EG2310

Group 3

G2 Report

Student Team Members:

Tan Ping Hui	A0272169X
David Michael Indraputra	A0276057X
Peteti Harshith	A0288271W
Song Cheng Yan	A0276267R
Basudeb Chakraborty	A0288113E

Table of Contents

1. Introduction

This manual presents a comprehensive overview of the TurtleBot, an advanced robotic platform engineered to perform various tasks efficiently and precisely. At the core of TurtleBot's functionality is integrating the Robot Operating System (ROS), a versatile middleware suite that facilitates complex robotic operations. The TurtleBot's design is centered around a custom chassis, which houses an array of sensors and actuators tailored to enhance its operational capabilities.

Its dual operational objectives distinguish the TurtleBot: Firstly, to autonomously navigate through intricately designed mazes to achieve the shortest possible completion time. This involves the utilisation of advanced pathfinding algorithms and dynamic environmental adaptation strategies. Secondly, the TurtleBot is tasked with executing a precise object manipulation operation, specifically launching ping pong balls into a designated container, showcasing its ability to perform targeted actions with high accuracy.

- Advanced Navigation System: Employing two infrared (IR) sensors mounted on its underside, the TurtleBot leverages environmental markers for navigation, enabling it to traverse complex mazes efficiently.

- Remote System Interaction: The TurtleBot can interface with external systems by making HTTP requests to unlock doors within its operational environment, exemplifying its integration with Internet of Things (IoT) technologies.

- Precision Dropping Mechanism: Equipped with a dropping assembly, the TurtleBot precisely drops all five balls into the bucket by release the servo motor which is blocking the balls.

TurtleBot's operational framework and system design reflect a harmonious blend of mechanical engineering, software development, and robotics. This manual delves into the technical specifications, operational procedures, and maintenance guidelines necessary to fully harness TurtleBot's capabilities.

2. Problem Definition

The goal of this challenge is for the robot, TurtleBot, to complete a series of tasks within a 20-minute time frame. These tasks involve navigating a maze, interfacing with a secured door via HTTP, and performing a precise task within a designated room. Below, the tasks are delineated into detailed subtasks:

2.1 Maze Navigation

The TurtleBot starts at a designated point and must navigate through a maze of randomly placed obstacles. The objective is to find the exit efficiently using its onboard navigation systems.

2.2 Mapping

The TurtleBot is tasked with mapping the maze by identifying and documenting its structural elements. It will employ Simultaneous Localization and Mapping (SLAM) techniques to create an accurate, comprehensive map, ensuring full map closure for effective navigation.

2.3 Door Access

Upon exiting the maze, the TurtleBot will approach two rooms, each secured with an electronically locked door. To gain entry, the TurtleBot must make a successful HTTP request to a web server that controls the locks. A valid request triggers the unlocking of one door, and the server responds with the ID of that door, guiding the TurtleBot to the correct room.

2.4 Ball Dropping

Inside the unlocked room, the TurtleBot launches five ping-pong balls into a designated bucket, demonstrating precision and control.

2.5 Optional Markers

The mission may be enhanced by placing temporary markers at the start. These markers act as reference points within the environment, aiding the TurtleBot in identifying key locations and navigating more effectively.

3. Literature Review

3.1 Navigation

3.1.1 Navigation using IR sensors

The technique used in navigating a maze with a line-following TurtleBot involves integrating two line sensors, and algorithms to achieve precise and efficient movement. The TurtleBot employs two infrared sensors and black tape, which will be used as a marker to detect the contrast between a marked line and the surrounding surface. These sensors provide real-time data processed by the TurtleBot's onboard Raspberry Pi (RPI) to discern the TurtleBot's position relative to the line. Adjustments in the TurtleBot's path are executed through differential wheel speeds, enabling it to make sharp turns, follow curves, and handle intersections.

3.1.2 Navigation using path planning algorithm

These algorithms enable the identification of the most efficient routes through a maze while adeptly recognising and avoiding potential obstacles. The A* Search Algorithm and the Rapidly exploring Random Tree (RRT) are two prominent algorithms used for this purpose.

A Search Algorithm*

The A* Search Algorithm is a highly effective, graph-based search method that explores the search space meticulously. It evaluates each node by combining the cost of reaching it from the start with an estimated cost from the node to the goal. This algorithm employs heuristics, which guide the search process towards the goal more efficiently, making it especially useful in environments where a clear and optimal path needs to be determined.

Rapidly Exploring Random Tree (RRT)

Rapidly Exploring Random Tree (RRT) is an innovative, sampling-based algorithm for path planning. Unlike methods that exhaustively examine every possible path, RRT grows a tree rooted at the initial configuration and expands by randomly sampling the space around it. Each new node is connected to the nearest existing node in the tree, efficiently exploring the configuration space. This method is particularly advantageous in complex or high-dimensional spaces where traditional algorithms need more time efficiency and computational load.

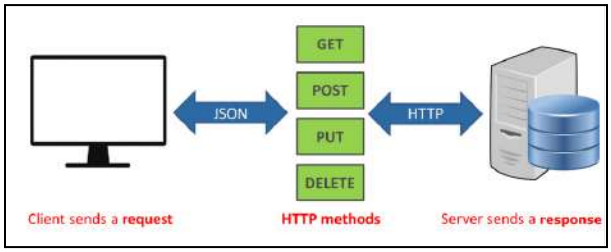
3.1.3 Decision and Rationale

Our group collectively decided to adopt line following for navigation, using black tape as a marker to outline the shortest possible route through the maze. Manually designing the most direct pathway could significantly reduce navigation time and enhance the speed of our TurtleBot. This approach leverages human intelligence in planning the path, bypassing more complex algorithms. Our strategy aims to expedite the completion of the task, thereby maximising our score and improving our competitive edge in the challenge.

3.2 Entering the Locked Room

3.2.1 HTTP Call

HTTP, an application layer protocol, exchanges information between networked devices, such as the TurtleBot and an electric lock system linked to a web server. This protocol facilitates communication via requests that include a URL and an HTTP method, like POST or GET, which specify the server's required actions. A POST request is typically used to unlock doors. This request targets a specific endpoint on the server dedicated to door control operations, such as "https://your-door-server.com/unlock". When this request is sent, the web server processes the incoming data, interprets it, and interacts with the electric lock system to trigger the unlocking mechanism. Upon successful validation, the system unlocks one of the doors, and the server sends a confirmation signal back to the TurtleBot, specifying which door has been unlocked.



3.2.2 Locate Unlocked Room

Once the TurtleBot has navigated through the maze, it will come across a T-shaped configuration of black tape. This configuration leads to minimal IR light reflection back to the TurtleBot's sensors. Detecting this unique pattern, the TurtleBot stops and makes an HTTP call to determine which door has been unlocked. Based on the server's response, which includes the ID of the unlocked door, the TurtleBot will then decide to turn either left or right. This turn guides the TurtleBot through the correct door, seamlessly continuing its mission.

3.3 Mapping

The TurtleBot has an LDS-02, a 360-degree Light Detection and Ranging (LiDAR) sensor. This sophisticated sensor uses pulsed lasers to accurately measure distances around its entire perimeter, facilitating SLAM (Simultaneous Localization and Mapping) and precise navigation.

3.3.1 Mapping Algorithms

GMapping

GMapping is an advanced algorithm that simultaneously generates a grid map of the environment and determines the TurtleBot's precise location. It harnesses sensor data to continually update the map's occupancy probabilities and the TurtleBot's pose. This process is underpinned by a probabilistic framework, specifically the Rao-Blackwellized Particle Filter, which encompasses stages like prediction, measurement update, particle resampling, map updating, and loop closure. These steps collaboratively refine the map and the TurtleBot's positioning cyclically, enhancing accuracy and reliability in dynamic environments.

Cartographer

Cartographer is another robust mapping algorithm that is well-regarded for utilising scan matching and loop closure to map environments and localise the TurtleBot effectively. It excels in optimising pose estimates and ensuring the consistency of the map, integrating sensor data to support real-time operation across various robotic applications. Cartographer's sophisticated techniques allow for precise and efficient mapping, making it an ideal choice for complex navigation tasks in fluctuating settings.

3.4 Ping Pong Dispenser

3.4.1 Flywheel Launcher

The flywheel launcher utilises motors characterised by high speed and low torque to spin its wheels rapidly. This design is available in single and dual-wheel configurations, which are adept at accelerating a ping-pong ball as it passes through the rapidly spinning wheels. The precise ejection of the ball is one of the critical advantages of this system, which makes it suitable for applications requiring high accuracy and repeatability.

3.4.2 Spring Launcher

The spring launcher mechanism compresses a spring to store potential energy, which is subsequently released to convert it into kinetic energy that propels the ping pong ball forward. The degree of spring compression can be adjusted, thus enabling control over the launch force. This adjustability allows the user to achieve specific velocities and trajectories of the ping pong ball, tailored to varying operational requirements.

3.4.3 Ball Dropping System

A straightforward and effective method, the ball-dropping system consists of a pipe, the diameter of which is slightly greater than that of the ping pong balls. This pipe is mounted at a declined angle atop a turtlebot, with balls being temporarily halted by a servo motor. When the balls align with the target bucket, the servo motor actuates, allowing the balls to drop directly into the bucket. This system's simplicity and direct nature contribute to its high accuracy and precision.

3.4.5 Decision and Rationale

Initially, we employed a dual-motor flywheel launcher to facilitate quick ball ejection into the target bucket. However, this system presented issues with uneven ball release and inconsistent ejection speeds, prompting a reassessment of our strategy. In response to these difficulties, we shifted to the ball-dropping system during the later phases of our project. This change simplified our operational approach and markedly improved the precision of ball placement, yielding near-perfect accuracy.

4. Concept Design

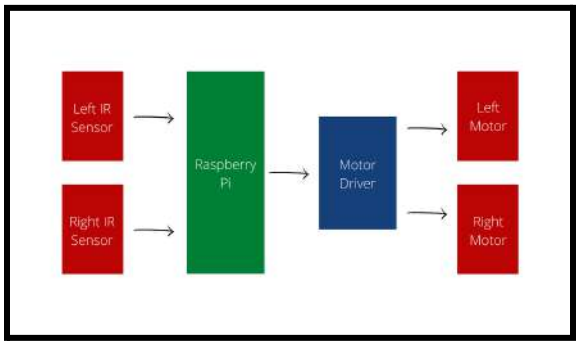
4.1 Line Following System

4.1.1 Infrared Sensor (TCRT5000)

Line navigation will employ IR Sensors positioned at each TurtleBot's end. These sensors comprise a transmitter, responsible for emitting signals, and a receiver, which detects the emitted signals. When the IR sensors emit infrared light onto an object, light absorbed by black surfaces results in low output. In contrast, light reflected by white surfaces returns to the transmitter, detected by the infrared receiver, yielding an analog output.

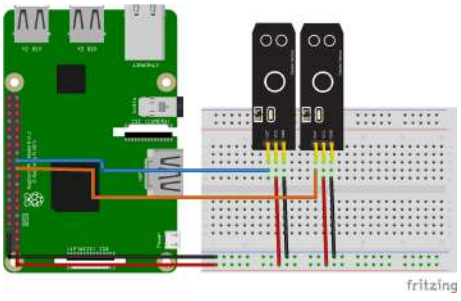
4.1.2 Software

The R-Pi processes input from the IR sensors to control the TurtleBot's motors. When the TurtleBot veers off-course to the left, the right IR sensor detects the black tape, generating a low output. In response, the R-Pi decreases the right wheel's speed, nudging the TurtleBot back on track to the right. The process is reversed for deviations to the right. Upon encountering a T-shaped configuration of black tape, both IR sensors detect it, prompting the R-Pi to halt the wheels completely and prepare for the next command. Detailed descriptions of these scenarios are provided in Appendix A. An example of the code is also provided in Appendix B.



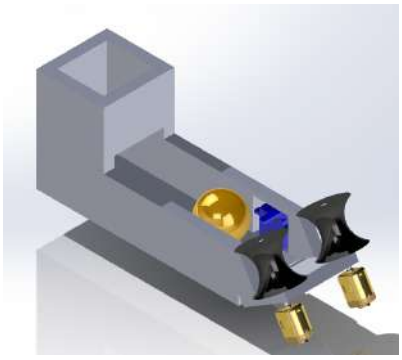
4.1.3 Electrical

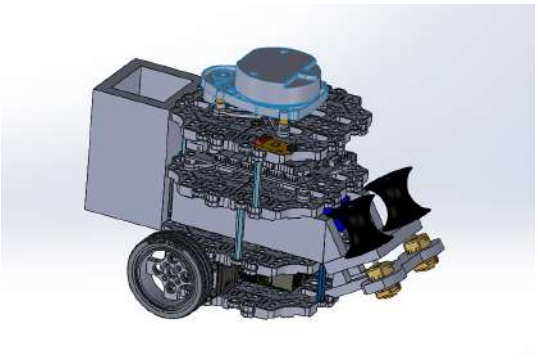
The output pins of the IR sensors will be connected to the GPIO pins on the R-Pi for reading their signals. The IR sensors, TCRT5000, operate at a voltage range of 5V ± 0.25V (Appendix C), which is supported by the output of the R-Pi.



4.2 Flywheel Launcher System

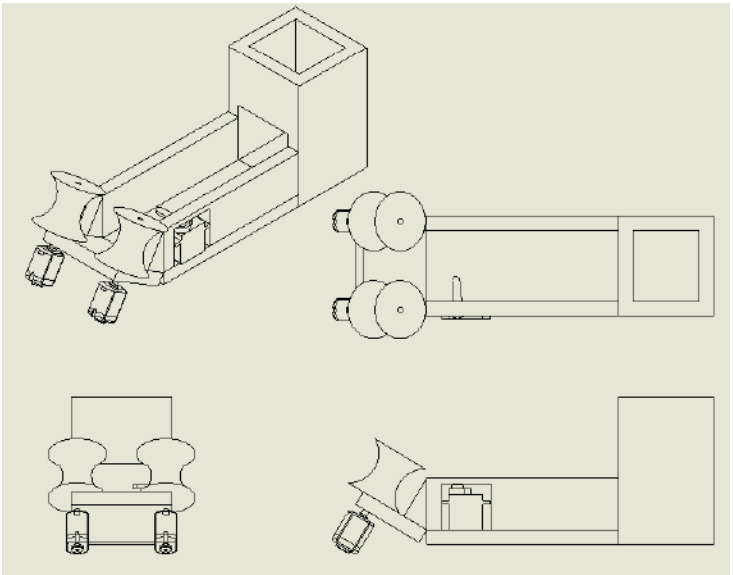
4.2.1. Mechanical



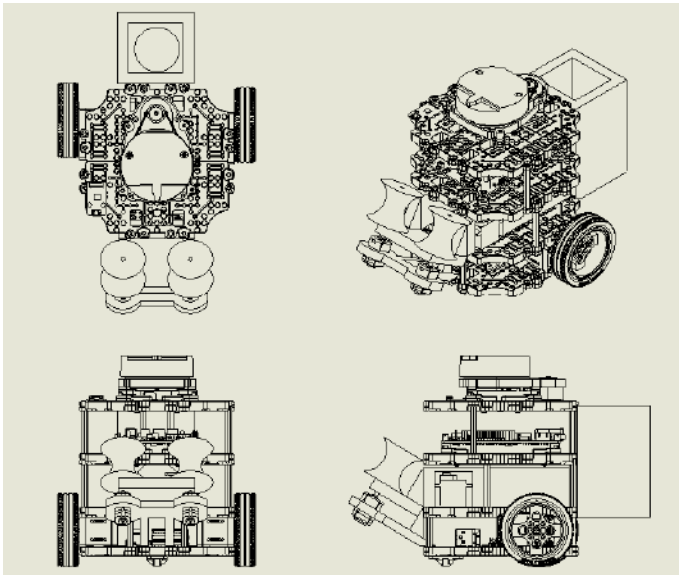


The design incorporates a high-speed flywheel mechanism, which is designed to be able to accomplish the goal of shooting five balls continuously into a pail 50 cm away. It utilizes two 20000 rpm motors that can reach the needs of 17000 rpm. The system stores balls in the second layer of turtlebot and a carriage behind it. When the turtlebot reaches the shooting range, the servo will release the balls after the acceleration of the flywheels. This design intends to keep the center of gravity in the middle so as not to decrease too much speed and avoid having the carriage blocking the lidar when mapping. The figures above show the CAD model of the prototype. More detailed CAD drawings and calculations are provided here:

4.2.1.1 CAD Drawings



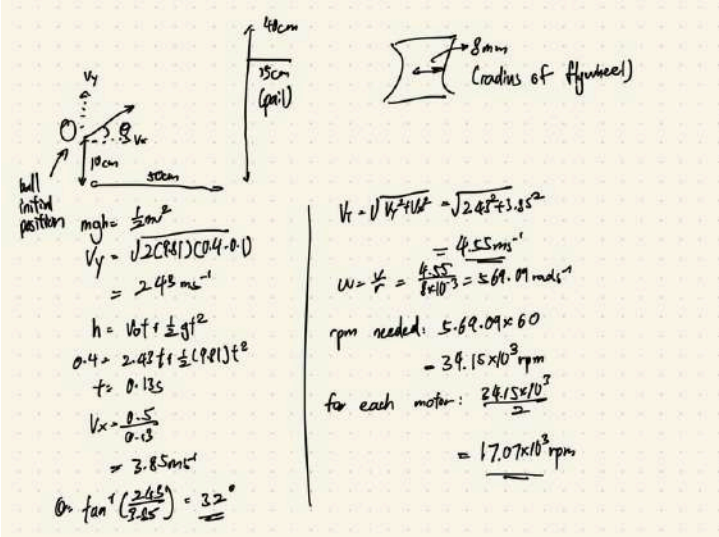
Flywheel System



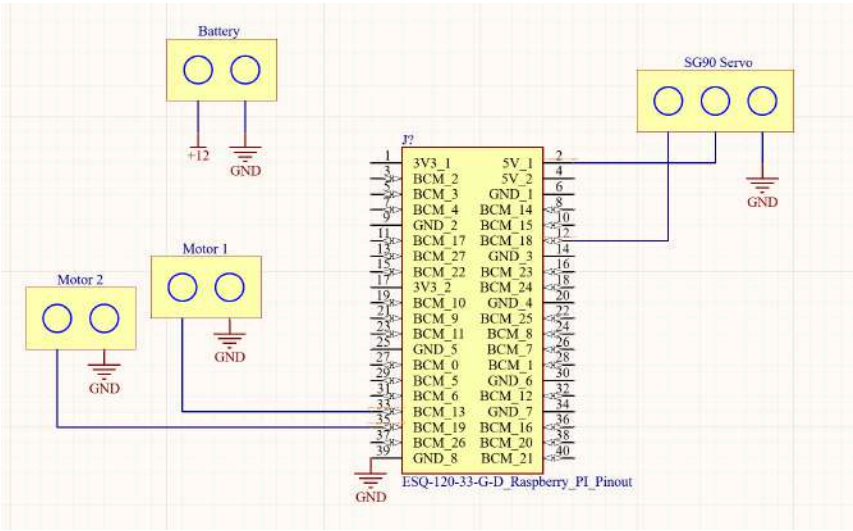
Integration with TurtleBot

4.2.1.2 Calculation for Ping Pong Launching

For the ball to shoot into a 35cm height bucket, both flywheels' motors need to be at least 17000rpm, and the track needs to be tilted up by 32 degrees, as shown in the calculations below. However, to account for air resistance, which was neglected in the calculations, we chose motors that can run up to 20000 rpm.



4.2.2 Electrical



4.2.3 Software

Upon precise positioning of the TurtleBot and alignment with the bucket, the R-Pi will activate the servo to launch all 5 balls continuously, which will be powered by a predetermined voltage for a timed duration. With the completion of this task, all mission objectives are achieved, prompting the automatic cessation of all program operations.

5. Preliminary Design

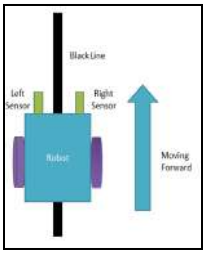
5.1. Phase 1 - Line Navigation

Objective of this phase: To navigate along a predetermined path efficiently, using the shortest possible route.

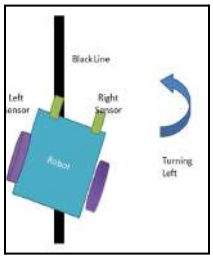
Description of operation: The robot utilizes the two IR sensor inputs to decide its next movement. It will then publish to cmd_vel the respective linear and angular velocities required to move towards that direction. The tape must also be laid strategically to ensure it is the shortest path to the lift lobby area to complete the mission fast.

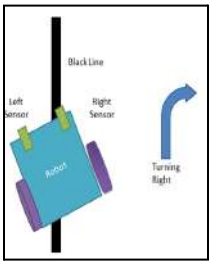
5.1.1 Line Following

The sensors play a pivotal role in guiding the robot through the maze with precision. Their function in various scenarios is as follows:

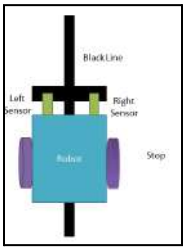


1. Both Sensors Off the Line: In this scenario, when both sensors detect that the line is not beneath them, the TurtleBot will continue moving forward along its current trajectory. This ensures smooth progression through straight sections of the maze.





2. One Sensor On the Line: If only one sensor detects the presence of the line while the other remains off it, the TurtleBot will adjust its direction slightly towards the sensor that is on the line. This nudge ensures that the robot remains centered on the path, correcting its course as needed.

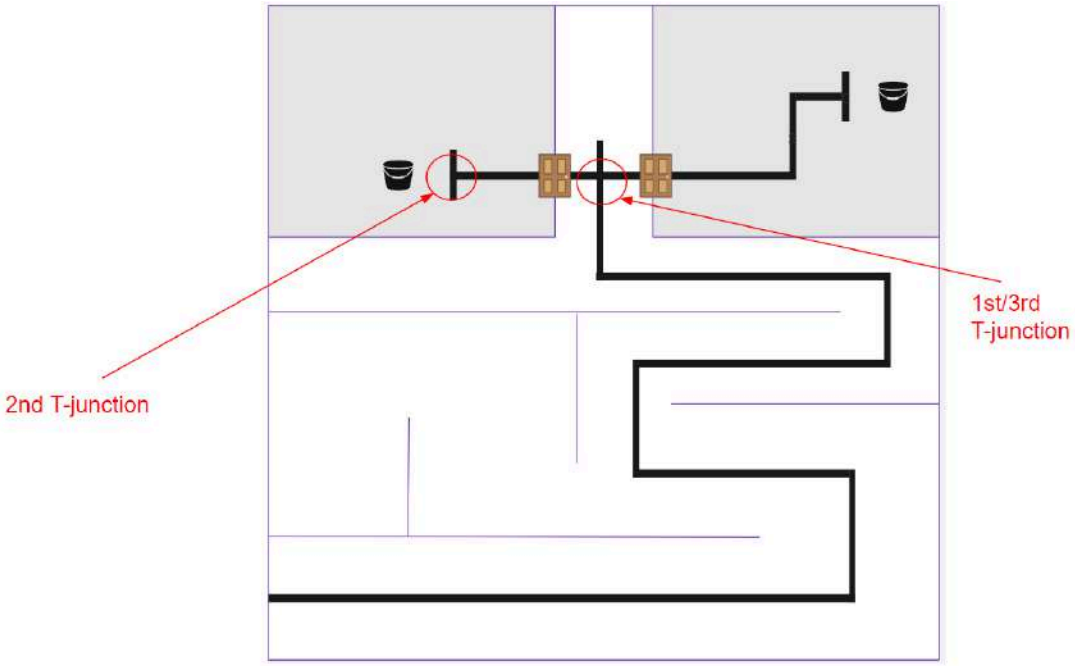


3. Both Sensors On the Line (T-Junction): At T-shaped junctions where both sensors simultaneously detect the line, the TurtleBot will come to a halt. This specific scenario triggers the execution of preconfigured commands, allowing the robot to perform designated actions such as initiating turns, communicating with servers, or activating mechanical components.

5.1.2 Tape Layout

The diagram below is an example of the mission map. Before the mission starts, we will lay tapes such that it is the shortest path from the starting point to the lift lobby area. This ensures that the TurtleBot can complete the mission in the shortest time possible, maximizing the score for competitive mission completion time. Furthermore, as the mission objectives hold a significantly higher percentage in the overall score, the strategy of placing mapping at the end allows us more time to address any unforeseen circumstances that may arise during the mission.

Of particular significance within this mission plan is the utilization of T-shaped tape configurations at specific junctures along the route. These configurations serve as markers for the TurtleBot, signaling moments requiring deviation from standard traversal behavior. First of which is seen between the two doors at the lift lobby area. Second of which is seen before the pails within the 2 rooms. Note that the first is in a “t” shape, so that the TurtleBot will also treat it as a T-junction encounter during its return path from the pail.



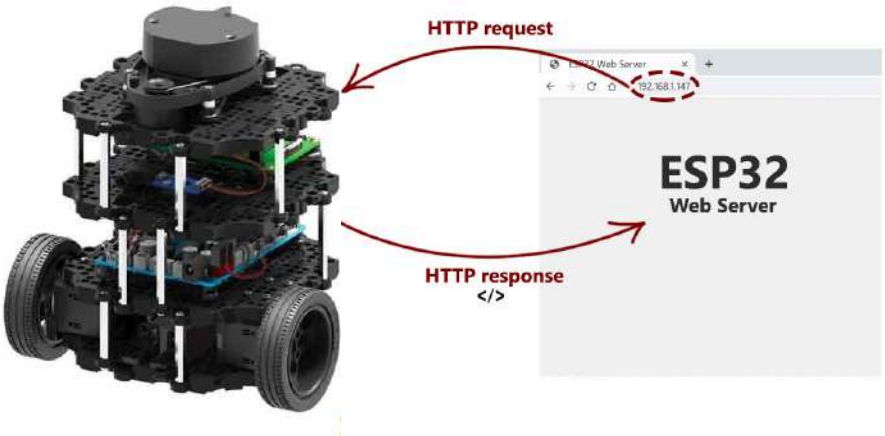
5.2 Phase 2 - HTTP Call

The objective of this phase is to receive the door ID of the lift door that is open for the TurtleBot to navigate through.

Description of operation: The TurtleBot will communicate with an ESP32 local server to decide whether to turn left or right to the unlocked door.

Pseudocode:

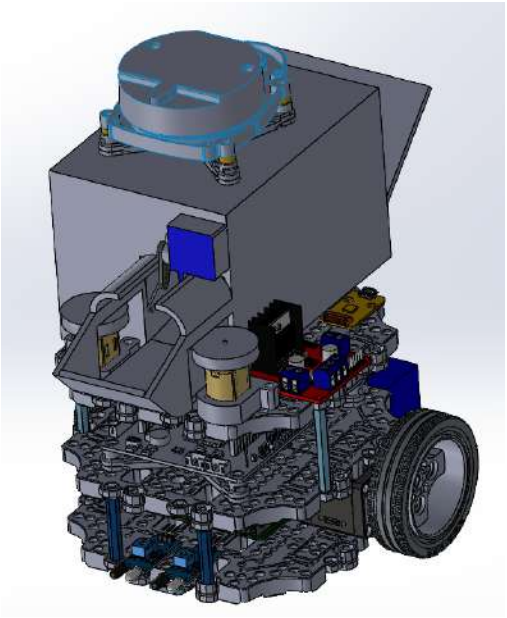
1. **First Junction Detection:** The TurtleBot utilises its onboard sensors to detect junctions within the maze. Upon encountering the first junction, it prepares to make an HTTP POST request to the ESP32 server.
2. **HTTP POST Request:** By executing the make_http_post_request function, TurtleBot sends a JSON payload to the specified IP address. The payload requests an "openDoor" action, identifying the robot with its **robotId**.
3. **Server Response Handling:** The server's response is awaited and checked for a successful HTTP 200 status code. If successful, the response's JSON content is parsed to extract the door instruction, either "door1" or "door2".
4. **Action Outcome:** The TurtleBot adjusts its path towards the directed door depending on the server's response.

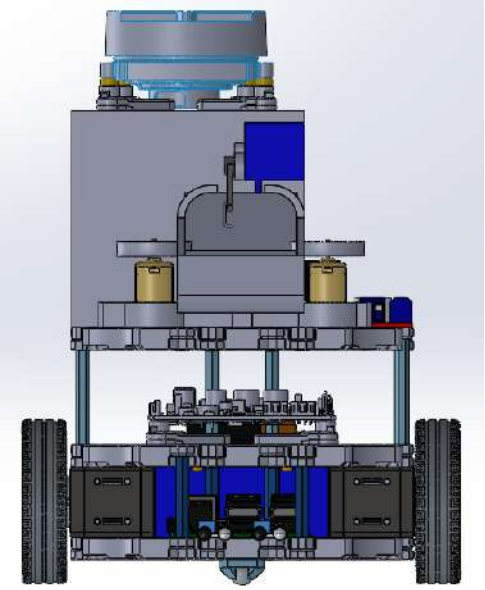


5.3 Phase 3 - Deposit Ball in Pail

This phase aims to deposit five ping pong balls into a pail within the room.

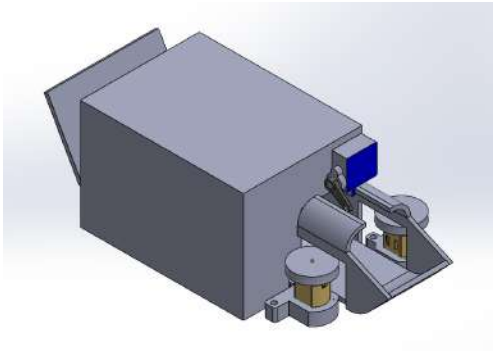
Description of operation:

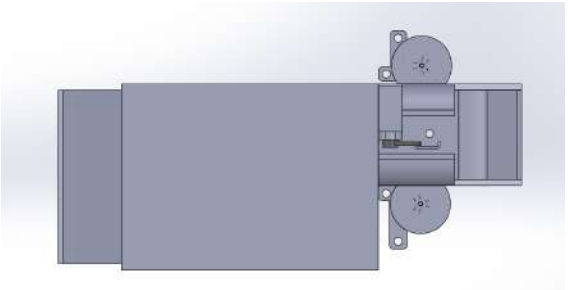


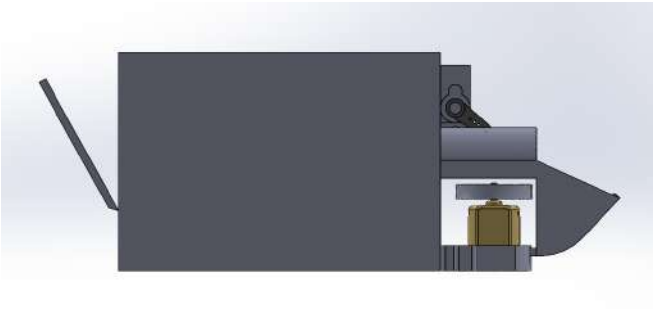


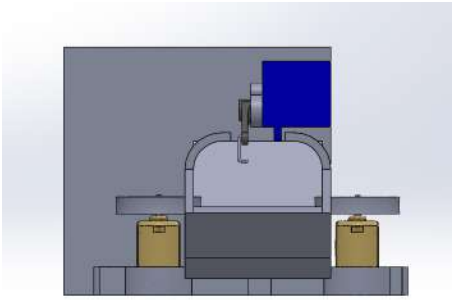
The design is an enhanced version of part 4.2.1, where we create a storage system to load the balls. The storage system is box-shaped, and a hatch allows us to put the balls in the box. The balls would then follow two adjacent slopes to move into the flywheels' entrance eventually. A servo motor blocks the entrance to release the balls when there is a command. The flywheels consist of two motors with wheels that spin in different directions so the ball would fly out when in contact with the wheels. Upon this, a sloppy track with an angle of 46.7 degrees allows the ball to move up and reach the bucket. Moreover, we plan to place this flywheel system on top of our turtlebot and put the LiDAR on top so it would not be blocked. Illustrations of the system are elaborated below.

Payload System:

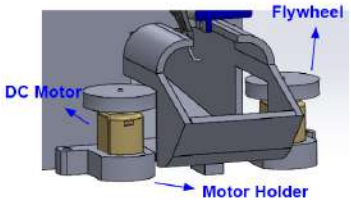


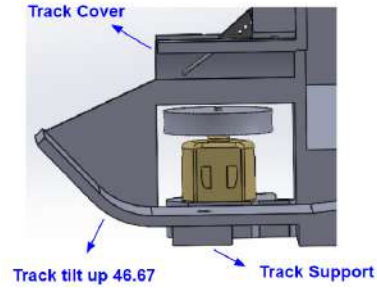






Shooting Mechanism and Calculations:





$\frac{1}{2}mv_y^2 = mgh,\ v_y = \sqrt{2gh} = \sqrt{2(9.81)(40-13)} = 2.3ms^{-1}$

$S = v_0 t + \frac{1}{2}at^2,\ 0.27 = \sqrt{2 \times 9.81 \times 0.27} + \frac{1}{2}(-9.81)t^2,\ t = 0.23s$

$v_x = \frac{x}{t} = \frac{0.5}{0.22} = 2.17ms^{-1}$

Ball in the bucket = $S_y = 25cm$

$0.22 = 2.426t + \frac{1}{2}(-9.81)t^2$

$t = 0.137s,\ t = 0.336s$

$S_x = 0.336 \times 2.17 = 0.729m$

$0.729 - 0.5 = 0.229$, so ball will go in the bucket (assuming no drag)

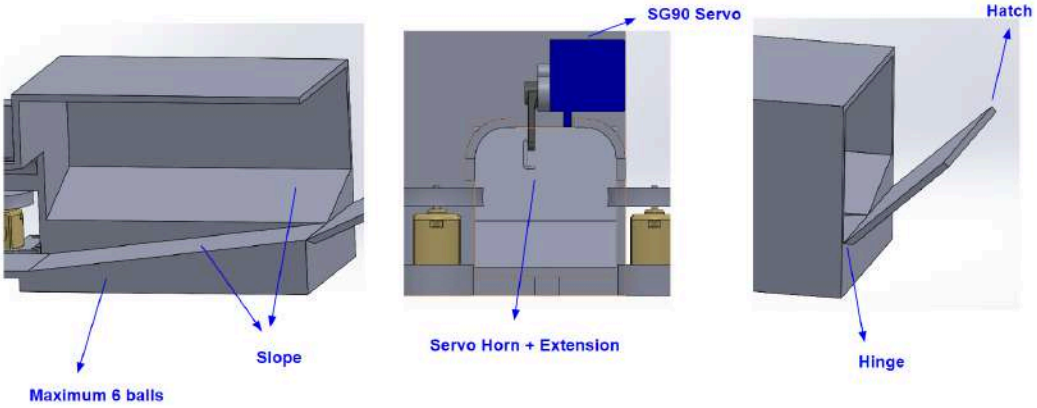
$\theta = tan^{-1}\frac{v_y}{v_x} = 46.67^o$

$v_T = 3.16ms^{-1}$, each wheel $= 1.58ms^{-1}$

$w = \frac{v}{r} = \frac{1.58}{0.015} = 98.75s^{-1}$

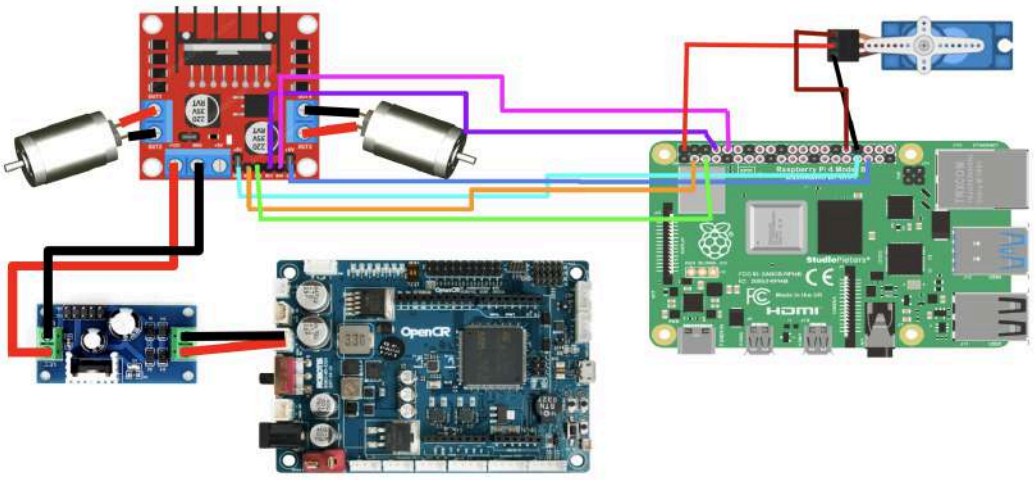
Motor RPM needed = $98.75 \times 60 \times \frac{100}{80} = 7406RPM$ (Assuming 80% efficiency)

Storing Mechanism:



Electrical Component Connection:

Here, we need an L298N motor driver to control the speed of the motors. As we plan to use a 9V DC motor capable of spinning at a rate of 7406 RPM, we add a linear 9V DC regulator



(LM7809) to step down 12V power from OpenCR to 9V.

5.4. Phase 4 - Complete Map Coverage

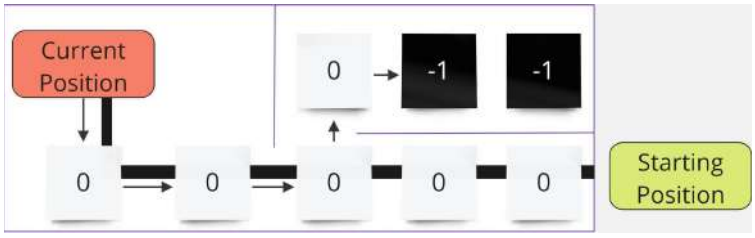
Objective of this phase: Ensure full coverage of the designated mapping area by navigating the maze autonomously.

Description of operation: Upon reaching the third T-Junction, the robot now switches to the autonomous navigation algorithm. A remote computer will subscribe to essential topics such as 'odom', 'scan', and 'map' to generate a comprehensive maze map, which persists throughout the mission. Upon receiving the 'done' signal, the remote computer triggers an autonomous navigation algorithm, which applies the concept of frontier-based exploration, where the TurtleBot will navigate towards unexplored areas within the maze. As it reaches new coordinates, the process iterates, systematically revealing and documenting the maze until full coverage is achieved.

Pseudocode: The subscription of map topics allows us to create a 2D array with OccupancyGrid data. Then, with it, we will use an algorithm similar to flood-fill to search for surrounding -1 bins.

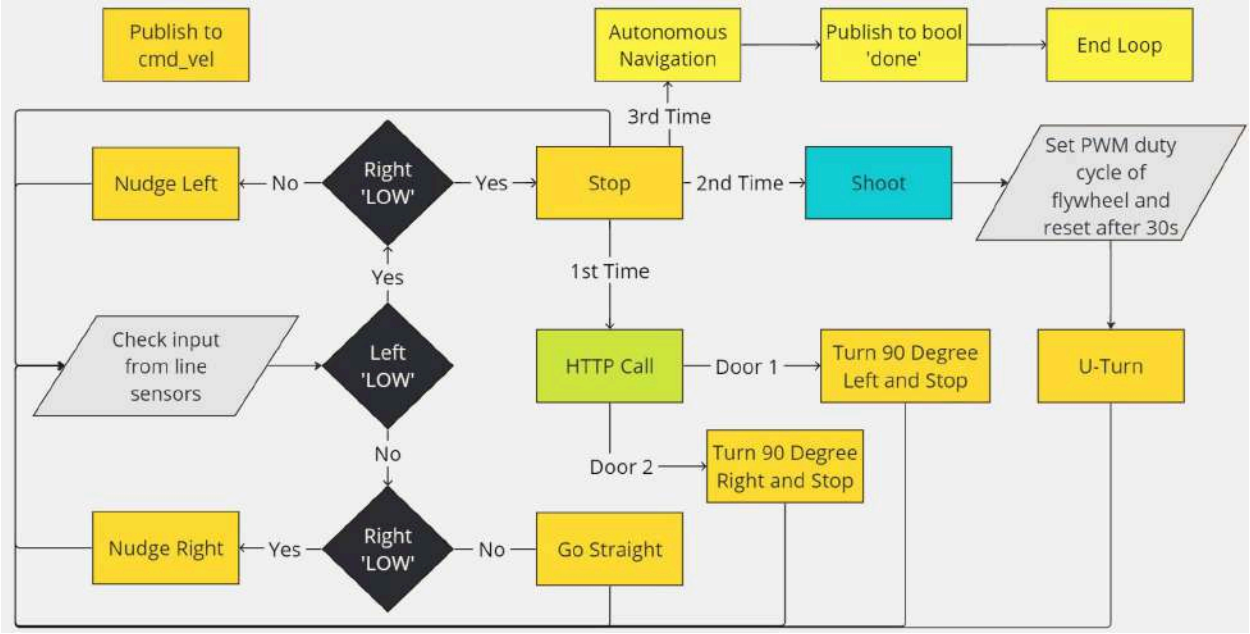
- If it is a 100 bin, it indicates that the path is obstructed so it will return.
- If it is a 0 bin, the code will recurse to its surrounding bins.
- If it is a -1 bin, it will return the coordinates for A* search algorithm to plan a path towards it.

The computer will then publish to cmd_vel the respective linear and angular velocity needed for the turtlebot to follow that path. Once it reaches that coordinate, the loop repeats to explore further unexplored areas. The diagram below shows a simple illustration of how the Turtlebot should navigate to -1 bins.



This two-tiered approach ensures efficient utilization of computational resources and seamless execution of complex tasks in navigating and mapping the maze environment.

5.5 Logic Diagram



Line following will be the main navigation tool at the start to complete the mission in the shortest time possible. Being computationally light, the line following script will be run entirely by the R-Pi to minimize potential delays associated with frequent data exchanges in the ROS network that were encountered numerous times in our testing. The R-Pi continuously monitors input from the IR sensors, detecting whether it's on the line (resulting in a 'low' input) or off (resulting in a 'high' input). Utilizing this information, the R-Pi publishes adjustments to the turtlebot's angular and linear velocities via the 'cmd_vel' topic, ensuring precise navigation along the line.

Additionally, a counter variable tracks the occurrence of T-junctions, with each encounter prompting specific actions. The first encounter will initiate an HTTP call to a server, and turn left (door 1) or right (door 2) depending on the server reply. The next encounter will activate the servo, dropping the ping pong balls into the pail, and then it will perform a U-Turn. The last encounter implies the end of the mission, so it will publish 'true' to the 'done' boolean topic, signaling the start of autonomous navigation, which will be run on a remote computer due to its high computation power requirements.

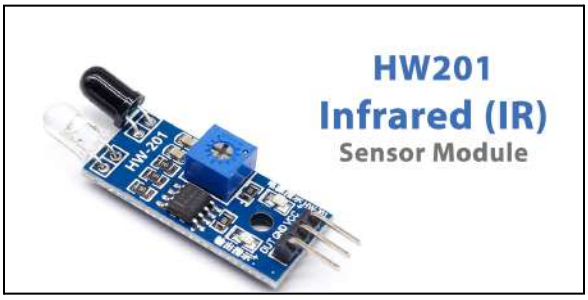
6. Prototyping and Testing

6.1 Navigation

6.1.1 Line Sensor

Due to the simplicity of line sensors, we achieved a working setup in only two weeks after the preliminary design review. However, after extensive testing, we identified a few faults and unforeseen circumstances using line sensors.

For example, the line sensors we initially used were the HW201 IR Sensor Module. The critical flaw with this sensor is that the two bulbs are easily displaced, leading to varying and unreliable readings, which can sabotage our mission success. We then changed to the TCRT5000 IR sensor to mitigate this issue, as the bulbs' positions are fixed. Furthermore, it can be screwed straight into the TurtleBot waffle plate; a custom holder is not required.



6.1.2 Autonomous Navigation

Initially, we tested the setup and algorithm as planned in our design review, but we faced various constraints that we could not fix. The plan involves using existing libraries that we can integrate into our software. However, due to its complexities and lack of online information, we cannot modify or redesign the software to fix our unique problems. For example, the TurtleBot will identify wall gaps as openings that it should pass through to explore the unexplored area behind it. Hence, the TurtleBot will either crash into the wall or get stuck in place, causing the mapping mission to fail. Even after numerous calibrations and attempts, we cannot pass this phase reliably and consistently. Other groups also encountered this problem, so my group collectively decided to change our plan to use line sensors to navigate throughout the mission.

6.1.3 Mapping with Line Sensors

To traverse through the maze with only tapes, we need to have U-shaped tapes so that the robot can enter and exit a dead-end to achieve full map closure. This implies a need to change our setup.

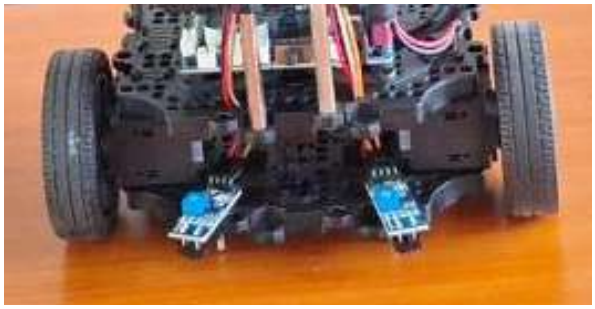
First, the U-shaped tape must be wider than the distance between the two IR sensors to prevent interference with the current tape the TurtleBot is following.

Second, the IR sensors must be close to each other so that the line is not too wide and might cause the TurtleBot to crash into the wall.

Third, the tape must be thin and flexible so that it is still narrower than the IR sensors and can form curved shapes without deforming.

Before:

After:



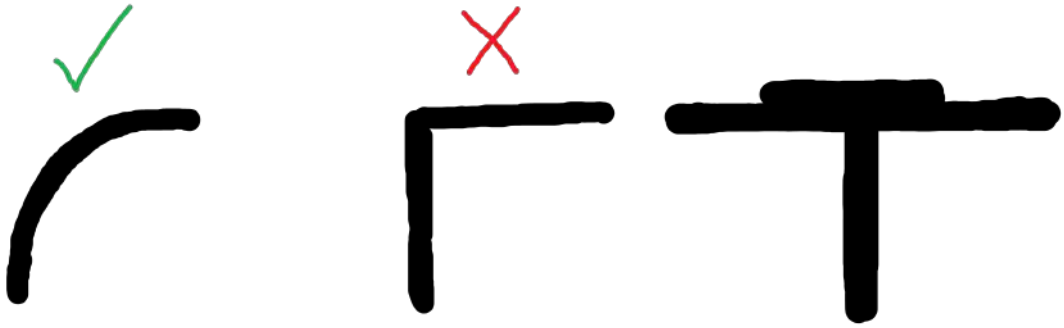
We extensively tested and calibrated this setup till it was able to work reliably. However, due to the last-minute ban on using line following to map the entire maze, we had to revert our setup to prioritize the shortest path to the lift lobby, so as to maximize our mission score. Hence, we regrettably had to give up on mapping as we were unable to solve the issues faced earlier with autonomous navigation (6.1.2) in one week.

6.1.4 Tape Design

Our group decided to continue to use thin black insulating tape to navigate as it decreases the occurrence of the TurtleBot needing to nudge, which leads to faster overall speed.

During the testing, there are occasions when the TurtleBot cannot nudge in time in sharp turns due to its high linear speed. To not compromise on the mission speed, we used curve lines for turning, which also improved the reliability of making accurate turns.

Due to similar reasons, there are occasions where the TurtleBot misses the T-Junction, as it will either continue forward or nudge right into a turn. To mitigate this, we simply add an additional layer of black tape on T-junctions to ensure that both IR sensors will definitely be on the line for it to stop reliably.



Lastly, the T-junction before the pail should be tangential to the bucket with a gap distance of approximately 3 cm between the bucket and the top layer of the T-junction. This ensures that it is close enough to drop the ball reliably (Changing to drop will be explained in 6.3.2), but far enough for it to not crash into the pail before detecting stop.

6.2 ESP32

6.2.1 Problems

During the development of the HTTP call algorithm, we faced a hardware-related challenge with the ESP32. The issue was in transitioning the ESP32 from normal operational mode to its coding mode. Initially, we found that the ESP32 required grounding the IO0 pin and toggling the EN (enable) pin to reset the board properly. This was a cumbersome step that needed to be refined for seamless operation.

Another significant hurdle was configuring the ESP32 with a static IP address. The initial thought was that a static IP would simplify the connection process. However, this approach proved to be inflexible, as it tied the TurtleBot to a single access point or gateway, which would be impractical in varying environments or when switching between different mobile hotspots.

6.2.2 Solutions

To streamline the setup process for the ESP32, we utilized a female-to-female connector cable. One end of the cable was attached to the IO0 pin and the other end to the GND (ground) pin on the ESP32 board. This direct connection effectively grounded the IO0 pin, eliminating the need to manually press the IO0 button each time we wanted to ground it.

To address the hardware challenge, we streamlined the boot mode selection process for the ESP32.

In solving the connectivity issue, we opted for a more dynamic approach by allowing the TurtleBot to accept the server's IP address as an input. This method increased flexibility, enabling the robot to connect to various networks, as the IP address can be easily obtained from any mobile device serving as a hotspot.

6.2.3 Final Algorithm

The finalized HTTP call algorithm is a Python function that performs an HTTP POST request to an ESP32 server. Here's how it works:

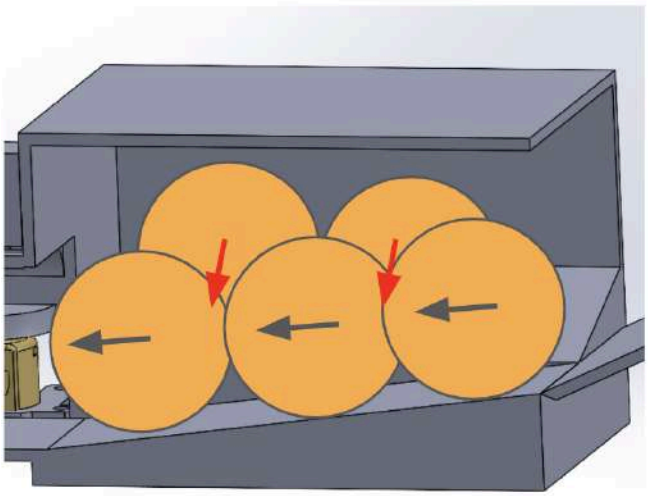
```
def httpCall(self):

    url = f'http://{ip_address}/openDoor'

    headers = {'Content-Type': 'application/json'}

    data = '{"action": "openDoor", "parameters": {"robotId": 32}}'

    door_value = ""



    try:

        response = requests.post(url, headers=headers, data=data)

        response_data = response.json()

        door_value = response_data.get('data', {}).get('message', door_value)

        print('Status Code: ', response.status_code)

        print(door_value)

        if (door_value == 'door1'):

            self.turnLeft()

        else:

            self.turnRight()

    except requests.exceptions.RequestException as e:

        print('HTTP Request failed', e)
```

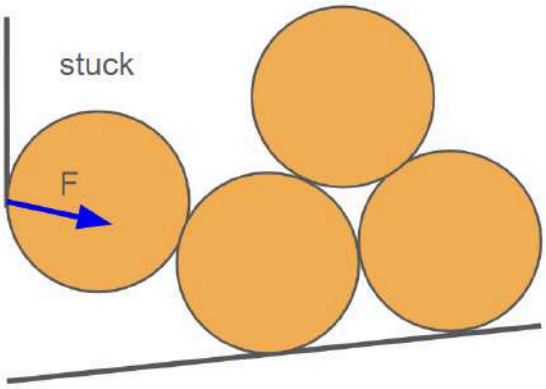
- **url = f'http://{ip_address}/openDoor'**: Constructs the URL for the HTTP request by inserting the provided IP address into the string.

- **headers = {'Content-Type': 'application/json'}:** Sets up the header for the HTTP request, indicating that the data being sent is in JSON format.
- **data = '{"action": "openDoor", "parameters": {"robotId": "32"}}':** Defines the JSON data payload for the POST request, which contains the action to be taken ('openDoor') and the ID of the robot ('32').
- **response = requests.post(url, headers=headers, data=data):** Sends the HTTP POST request to the server with the specified URL, headers, and data.
- **door_value = response_data.get('data', {}).get('message', door_value):** Extracts the 'message' field from the JSON response, which contains the door operation result.
- **if (door_value == 'door1'):::** Checks if the server's response is to open 'door1' and, if so, calls the turnLeft() method.
- **except requests.exceptions.RequestException as e::** Catches any exceptions that occur during the request, such as network errors.

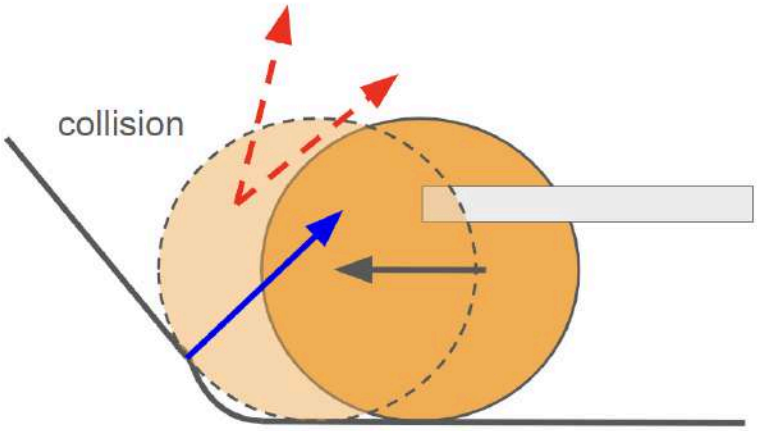
6.3 Flywheel System

6.3.1 Problems with Flywheel System





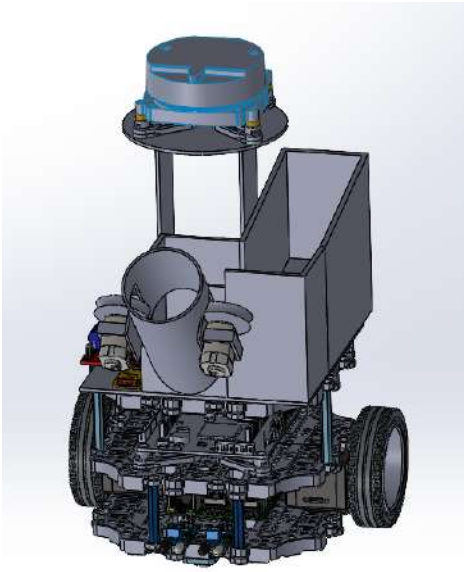
When we implemented this design, there were some existing problems. First, as we only had two slopes to let the balls fall, the balls on the top track attempted to precede the balls on the bottom track, which were supposed to move to the flywheel's entrance first. As the top balls fall, they might get stuck, as there is a reaction force from the wall that stops the ball from moving.



Another issue that we faced was that in some cases, one of the balls bounced as there was a sloppy track. Collision might still happen even if we already made a curvy track to connect the two slopes.

Furthermore, it is difficult to get the accurate projectile trajectory for every ping pong ball. This is because the two motors' speed decreases every time a ball is launched as there is a sudden change in load torque. This also needs to include friction and drag for consideration.

6.3.2 Design Changes



After modifying the design, five ping pong balls are able to be stored inside the storage and are able to feed by themselves without getting stuck. Motors are tilted to the angle required for the projectile instead of having a curvy track to guide the balls, this modification increases the efficiency and accuracy of the flywheel launcher.

The inaccuracy and inconsistency of the flywheel launcher however still remains. The flywheels slowed down after the shoot and took a while to recover the RPM required, in addition, if two flywheels not spinning at the same speed will cause the balls to rotate to the side.

After a few prototype testing, we decided to change the design to a dropping mechanism due to the risks of the flywheel launcher stated above. The dropping mechanism ensures the accuracy of feeding five ping pong balls into the pail which can guarantee our scores.

6.4 Dropping System

The previous flywheel mechanism is replaced by a gravity-based dropping system. Positioned correctly over the bucket, the robot releases the ping pong balls. The dropping mechanism is activated by a servo motor which, when triggered, opens a trapdoor allowing the balls to fall.

6.4.1. Pipe Support

The supports were initially an L-shaped design with one short triangle (Fig 6.4.1.1), however, we found that they might wrap during the 3D printing process and were not stiff enough to hold the pipe. Therefore, the supports were redesigned to have two triangles all the way to the top (Fig 6.4.1.2).

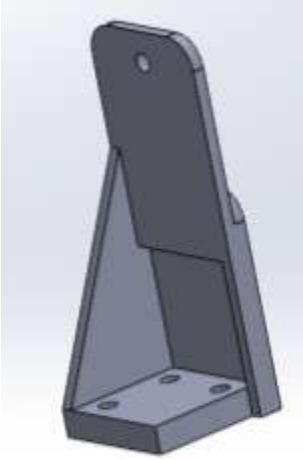


Figure 6.4.1.1

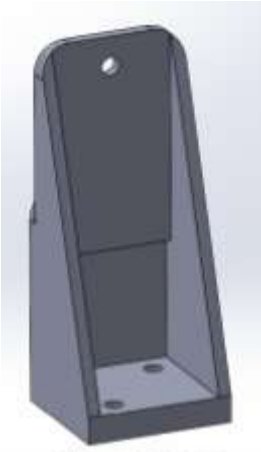


Figure 6.4.1.2

6.4.2. Servo Holder

The holder was initially too weak as the holes to screw into the pipe were too close to the edge (Fig 6.4.2.1), this causes the holder to break when compressed by bolts and nuts. Therefore, the holes were shifted inwards (Fig 6.4.2.2).

Figure 6.4.2.1

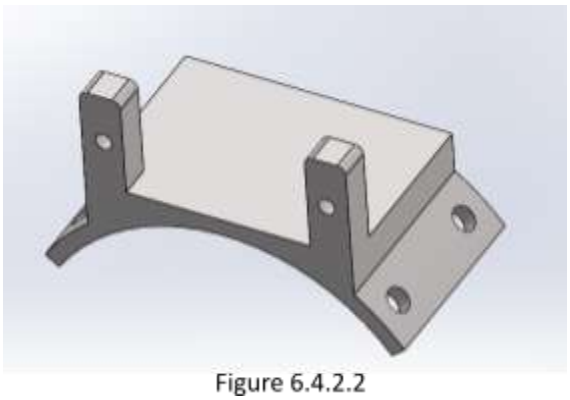


Figure 6.4.2.2

6.4.3. Ball Backward Slippage Issue

When the robot is moving, there is a risk that the ping pong ball will slip out from the back of the pipe, therefore the back of the pipe needs to be closed. To make sure the stopper is rigid, we added two bolts and nuts at the end of the pipe (Fig 6.4.3.1).

Figure 6.4.3.1

6.4.4. High Center of Gravity (CG) Issue

Due to the mass and height of the PVC pipe, our robot has a higher CG around 23cm from the ground. This caused our robot to be unbalanced, it shakes when moving and easily loses three points of contact with the ground. To solve this problem, we tried to add weight at the bottom front or back of the robot to minimize the vibration (Fig 6.4.4.1). Although it did work it was not enough, it needed more weight to make the CG lower. Therefore, we came up with the idea of a mass damper to counter the momentum when the robot is moving (Fig 6.4.4.2). It worked perfectly, the robot is now more stable and the faster it goes the more stable it is. The damper is then enhanced by using wires (Fig 6.4.4.3).

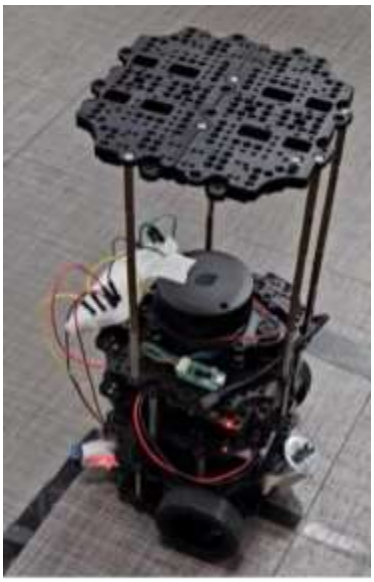


Figure 6.4.4.1

Figure 6.4.4.2

Figure 6.4.4.3

7. Final Design

7.1 Key Hardware Specifications

	List	Specifications	Notes
Turtlebot	Dimensions (mm) (LxWxH)	Dispenser: 220.21 x 175.52 x 209.09 Turtlebot3 & Dispenser: 220.21 x 175.52 x 421.46	
	Weight (kg)	Dispenser: 0.728kg Turtlebot: 1kg Total weight:1.728 kg	
	Wheel Base (mm)	80.66	From center of DYNAMIXEL to the center of ball caster
	IR Sensor	2x TCRT5000	
	Servo Motor	1x SG90	
	Damper Weight (kg)	0.2	
System	Battery Capacity	LiPo Battery 11.1V 1,800mAh	
	Expected Operating Time	30 minutes	
	Communication Interface	GPIO, PWM	

7.2 Bill of Materials

The bot was built using a TurtleBot3 Burger Set, from ROBOTIS. The ball dispenser however was built by the team. We had an allocated budget of S$100 which we used to buy a few additional components and to manufacture any specific parts that we had. Below is the bill of materials which included components we used.

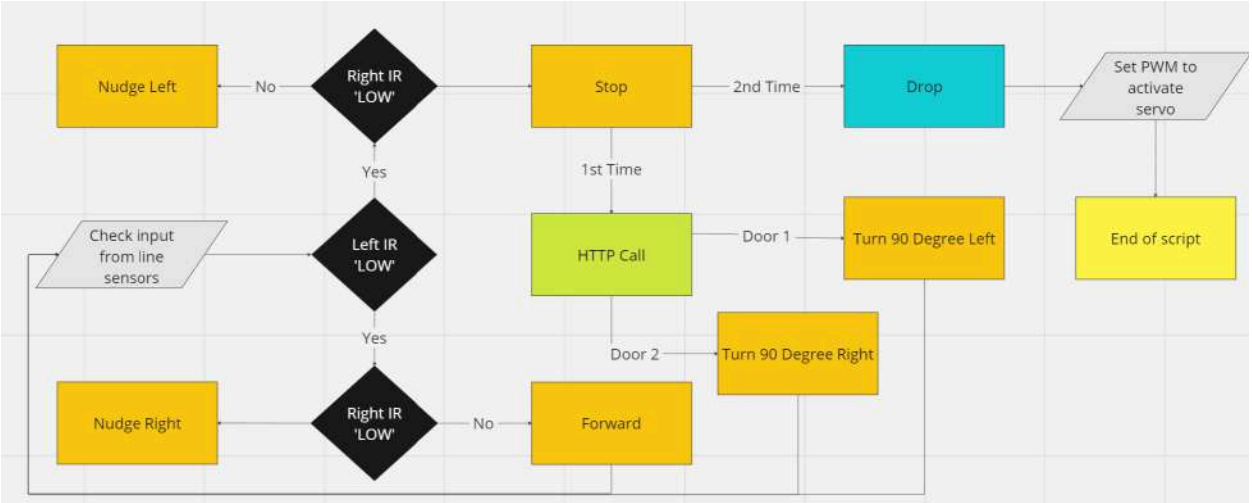
Item	Quantity	Unit Cost	Total Cost
First Layer			
SG90	1	Provided	Provided
Servo Horn	1	10 min 3D print	Self Sourced
Servo Support	1	1 hour 3D print	Self Sourced
PVC Pipe	1	12.68	12.68
Pipe Support (Long)	2		
Pipe Support (Short)	2	3 hour 50 min 3D print	Self Sourced
M2x15 for Servo	2	Sourced from Lab	Sourced from Lab
M3x8 for Servo Holder	4	Sourced from Lab	Sourced from Lab
M3x15 for Pipe Ball Stopper	2	Sourced from Lab	Sourced from Lab
M3x10 for Pipe to Pipe Holder	4	Sourced from Lab	Sourced from Lab
M3x10 for Pipe Holder to Waffle	12	Sourced from Lab	Sourced from Lab
M2 Nut	2	Sourced from Lab	Sourced from Lab
M3 Nut	24	Sourced from Lab	Sourced from Lab
M3x8 for Waffle Plates	4	Sourced from Lab	Sourced from Lab
Second Layer			
M3x10 for M3 Support	8	Sourced from Lab	Sourced from Lab

M3x35 Spacer	8	Sourced from Lab	Sourced from Lab
M3x35 Support	4	Sourced from Lab	Sourced from Lab
M3x8 for Waffle Plates	4	Sourced from Lab	Sourced from Lab
M3x10 for Spiderwire	4	Sourced from Lab	Sourced from Lab
M3 Hex Nut	8	Sourced from Lab	Sourced from Lab
200-gram Weight Calibration (Mass Damper)	1	Self Sourced	Self Sourced
Spiderwire & Rubberband	1	Self Sourced	Self Sourced
Third Layer			
M3x10 for M3 Support	8	Provided	Provided
M3X25 Spacer	4	Sourced from Lab	Sourced from Lab
M3X45 Support	4	Provided	Provided
M2.5x8 from LiDAR to LiDAR Support	4	Provided	Provided
White small tube from LiDAR to LiDAR Support	4	Provided	Provided
M2.5 Hex Nuts for LiDAR Support	8	Provided	Provided
M2.5x8 for LiDAR Support	4	Provided	Provided
LiDAR	1	Provided	Provided
M3x8 for Waffle Plates	4	Provided	Provided
M3 Hex Nut	4	Provided	Provided
Fourth Layer			
M3x10 for Support	12	Provided	Provided
M3X45 Support	6	Provided	Provided
USB to LDS Cable	2	Provided	Provided

USB2LDS	1	Provided	Provided
RPi	1	Provided	Provided
M2.5x10 for RPi to RPi Support	4	Provided	Provided
M2.5 Hex Nut for RPi Support	8	Provided	Provided
M2.5x8 for RPi Support	4	Provided	Provided
3X3 Waffle Support for USB2LDS	1	Provided	Provided
Connectors for USB2LDS	2	Provided	Provided
RPi Support	4	Provided	Provided
M3x8 for Waffle Plates	4	Provided	Provided
M3 Hex Nut	4	Provided	Provided
Fifth Layer			
OpenCR1.0	1	Provided	Provided
OpenCR Support	4	Provided	Provided
Battery to Power Cable	1	Provided	Provided
5V to RPi Cable	1	Provided	Provided
M3x10 for M3 Support	8	Provided	Provided
M3X45 Support	4	Provided	Provided
M2.5x8 for OpenCR to OpenCR Support	4	Provided	Provided
M2.5x8 for OpenCR Support	4	Provided	Provided
M2.5 Hex Nut	4	Provided	Provided
M3x8 for Waffle Plates	4	Provided	Provided
M3 Hex Nut	4	Provided	Provided

Sixth Layer			
M3X35 Support	4	Provided	Provided
M3x10 for M3 Support	4	Provided	Provided
M2.5x8 for Dynamixel Motor	8	Provided	Provided
Jumper Wires	6	Sourced from Lab	Sourced from Lab
L Shaped Bracket with 2 Screws for Tightening	6	Provided	Provided
M3x15 for IR Sensor Mount	2	Sourced from Lab	Sourced from Lab
IR Sensor TCRT5000	2	0.73	1.46
Ball Caster	2	Provided	Provided
M1x15 for Ball Caster	4	Provided	Provided
M3x8 for Waffle Plates	4	Provided	Provided
M3 Hex Nut	6	Sourced from Lab	Sourced from Lab
Markers			
Black Tape MP002141 19 mm x 8 m	15	1.04	15.6
Miscellaneous			
ESP32	1	Provided	Provided
Total Cost			29.74

7.3 Final Logic Diagram

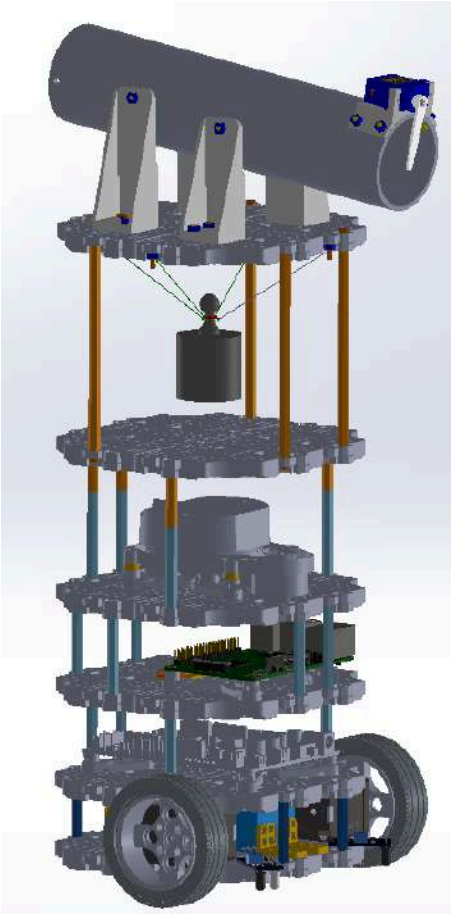


Due to the various changes in our mission plan and design, mainly autonomous navigation (6.1.2) and the flywheel system (6.3.1), the software logic of our turtlebot changed. The main difference is that in the drop function, the PWM will only be set for approximately 2s to move the servo stick aside and allow the balls to drop. Afterwards, the script will end and autonomous navigation will not be used.

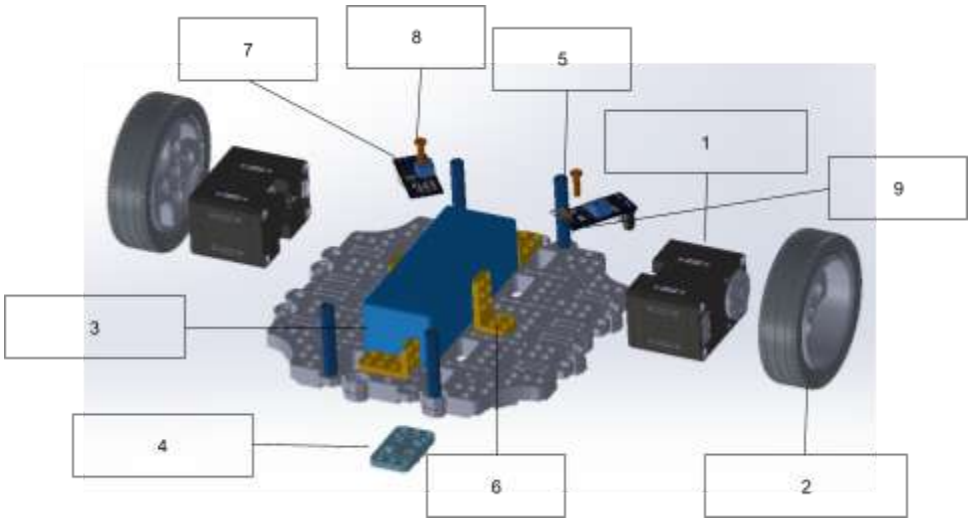
8. Assembly Instructions

8.1 Mechanical Assembly

8.1.1. Overall Turtlebot

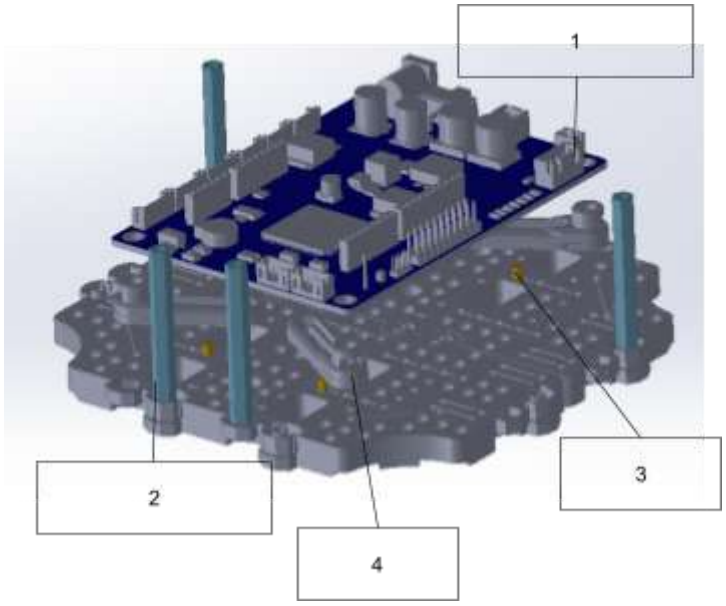


8.1.2. Sixth Layer



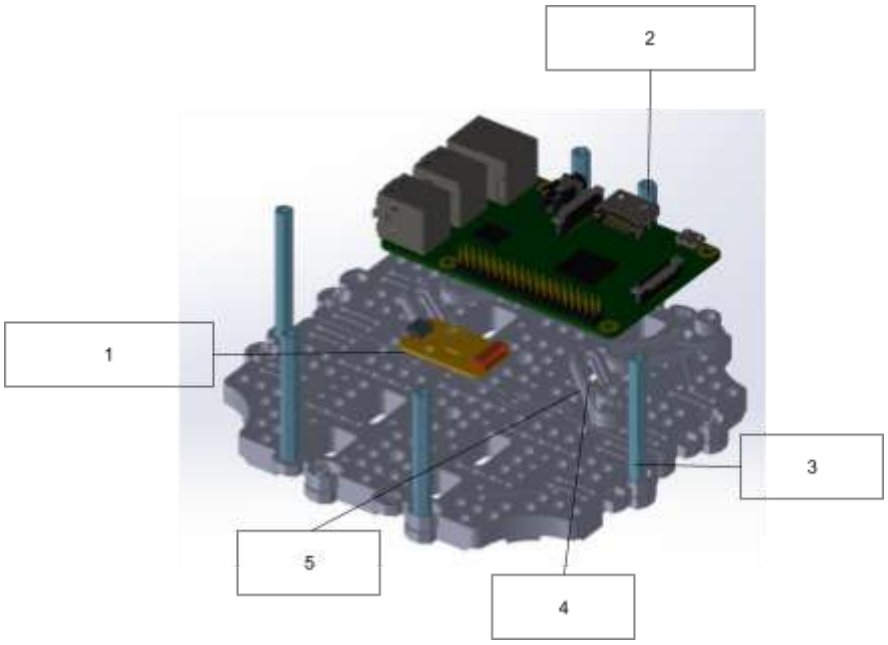
No.	Part	Qty
1	DYNAMIXEL Motor	2
2	Wheel & Tire	2
3	Li-Po Battery	1
4	Ball Caster	1
5	M3x35 Standoff	4
6	L Bracket	5
7	IR Sensor	2
8	M3x15	2
9	M3 Hex Nut	2

8.1.3. Fifth Layer



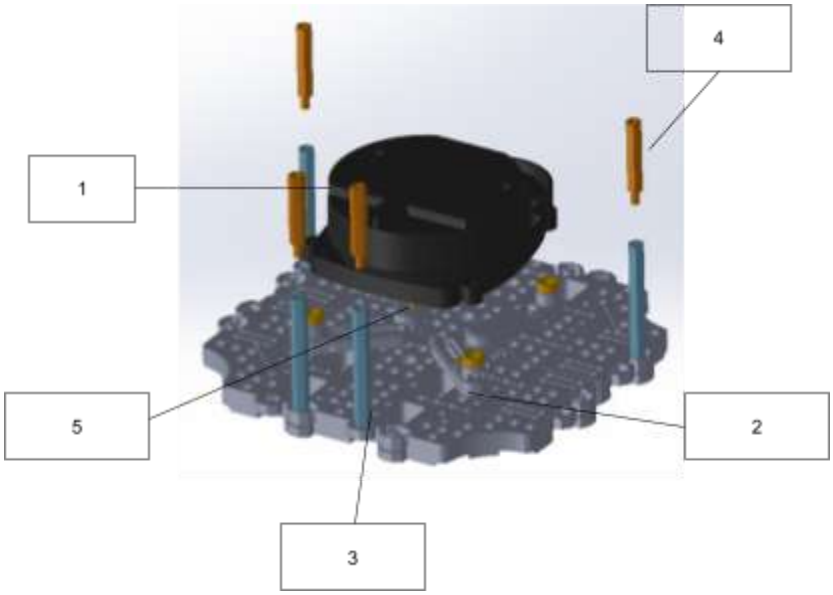
No.	Part	Qty
1	OpenCR 1.0	1
2	M3x45 Standoffs	4
3	PH_2.5x8_K	4
4	OpenCR Support	4

8.1.4. Fourth Layer



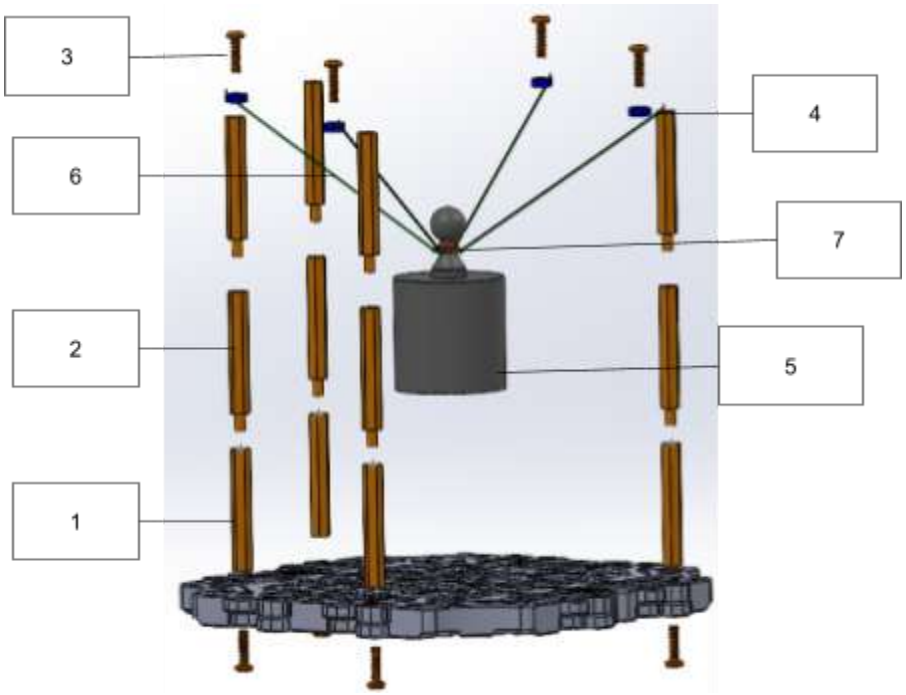
No.	Part	Qty
1	USB2LDS	1
2	Raspberry Pi	1
3	M3x45 Standoffs	6
4	PH_2.5x8_K	4
5	RPI Support	4

8.1.5. Third Layer



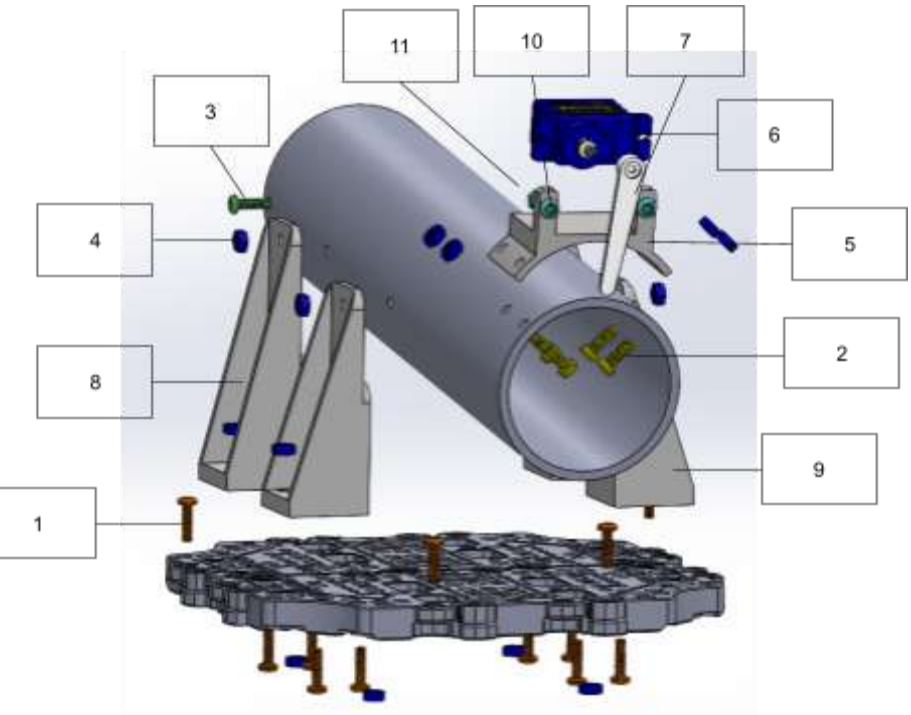
No.	Part	Qty
1	LIDAR	1
2	LIDAR Support	4
3	M3x45 Standoffs	4
4	M3x25 Spacer	4
5	PH_2.5x8_K	4

8.1.6 Second Layer



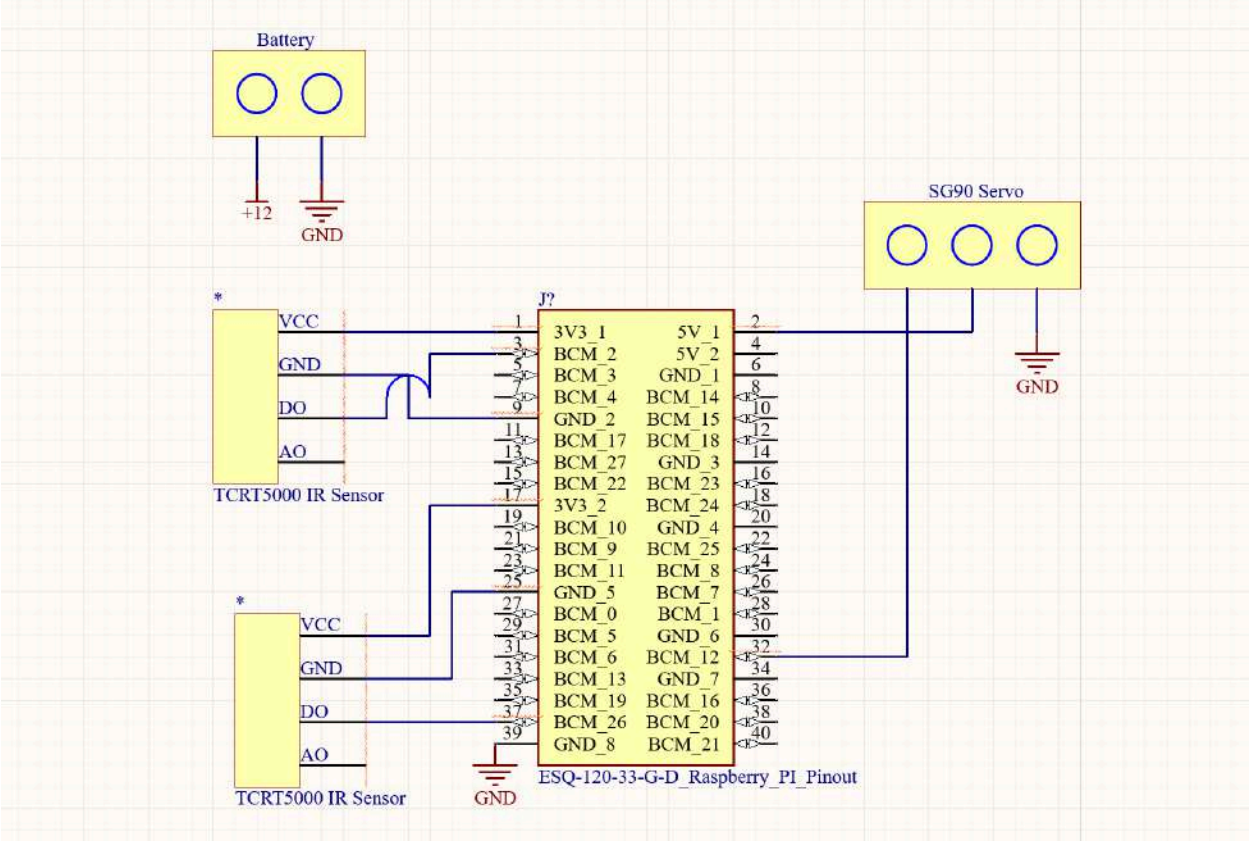
No.	Part	Qty
1	M3x35 Support	4
2	M3x35 Spacer	8
3	M3x10	8
4	M3 Hex Nut	4
5	200-gram Weight Calibration	1
6	Spiderwire	1
7	Rubberband	1

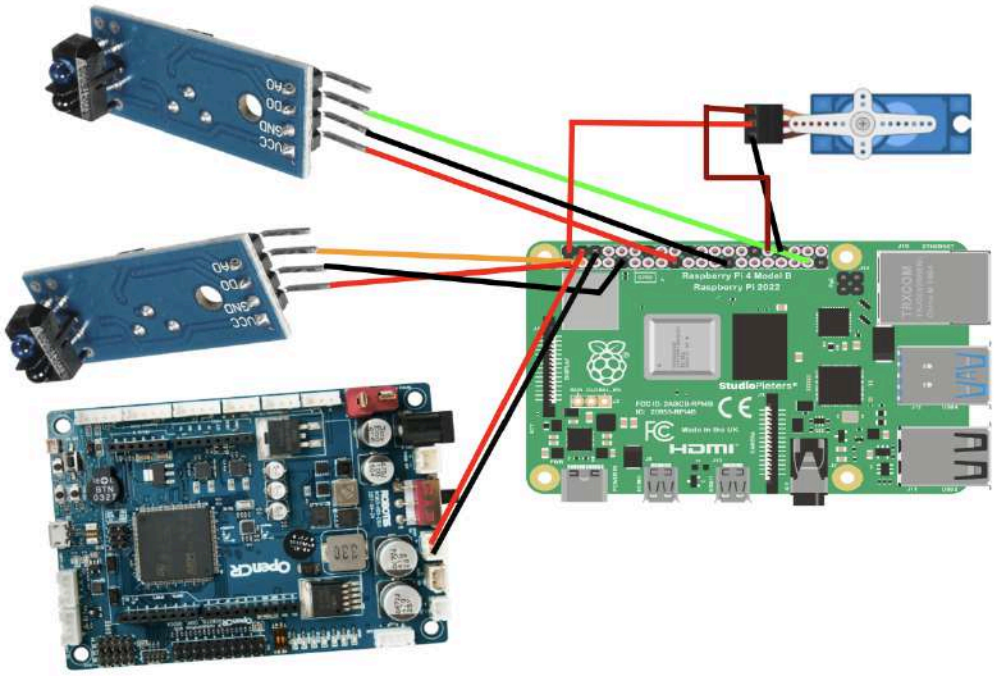
8.1.7. First Layer



No.	Part	Qty
1	M3x10	18
2	M3x8	4
3	M3x15	2
4	M3 Hex Nut	24
5	Servo Holder	1
6	SG90	1
7	Servo Horn	1
8	Pipe Support (Long)	2
9	Pipe Support (Short)	2
10	M2x15	2
11	M2 Hex Nut	2

8.2: Electrical Assembly





8.3 Software Assembly

8.3.1 Raspberry-Pi Setup

1. Burn ROS2 Foxy image to a SD card using this guide.
2. Access the RPi with its IP address.
    - ssh ubuntu@{IP_ADDRESS_OF_RASPBERRY_PI}
3. To ease future operation, include the following lines in the ~/.bashrc file (nano ~/.bashrc)
    - export ROS_DOMAIN_ID={NUMBER}
    - export TURTLEBOT3_MODEL=burger
    - alias rosbu='cd colcon_ws/src && ros2 launch turtlebot3_bringup robot.launch.py'
    - alias run='cd colcon_ws/src && ros2 run auto_nav final.py'
4. Run the following command to ensure that the ROS environment is working. The TurtleBot should be able to move.
    - On one terminal: rosbu
    - On the another terminal: rteleop
5. Create a ROS2 package.
    - cd ~/colcon_ws/src
    - ros2 pkg create --build-type ament_python auto_nav
    - cd auto_nav/auto_nav
    - mv __init__.py ..
2. Clone the Github Repository.
    - git clone https://github.com/cyanhere/r2auto_nav.git.
    - mv ../__init__.py .
3. Build the package
    - cd ~/colcon_ws && colcon build
4. Edit the package.xml to include dependencies
    - cd auto_nav

- nano package.xml
- <export>

    <build_type>ament_python</build_type>

    <exec_depend>rclpy</exec_depend>

    <exec_depend>std_msgs</exec_depend>

    <exec_depend>time</exec_depend>

    <exec_depend>requests</exec_depend>

  </export>

5. Edit setup.py to run scripts
    - nano setup.py
    - entry_points={

        'console_scripts': [

            'final = auto_nav.final:main',

        ],

      },

6. To run the script
    - On one terminal: run
    - On another terminal: rosbu

8.4 Software Explanation

8.4.1 Library and Constants

- The maximum linear speed (0.21) that can be published to cmd_vel is used to ensure TurtleBot can complete the mission in the fastest time possible.
- nudgeAngle value is relatively high so that it can turn swiftly.
- nudgeSpeed is low to minimize the chances of it not being able to turn in time.
- turnSpeed, turnAngle and turnTime are calibrated such that TurtleBot can execute a 90 degree turn accurately.

```
# Library

import rclpy

from rclpy.node import Node

import RPi.GPIO as GPIO

from geometry_msgs.msg import Twist

import time

import requests



# constants

maxSpeed = 0.21 # forward linear

reducedSpeed = 0.01 # nudge linear

nudgeAngle = 2.3 # nudge angular

turnSpeed = 0.05 # turn linear

turnAngle = 1.5 # turn angular

turnTime = 1.0 # calibrated 90 deg turn time

ip_address = "192.168.18.87" # ip-address of ESP32
```

8.4.2 Sensor setup

```
# IR setup to navigate
```

```
GPIO.setmode(GPIO.BCM)

leftIr = 26 # Left IR GPIO Pin

rightIr = 2 # Right IR GPIO Pin

GPIO.setup(leftIr, GPIO.IN)

GPIO.setup(rightIr, GPIO.IN)



# Servo setup to drop

servoPin = 12 # Servo Pin

GPIO.setup(servoPin, GPIO.OUT)

servo = GPIO.PWM(servoPin, 50)
```

8.4.3 Publisher Node

- A publisher node is created to publish linear and angular data to cmd_vel topic.

```
class Mission(Node):



    def __init__(self):

        super().__init__('mission')

        self.publisher_ = self.create_publisher(Twist,'cmd_vel',10) # Create Pub

        self.juncCount = 0; # Junction encountered
```

8.4.4 IR Navigation

The logic flow is demonstrated in the logic diagram (Refer to 7.3)

```
    def irmover(self):

        leftOnLine = GPIO.input(leftIr)

        rightOnLine = GPIO.input(rightIr)

        time.sleep(0.01)
```

```
        if leftOnLine:
            if rightOnLine:
                self.stop()
                print('stop')
            else:
                self.nudgeLeft()
                print('nudge left')
        else:
            if rightOnLine:
                self.nudgeRight()
                print('nudge right')
            else:
                self.forward()
                print('forward')
```

8.4.5 Movement

```
    def turnLeft(self):
        twist = Twist()
        twist.linear.x = turnSpeed
        twist.angular.z = turnAngle
        time.sleep(0.2)
        self.publisher_.publish(twist)
        time.sleep(turnTime)
```

```
def turnRight(self):
    twist = Twist()
    twist.linear.x = turnSpeed
    twist.angular.z = -turnAngle
    time.sleep(0.2)
    self.publisher_.publish(twist)
    time.sleep(turnTime)

def nudgeLeft(self):
    twist = Twist()
    twist.linear.x = reducedSpeed
    twist.angular.z = nudgeAngle
    self.publisher_.publish(twist)

def nudgeRight(self):
    twist = Twist()
    twist.linear.x -= reducedSpeed
    twist.angular.z = -nudgeAngle
    self.publisher_.publish(twist)

def forward(self):
    twist = Twist()
    twist.linear.x = maxSpeed
```

```
        twist.angular.z = 0.0

        self.publisher_.publish(twist)
```

8.4.6 Stop Movement

The juncCount variable will increment for every stop call. When it is one, it will perform httpCall (Refer to 6.2.3), else it will call drop (Refer to 8.4.7).

```
    def stop(self):

        twist = Twist()

        twist.linear.x = 0.0

        twist.angular.z = 0.0

        time.sleep(0.1)

        self.publisher_.publish(twist)

        self.juncCount += 1 # Junction encountered +1

        if (self.juncCount == 1):

            self.httpCall()

        elif (self.juncCount == 2):

            self.drop()
```

8.4.7 Drop Ball

Increasing the duty cycle will cause the servo stick to move and thus the ball to roll down the pipe.

```
    def drop(self):

        servo.start(0)

        servo.ChangeDutyCycle(12.5) # Extend servo

        time.sleep(3)

        servo.ChangeDutyCycle(2.5) # Retract servo

        time.sleep(1)
```

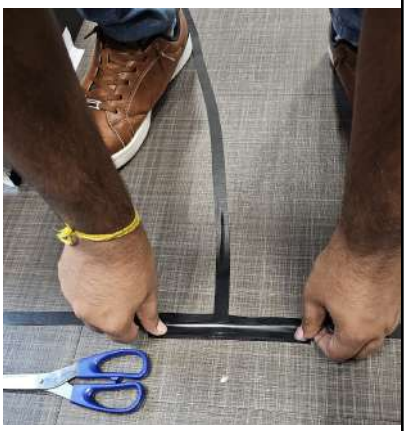
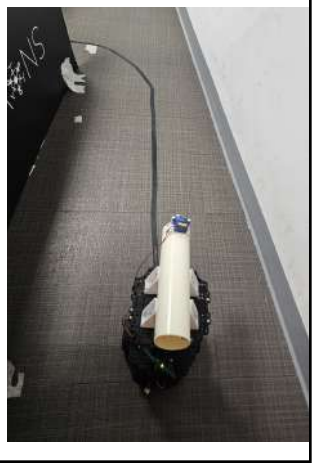
8.4.8 Main Function

It will run till juncCount is 2, implying the end of the mission.

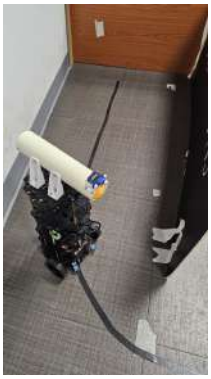
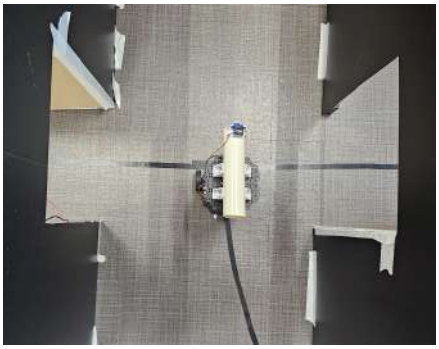
```
    def mover(self):
        try:
            while True:
                if self.juncCount == 2:
                    break
                self.irmover()
        except Exception as e:
            print(e)
        finally: # Ctrl-c detected
            self.stop() # stop moving
            servo.stop()
            GPIO.cleanup()


def main(args=None):
    rclpy.init(args=args)
    mission = Mission()
    mission.mover()
    mission.destroy_node()
    rclpy.shutdown()
```

9. Systems Operation Manual

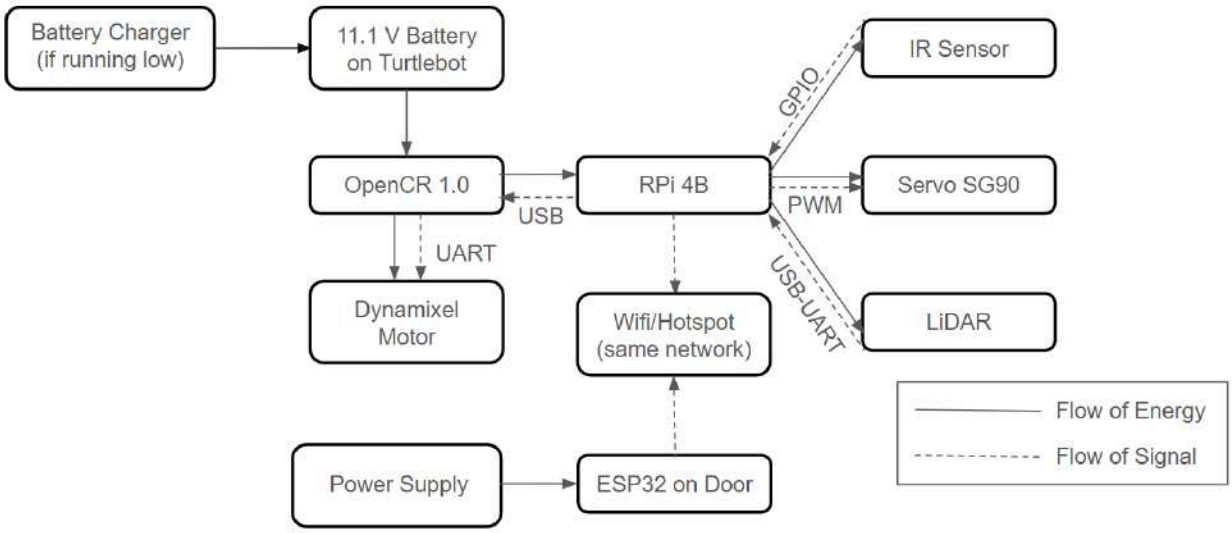
9.1 Turtlebot Positioning and Markers Positioning

<table>
<tr>
<td>1) Lay tape such that it is the shortest path to the lift lobby
  </td>
<td>2) Do double taping for every T-junction
  </td>
<td>3) Leave one finger distance from the pail's T-junction
  </td>
</tr>
<tr>
<td>4) Move servo stick aside and load all the ping pong balls
  </td>
<td>5) Return servo stick to the center
  </td>
<td>6) Place the turtlebot on the starting line and run the scripts
  </td>
</tr>
</table>

9.2 Mission Flow

1) Turtlebot will nudge towards the IR sensor that touches the tape and stop at T-junctions.	2) It will stop at the lift lobby area to initiate HTTP call and turns towards the returned door.	3) It will stop before the pail and move the servo to drop the ball
 		

9.3 Hardware Block Diagram



9.4 Factory Acceptance Test

Component	To be checked	Observation
OpenCR	Able to be powered by the LiPo Battery	Green LED lights up when connected to a power source, boot up the tune being played
RPi	Able to turn on the RPi when connected to OpenCR	The red light turns on while the green light flashes
	Can be connected from the remote laptop	Terminal returns “Welcome to Ubuntu…” when “sshrp” is run on the laptop
Lidar	Able to spin and collect data consistently	The environment will be mapped on Rviz
ESP32	Able to be powered by a wall plug adaptor	The constant red light on ESP32
SG90	Able to rotate the ball stopper to allow balls to drop	Balls drop down vertically when the stopper attached to the servo moves 180 degrees vertically.
IR Sensor	Able to detect black tape when it is close	Green light from the IR Sensor turns off if it detects black tape and turns on if it does not detect black tape
Wheels	Able to move the bot in all directions freely	Bot can be controlled properly when running ‘rteleop’
Ball caster	Able to roll in all directions freely	Bot able to move around in all directions smoothly with ball caster attached
Mass Damper	It should be tightened but able to move slightly. The strings are fastened to each of the corners.	The mass damper can move in different directions without being cut off due to the vibrations
Structural stability	Structural platforms and components installed correctly	Shake Turtlebot to verify all components are mounted securely
	Verify all fasteners installed and tightened	Verify fastener counts are consistent with the assembly document

10. Troubleshooting

10.1 Hardware and Electronics

A. Parts are loose or shaky.
    - Fasten all screws tightly

B. The battery is loose.
    - Wrap the cable around the support and put the battery back into the battery holder tightly
    - Worst case: add more L-shaped brackets to secure

C. The whole robot is wobbly due to the high center of gravity.
    - Compensate the speed for the balance of the robot
    - Worst case: Add mass damper with strings to balance the robot

D. Servo does not rotate as intended and therefore, doesn't let the ping pong balls come out.
    - Check the initial servo angle and move it to the vertical position
    - Ensure wires are connected to the correct GPIO pins

E. The line following sensor does not produce the correct output when detecting a black tape or floor.
    - Recalibrate the sensor's sensitivity by rotating the potentiometer

F. The robot may not stop properly to do tasks when detecting a T-junction.
    - Make the T-junction thicker yet not too wide to allow 90-degree turning

G. The robot hits the bucket when trying to stop and unload the balls.
    - Put a 1-finger wide gap between the T-junction and the outer side of the bucket
    - Ensure that the tape put is straight and tangential to the bucket

10.2 Software

A. The laptop cannot connect to Raspberry Pi

- Make sure that the laptop and Raspberry Pi are on the same network. Make sure not to use networks that run on VPN like NUS's WiFi as it causes some connectivity issues.
- Ensure that RPi is powered on and has started booting and outputting some sound.

B. No map received or RViz not working well
   - Restart rslam and rosbu, try rosbu first then rslam.

C. Connection Refused
   - Check the same hotspot (not the NUS_STU), restart the hotspot

D. HTTP Call not working on the first try
   - Check the IP address if it is inputted correctly.
   - Retry the call

11. Future Scope

Future Software Enhancements - Autonomous Navigation Using Frontier-Based Exploration

Objective of the Enhancement: To implement an autonomous navigation system that activates after the TurtleBot completes the task of dropping ping pong balls, enabling it to explore and map the entire maze without manual intervention.

Description of the Enhancement: Upon successfully completing the ping pong ball dropping, the TurtleBot will shift from task-oriented operations to autonomous exploration mode. This mode will leverage frontier-based exploration algorithms, where the robot will actively seek out and move toward the closest unexplored areas, marked as '-1' in the occupancy grid.

The software will utilize LIDAR data to update the occupancy grid continuously. Once a frontier, or an unexplored edge, is identified, the robot will calculate the most efficient path using the A* algorithm and proceed to explore that sector. This process will repeat, with the TurtleBot autonomously navigating through the maze, until complete map closure is achieved or the exploration is manually ended.

The anticipated outcome of this enhancement is a comprehensive and detailed mapping of the environment, achieved with no human oversight, thus optimizing the efficiency of the mapping process and the overall utility of the robot in dynamic environments.