Interface phase is where the user inputs the table number via the keyboard into the Dispenser. The Dispenser then communicates that location to the Delivery bot. The Loading Phase will include both Dispenser and Delivery bot starting from the manual loading of the can onto the dispenser and ending when the delivery bot has received it. The Navigation phase is the one where the delivery bot navigates autonomously to the correct table while avoiding obstacles. The final phase will be the Drop Off phase where the soda can is removed from the Delivery bot and the bot automatically goes back to the dispenser to collect the next can and restart the process again.

Navigation Phase
We initially begin with an occupancy map of the restaurant that we already produced using Cartographer SLAM while controlling the bot using teleoperation. After the Loading phase, a python script will launch the Nav2 stack to start Navigation as well as Localisation[3]. The Localisation will help update the present occupancy map with new data, if any, such as that of obstacles. After this, based on the destination table, the python script will run one of the Behaviour Trees[4] consisting of the predefined goal (coordinates of the table). The Delivery bot will then begin moving to the goal (table) via a path that avoids obstacles. (To be decided while prototyping: In case the Delivery bot does not reach close enough to the table, the Navigation would end almost near the table after which the bot will use the Pi Camera to run apriltag_docking to calibrate and align itself using unique AprilTags placed on the side of each table). Upon reaching the destination table, it will remain stationary until the user removes the can from the Container. This will raise the End Stop Switch of the Delivery bot which will run a Python script that reruns Navigation and Localisation to navigate back to the Dispenser. When the Delivery bot nears the Dispenser, its R-Pi should send a signal to turn on the R-Pi camera. The camera will be used to detect the AprilTag located on the Dispenser and align itself with it. This will ensure that the Delivery bot will dock at the Dispenser in the correct position and orientation.

Instructions:

a. Initialisation:

1. User places the Turtlebot at the docking point of the dispenser.
2. The hotspot configured with the system needs to be turned on.
3. User will flick the switch on the OpenCR.
4. User will enter ‘sshrp’ in two tabs of Linux Bash terminal of their computer to SSH into the Raspberry Pi.
5. User will enter ‘rosbu’ and ‘sensordata’ on two terminals on their computer..
6. User will enter ‘rslam’ and ‘map2base’ on two terminals on their computer.
7. Instruction: Waypoints, moveToGoal.

Subsystem	Test	Description	Expected Outcome
Software	To test the communication of the RPi and ESP32	RPi publishes a message to a topic that the ESP32 subscribes to.	LED connected to ESP32 blinks
	Navigation of Delivery Bot to the specific table	To ensure that the Delivery Bot accurately travels to the coordinated stipulated by the table number	Delivery Bot accurately travels to the coordinated stipulated by the table number
Mechanical	Barrier is able to stop and release a can	To ensure that the barrier is able to withstand the force when the can is rolled from the highest point on the ramp	The can stops and releases on detection of the Delivery Bot
	Can is able to roll down the slope in the correct orientation	Roll a can down the slope and video it to see the orientation of release	Can is parallel to the edge of ramp when released

Temp

4.1 Software Changes

4.1.1 Interface system

4.1.2 Navigation algorithm

Odometery to map2base

Path correction algorithm

- Added waypoints

Table 6 handling algorithm

4.1.3 Docking algorithm

4.2.3 Servo

We changed the model (MG90s) of the servo motor to a model (MG995) available in the Electronics Lab. MG995 is more powerful, thus eliminating the possibility of it being stalled. Our power budgeting was able to accommodate the higher power requirement of MG995.

4.2.4 Pi Camera

We were unable to interface the Pi Camera after days of trying. Eventually, we opted for the Line Following method for docking instead of AprilTag for convenience.

4.2.5 Infrared Sensors

As we switched to the Line Following method for docking, we added 2 infrared sensors at the back of the delivery bot to detect the black line. The black line absorbs infrared radiation emitted by the emitter on the sensor, thus the receiver will not receive the infrared radiation. The bot will rotate itself until both receivers do not receive infrared radiation, then it will move forward until it docks

4.2.4 Limit Switch

We changed the model (V-155-1A6) of the limit switch to a model (D2F-01L) available in the Electronics Lab. The maximum operating force is lower (1.47 N), hence we decided to place a sponge on top of the limit switch to cushion the impact.

Infrared Sensors

Due to the decision to change AprilTags as a means of docking, an alternative line following approach was adopted by us. For this, 2 infrared sensors were fixed in place below the container by adhesives at a calculated distance from each other. The distance between the two infrared sensors was the same as the width of the line that it was following.