EE2111A Group Project

Project Saturn - Automated Vacuum Cleaner

B03 Group 07

FANG TIANCHI	A0272402R
DU YANZHANG	A0277012M
CHAO JINCHENG	A0276289J
CHAO JINCHENG	A0276944M
WU YANZI	A0286443X
DAVID MICHAEL INDRAPUTRA	A0276057X
DUAN YIHE	A0276944M

Contents

1. INTRODUCTION

1.1 RATIONALE

This project aims to design a robotic vacuum cleaner that can vacuum an area of 1000 sq ft. As the area can be approximately evaluated as the area of a 3-bedroom HDB flat, the cleaner is applied to do the cleansing task within the given area. At the end of the process, the cleaner can go back to the starting point to be charged so that it can be further utilised. Also, there is a limitation alarm if the dust exceeds a certain amount to make the cleansing session more effective.

This report contains details of a vacuum cleaner prototype and a higher level design of it. The objectives of the prototype of our design are to clean all the possible areas within a 50 cm x 50 cm square with black tape as the border, to avoid obstacles within the parameter, and to return to the starting position, which serves as a docking place. The prototype consists of an mBot and a soldered vero board containing the IR receiver, force-resistive sensor, and analog-to-digital converter. There is an IR transmitter that is positioned in the docking place such that the robot will stop moving when detecting the signal. The force sensor will measure the weight of the object that is put in it, and if it exceeds a threshold value, a buzzer will turn on and an LED will light up. Besides, a higher level design of our prototype consists of the functional block diagram and the mechanical design of the robot, as well as the suction or cleansing mechanism, a pseudocode or the logic flow diagram of the algorithm of the real robot, the circuit schematics and the bills of material of the robot, and the sizing of the battery used for power.

Our higher level design was inspired by Mrs Goh, a working mother with two children and a pet, making her house the place for their daily activities. However, with her compact schedule, cleaning the room becomes challenging work for her. To deal with this situation, she started to seek a cleaning robot that can fit her needs. Firstly, as there are several rooms with different layouts, the robot should be able to scan and restore each environment. Also, it needs to have the function to detect and avoid objects, as there might be toys scattered on the floor. Secondly, as the required cleaning part varies each day, the function of path planning is also a must. Thirdly, she is not able to focus the remaining battery and charge it consistently, the robot needs to move to the charging station and do wireless charging on its own. Lastly, as the waste produced by children and pets consist of various types, including dusts and liquids, the suction and filtering functions are required for more efficient and thorough cleaning. The cleaning process of the dust bag would be easier also. Therefore, we designed this robotic vacuum cleaner that can fulfill the requirements of Mrs Goh and other working adults to improve their efficiency in household chores.

1.2 PROJECT TIMELINE

Week 9:

- Finalise document content
- Familiarise with demonstration tasks
- Research navigation algorithm and useful components for prototype
- Draw prototype functional block diagram

Week 10:

- Finalise prototype electrical schematics
- Finalise prototype algorithm regarding covering most areas, obstacle avoidance, docking process and using Force Sensor Board to turn on in-built LED
- Update most recent code on Github for version control

Week 11:

- Soldering Force Sensor Board circuit on one Vero board
- Soldering Infrared receiver circuit on one Vero board
- Iterate prototype in the Lab and update codes based on prototype performance
- Brainstorm higher-level design functions
- Draw higher level functional block diagram

Week 12:

- Keep iterating on the prototype and adjust codes
- Finalise higher level functional block diagram
- Prepare for presentation slide
- Delegate manpower for different tasks on the presentation date

Week 13:

- Conduct presentation and prototype demonstration in the Lab
- Finish algorithm and pseudocode for higher level design
- Finish sourcing components and filling in Bill of Material
- Finish electrical connection of higher level design
- Consider the advantages and drawbacks of the higher level design

Week 14:

- Final check for content
- Formatting the report and submit

2. PROTOTYPE

2.1 PROTOTYPE FUNCTIONAL BLOCK DIAGRAM

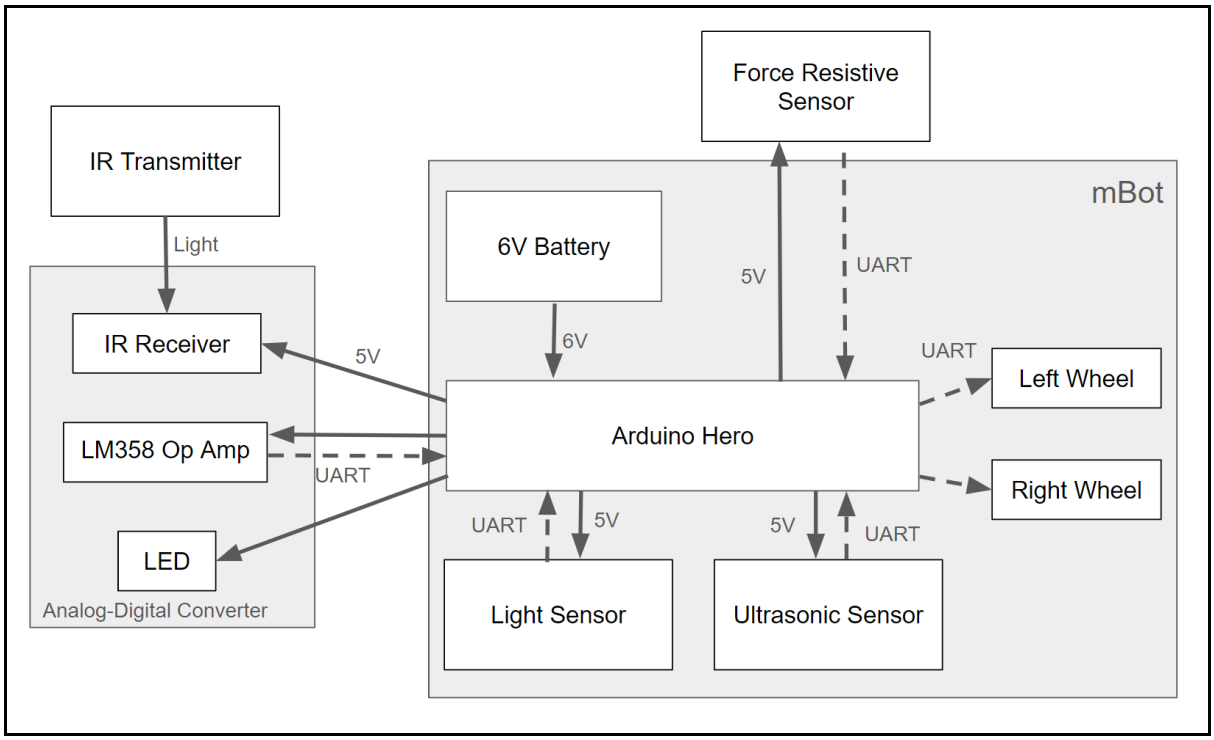


Fig 2.1. Prototype Functional Block Diagram

2.2 COMPONENT LIST

Item	Quantity
mBot robot kit	1
IR LED and Photodiode	1
Resistive force sensor	1
Variable resistor	1
LM358 OpAmp	1
Vero board for soldering	2

Table 2.2. Prototype Component List

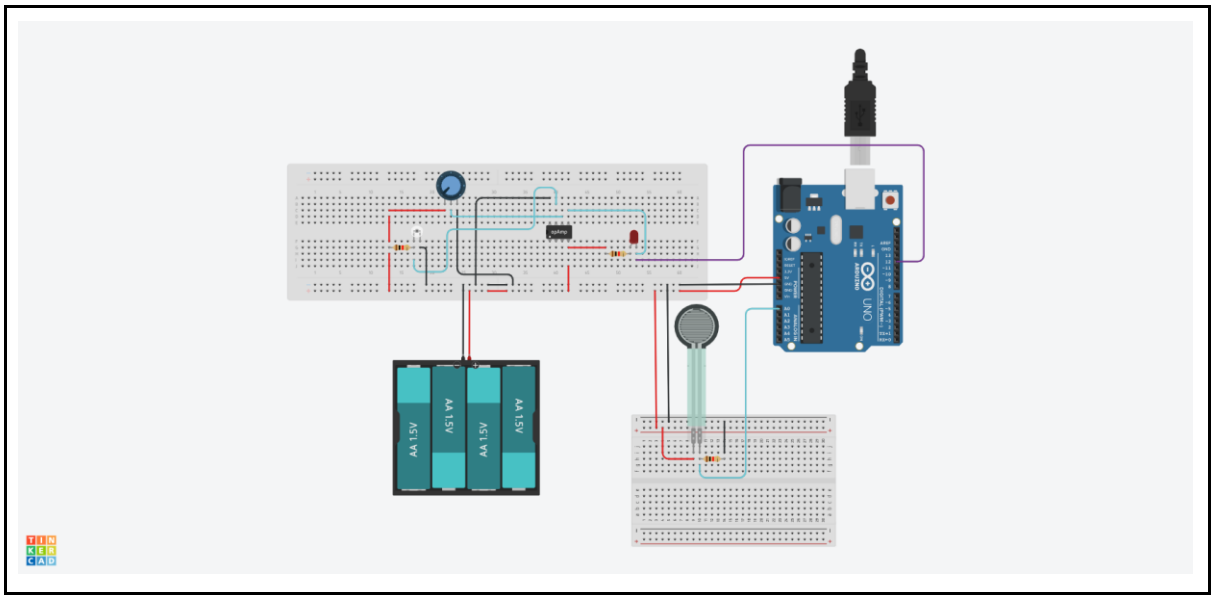


Fig 2.3.1. Infrared Receiver & Force Resistor Circuit Connection Diagram

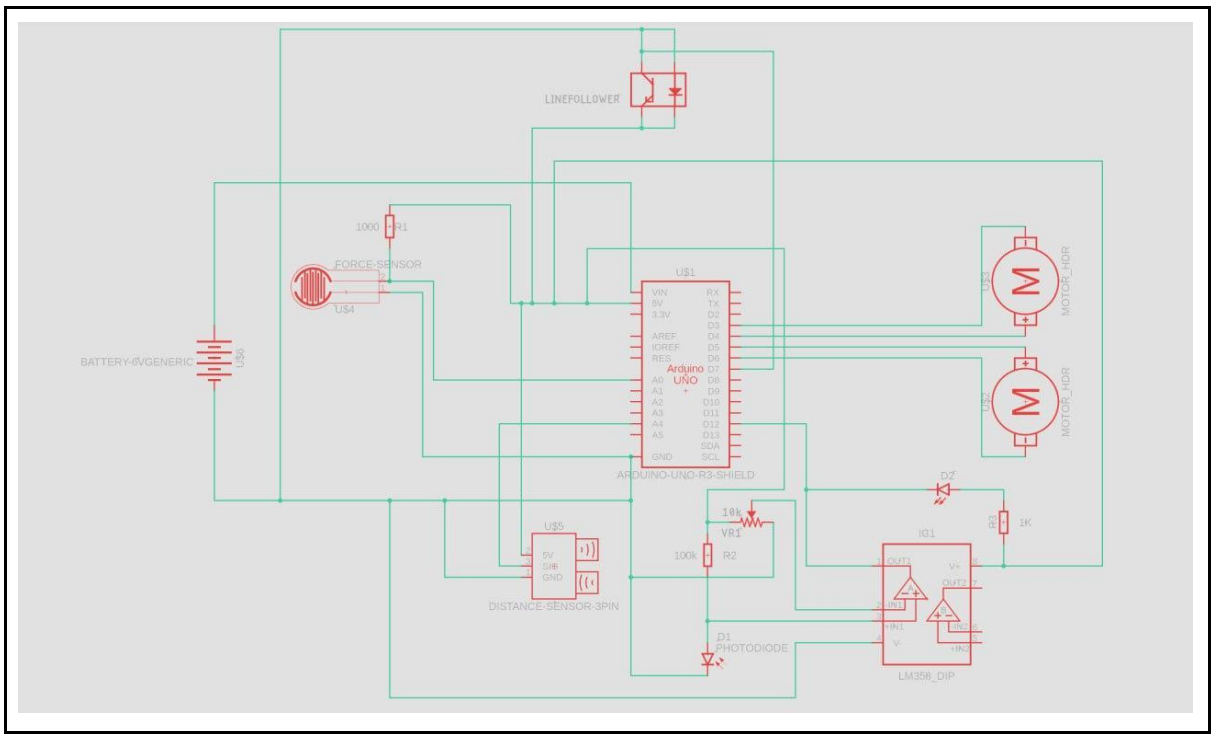


Fig 2.3.2. Infrared Receiver & Force Resistor Circuit Diagram

2.4 PROTOTYPE PHOTO

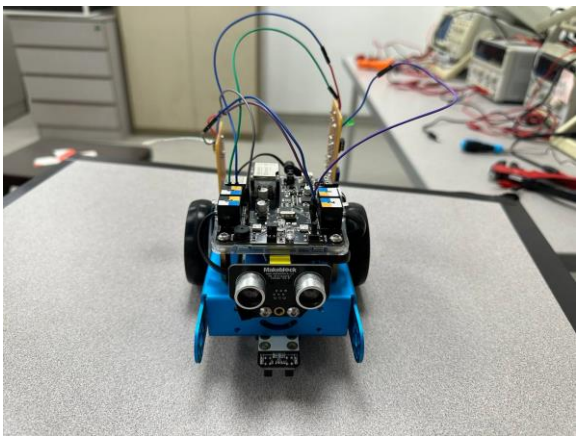


Fig 2.4.1. Prototype Front View

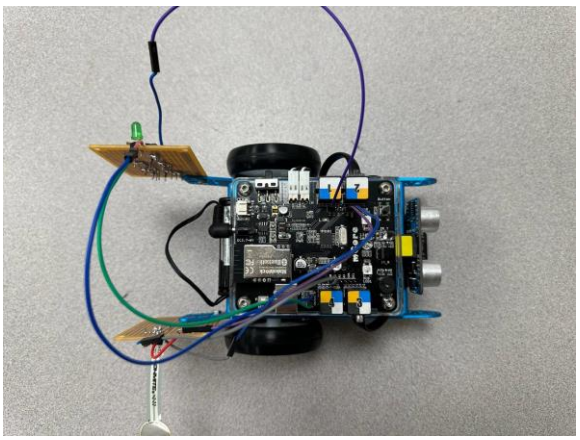


Fig 2.4.2. Prototype Bird's Eye View

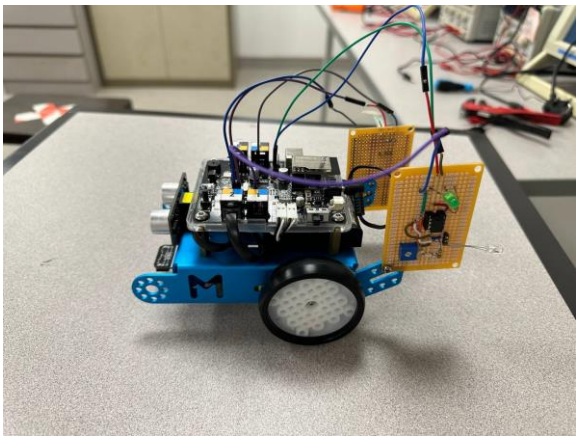


Fig 2.4.3. Prototype Left Side View

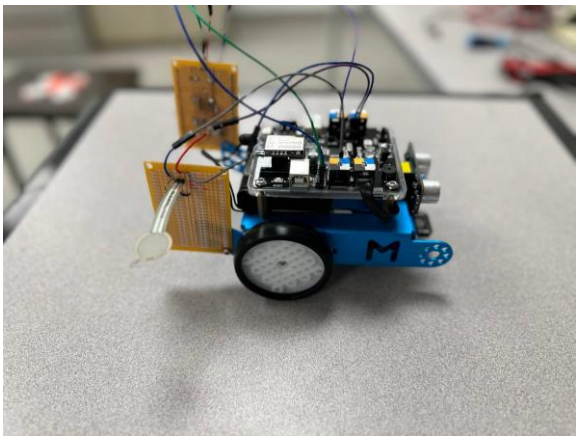


Fig 2.4.4. Prototype Right Side View

This project utilises one line finder, one ultrasonic sensor, two motors, one Arduino Uno board, one IR (Infrared) receiver, and the buzzer onboard the robot. This robot aims to complete 3 tasks.

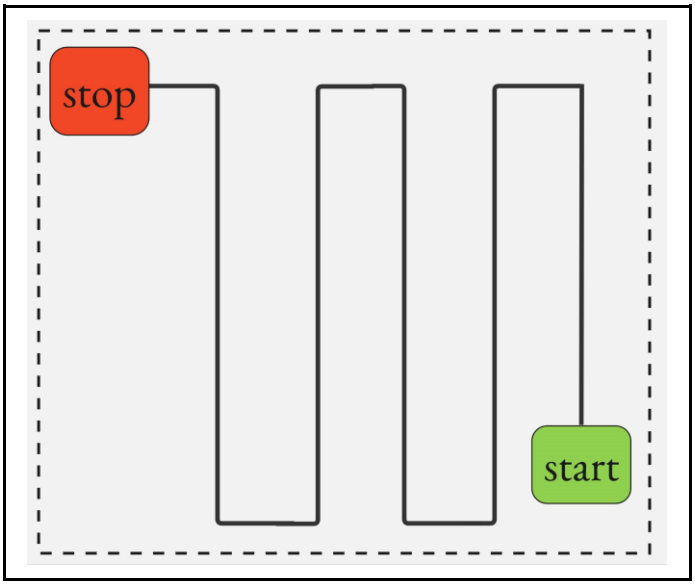


Fig 2.5.1 Snake Pattern

The robot is first tasked to navigate in a snake pattern (*Fig 2.5.1*) inside a 50 cm x 50 cm parameter. The parameter has a square shape. Black tape with a thickness roughly of 4 cm is stuck on the parameter. The robot will start at the left top of the square area. The default movement direction will be forward. Meanwhile, a global count variable will be set to zero. Once the light sensor detects the black tape the value of the count variable will be added by 1. Once reaching this value, the robot will execute a U-turn. The program also has an in-built counter variable, keeping track of the number of times black tape is detected. This count number dictates the U-turn direction: If the count variable value is even, the robot will turn clockwise by 180 degrees. Otherwise, if the value of the count variable is odd, the robot will turn counterclockwise by 180 degrees.

The second task is to avoid an opaque obstacle put randomly inside the parameter. The object is within the size of 10cm x 10cm. To avoid the obstacle, the robot's ultrasonic sensor flags up and executes a designated path to avoid the obstacle and return to its initial track. if the global count variable is odd, the robot will avoid the obstacle by first turning to its left. In the other case, if the count is even, the robot will first turn to its right. Basically, it will turn in the opposite direction from the last turn at the wall. which is large enough to avoid any obstacle of the predetermined size. We also added functions to prevent the robot from moving outside of the taped area during the object avoidance process.

We encountered one major challenge while devising this algorithm. Namely, the difficulty in distinguishing the walls from the obstacle. When the light sensor detects the black tape, there is a concern that the ultrasonic sensor will detect the walls first, therefore falsely triggering the obstacle avoidance logic of the program. Therefore we established a truth table to achieve this feature. As illustrated in the figure, the conditions are defined as follows: if the ultrasonic sensor's reading is less

than or equal to 'x' and the black line is detected, the outcome is considered True; conversely, if the ultrasonic reading is greater than 'x' and the black line is not detected, the outcome is considered False. When both conditions are True, the detected entity is classified as a wall, prompting an increment of 1 to the counter. If the conditions are True and False, respectively, the detected entity is identified as an obstacle.
When the robot finishes covering the entire 50cm * 50cm area, it will stop and delay for a few seconds before going back to the port. The robot will first turn 90 degrees counterclockwise and move forward until it has reached the black tape on the other side of the square area. It will iteratively continue this movement until the IR receiver on board receives the signal from the IR transmitter. That is when we will stop the motors and park the robot.

This prototype features a dust detection mechanism as well. It utilises a pressure sensor that mimics the condition of a full dust bag exerting pressure on its container. If the force exerted surpasses a predefined threshold, the robot's onboard buzzer is activated to play a tune. Consequently, this system serves to alert the user when the dust bag is nearing capacity.

Vero boards are put at the bottom sides of the robot. As such, this placement does not increase the robot's width, making it less likely to hit the obstacle when turning. The infrared receiver is also placed at the bottom horizontal to the ground. Its elevation is also adjusted to the height of the transmitter. These placement arrangements maximise the chance the IR receiver receives the signal.

2.6 PROTOTYPE CODE

2.6.1 PROGRAMMING LOGIC FLOWCHART

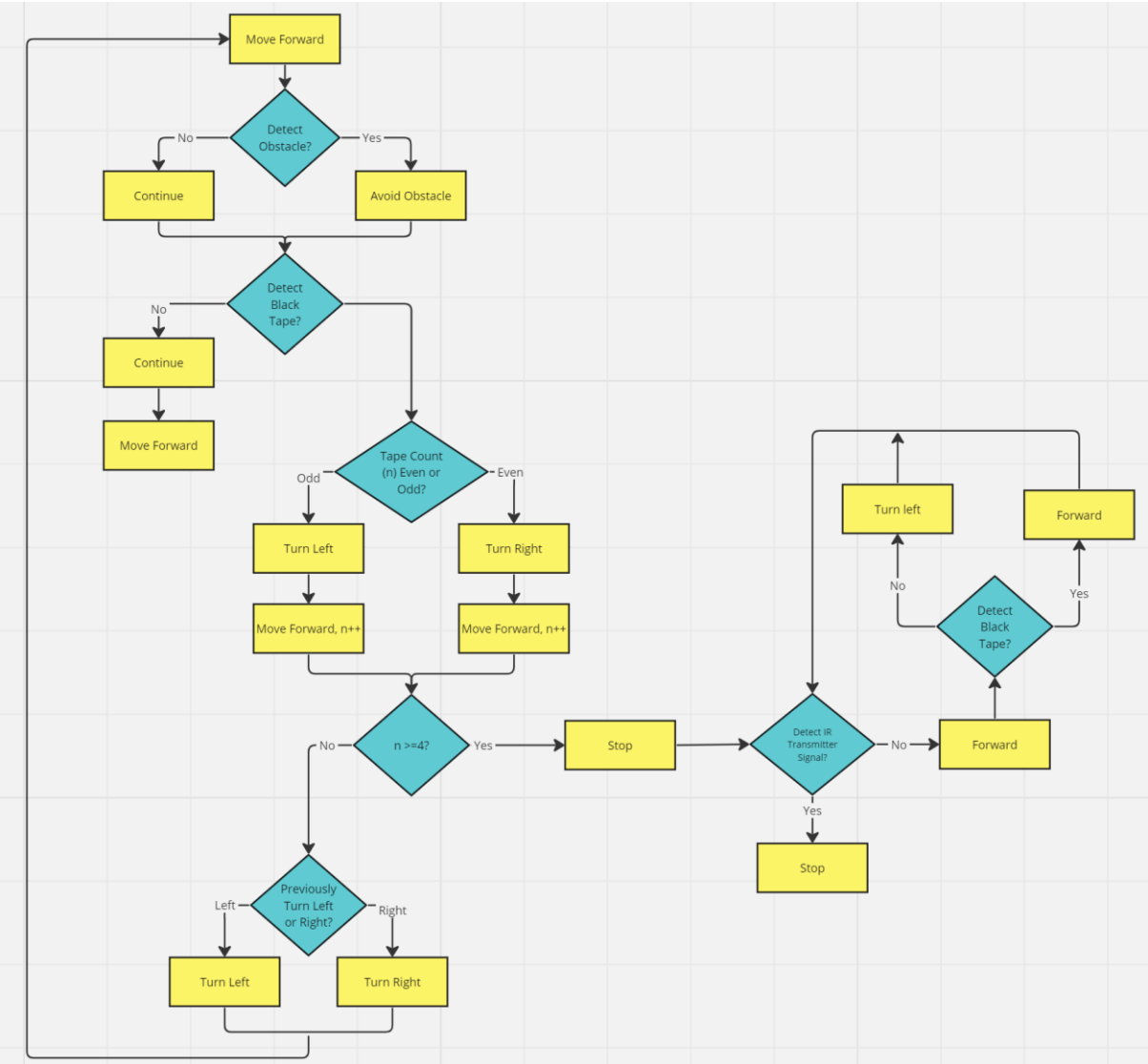


Fig 2.6.1.1. Logic Flowchart of Robot Motion, Obstacle Avoidance and Docking

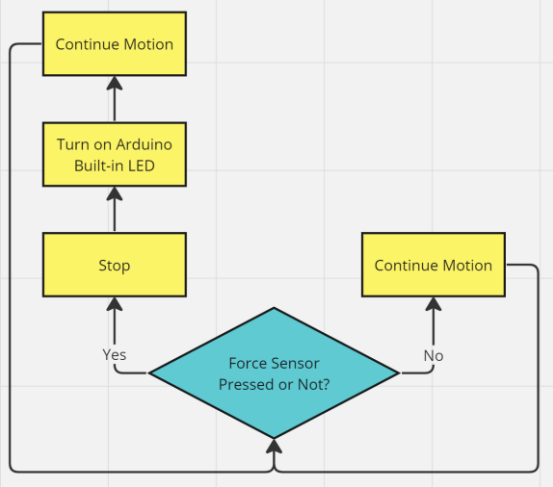


Fig 2.6.1.2. Logic Flowchart of Force Resistance Sensor Simulating a Full Dust Bag

2.6.2 PRESENTED CODE

```
#include "MeMCore.h"                    //include necessary libraries, assign ports and constants
#include <MeMCore.h>
#include <SoftwareSerial.h>
#include <Wire.h>
#define NOTE_G4 392
MeBuzzer buzzer;

MeDCMotor motor3(M1);
MeDCMotor motor4(M2);
MeLineFollower lineFinder(PORT_2);
MeUltrasonicSensor ultraSensor(PORT_3);
int val1 = 0;
int val2 = 0;
int val = 0;
int count = 0;
int dis = 0;

void setup() {
  Serial.begin(9600);
  pinMode(8, OUTPUT);
  pinMode(12, INPUT);
  pinMode(LED_BUILTIN, OUTPUT);       //set up code
}

void lcorrect() {                     //left-side correction manoeuvre
  motor3.run(80);
  motor4.run(0);
  delay(500);
}

void rcorrect() {                     //right-side correction manoeuvre
  motor3.run(-80);
  motor4.run(0);
  delay(500);
}

void turnl() {                         //left turn sequence
  motor3.run(100);
  motor4.run(100);
  delay(600);
  motor3.run(100);
  motor4.run(-100);
  delay(700);
  motor3.run(100);
  motor4.run(100);
  delay(600);
}

void turnr() {                         //right turn sequence
  motor3.run(-100);
  motor4.run(-100);
```

```
  delay(600);
  motor3.run(100);
  motor4.run(-100);
  delay(700);
  motor3.run(-100);
  motor4.run(-100);
  delay(600);
}

void forwardh() {                          //go forward sequence (high speed)
  motor3.run(254);
  motor4.run(-254);
}

void stop() {                              //stop sequence
  motor3.run(0);
  motor4.run(0);
}

void backwardh() {                         //go backward sequence (high speed)
  motor3.run(-254);
  motor4.run(254);
}

void forward() {                           //go forward sequence
  motor3.run(70);
  motor4.run(-70);
}

void backward() {                          //go backward sequence
  motor3.run(-80);
  motor4.run(80);
}

void Buzzer() {                            //activate buzzer
  buzzer.tone(8, 392, 1000);
}

void LEDon() {                             //turn on LED
  digitalWrite(LED_BUILTIN, HIGH);
  delay(1000);
  digitalWrite(LED_BUILTIN, LOW);
  delay(1000);
}

void ob() {                                //obstacle avoidance command
  if (count % 2 == 0) {                      if global variable count is an even number, turn left and
    motor3.run(80);                          go around the obstacle
    motor4.run(80);
    delay(850);
    motor3.run(150);
    motor4.run(-150);
    delay(600);
    motor3.run(-80);
```

```
   motor4.run(-80);
   delay(850);
   motor3.run(120);
   motor4.run(-120);
   delay(1600);
   motor3.run(-80);
   motor4.run(-80);
   delay(850);
   motor3.run(150);
   motor4.run(-150);
   delay(600);
   motor3.run(80);
   motor4.run(80);
   delay(900);
  }
  else {                                  //if global variable count is an odd number, turn right and
   motor3.run(-80);                         go around the obstacle
   motor4.run(-80);
   delay(850);
   motor3.run(150);
   motor4.run(-150);
   delay(600);
   motor3.run(80);
   motor4.run(80);
   delay(850);
   motor3.run(120);
   motor4.run(-120);
   delay(1600);
   motor3.run(80);
   motor4.run(80);
   delay(850);
   motor3.run(150);
   motor4.run(-150);
   delay(600);
   motor3.run(-80);
   motor4.run(-80);
   delay(900);
  }
}

void back() {                              //docking sequence
  forward();
  if (val == 0)                            //if detect white, turn left
    stop();
    delay(200);
    motor3.run(100);
    motor4.run(100);
    delay(600);
  }

  if (digitalRead(12) == 0) {
    stop();
    delay(100000);
  }
```

```
}

void black()                          //if detect black tape, variable count +1
  stop();
  delay(500);
  count++;
  delay(500);
  if (count % 2 == 0) {               //if count is an even number
    if (count == 4) {                 //if count is 4, turn left
      stop();
      delay(200);
      motor3.run(100);
      motor4.run(100);
      delay(600);
    } else {                          //if count is 2, turn right
      stop();
      delay(500);
      turnr();
      stop();
      delay(500);
    }
  } else {                            //if count is an odd number, turn left
    stop();
    delay(500);
    turnl();
    stop();
    delay(500);
  }
}


void loop() {
  val1 = lineFinder.readSensor1();
  val2 = lineFinder.readSensor2();
  val = lineFinder.readSensors();
  dis = ultraSensor.distanceCm();
  forward();
  if (val < 3 && count < 4) {         //if detect black tape and count is smaller than 4
    black();
  }
  if (dis < 10) {                     //if ultrasonic sensor reading is smaller than threshold,
    stop();                           run ob(), if touch black tape and count
    delay(500);                       smaller than 4, run black()
    ob();
    if (val < 3 & count != 4) {
      black();
    }
  }
  if (analogRead(A0) > 560) {
    stop();
    LEDon();
    delay(2000);
  }
```

```
  if (count >= 4) {                          //detect black tape 4 times, run back()
    back();
  }
}
```

2.7 SIZING CALCULATIONS

2.7.1 Prototype Power Consumption Table

Component	Model Number	Quantity	Voltage (V)	Current (mA)	Power (mW)
Arduino Hero	-	1	6	200	1200
Motor	ADA-3777	2	6	400	4800
Ultrasonic Sensor	AE-MB-11001	1	5	25	125
Line Sensor	-	1	5	100	500
LM358 OpAmp	-	1	5	1.2	6.0
IR Receiver	-	1	2.5	0.45	1.125
LED	-	1	2	20	40
Total Power					6672.125

Table 2.7.1. Prototype Power Consumption Table

2.7.2 Sizing Considerations

Determining the overall power usage is crucial to ensure that every robot operation is accomplished. The following calculation aims to demonstrate how power sources are chosen and the estimation of functional runtime. All numerical data is based on the Prototype Power Consumption Table *(Table 2.7.1).*

Total Power required = 6672.13 mW

As seen in *Table 2.7.1*, components such as the *Arduino Hero* and *motors* require 6 V to function. As such, 4 standard 1.5 V AA Alkaline Batteries are utilised and put in series. As such, the accumulative voltage supplied is 6 V. The *Arduino Hero* has an in-built voltage regulator that allows the Arduino to output various voltages, catering to voltage requirements from different components.

Voltage Supplied = 4 × 1.5 V= 6 V

It is known that each AA battery possesses an average capacity of 2500 mAh (Microbattery.com Blog). As such, the total source capacity, which is the same as the individual battery capacity, is 2500 mAh.

Total Source Capacity = 2500 mAh

As such, the total energy supplied could be calculated as below.

$$Total\ Energy\ Supplied = 2500\ mAh \times 6\ V = 15000\ mWh$$

Given the required power used, an estimation of runtime could be calculated as below.

$$Runtime = \frac{Total\ Energy\ Supplied}{Required\ Power} = \frac{15000}{6672.13} \approx\ 2.25\ hours$$

As such, the robot has an estimated runtime of 2.25 hours before the battery array entirely drains.

In reality, the robot batteries drain much faster than expected, causing a rapid decrease in the voltage supplied. A drop in voltage results in abnormal motor behaviors. Specifically, the robot does not turn with a preset angle and moves at a relatively slow speed. This causes disruptions during prototype testing and the robot demands battery changes frequently. To facilitate prototype testing, a threshold time of 1 hour is set to indicate a battery change, since the robot begins to behave abnormally after roughly one hour.

3. HIGHER LEVEL DESIGN

3.1 HIGHER LEVEL DESIGN FUNCTIONAL BLOCK DIAGRAM

3.1.1 Functional Block Diagram

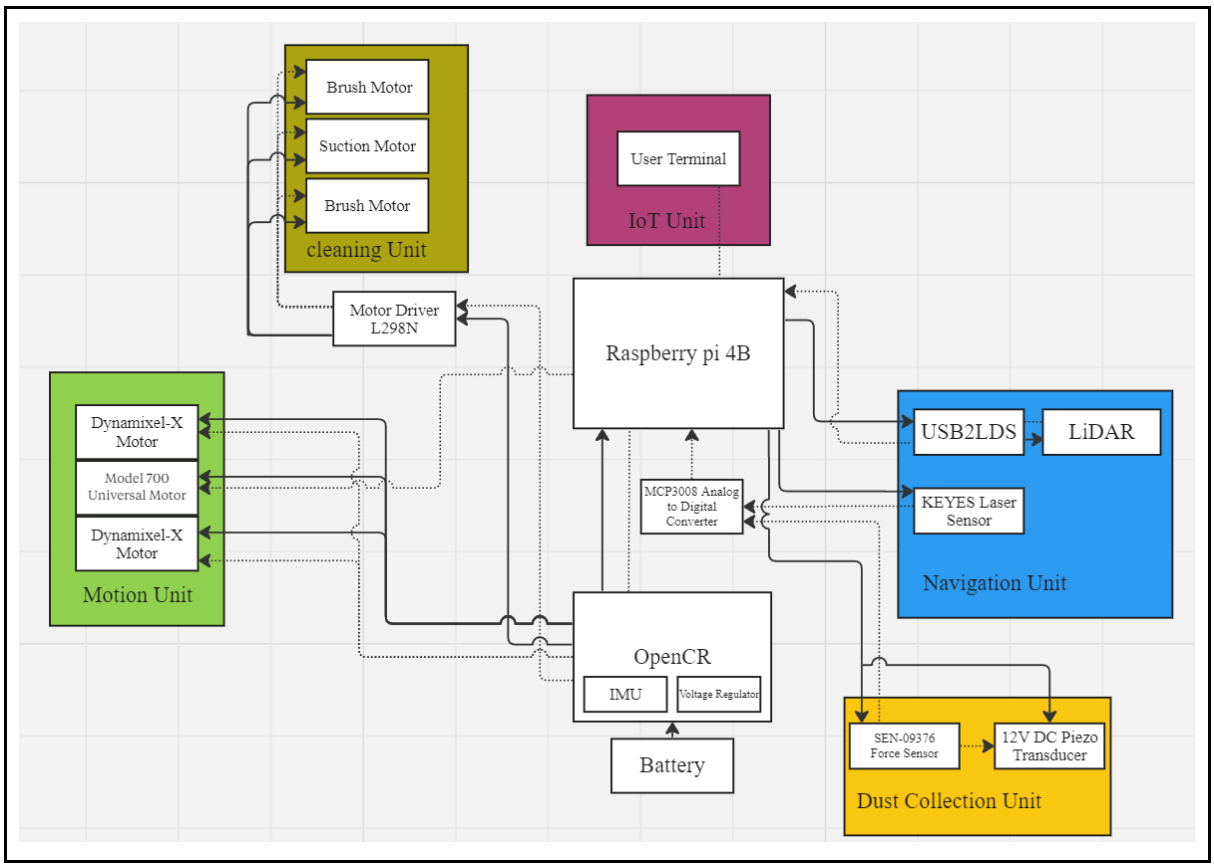


Fig 3.1.1. Higher level functional block diagram

3.1.2 State Machine Diagram

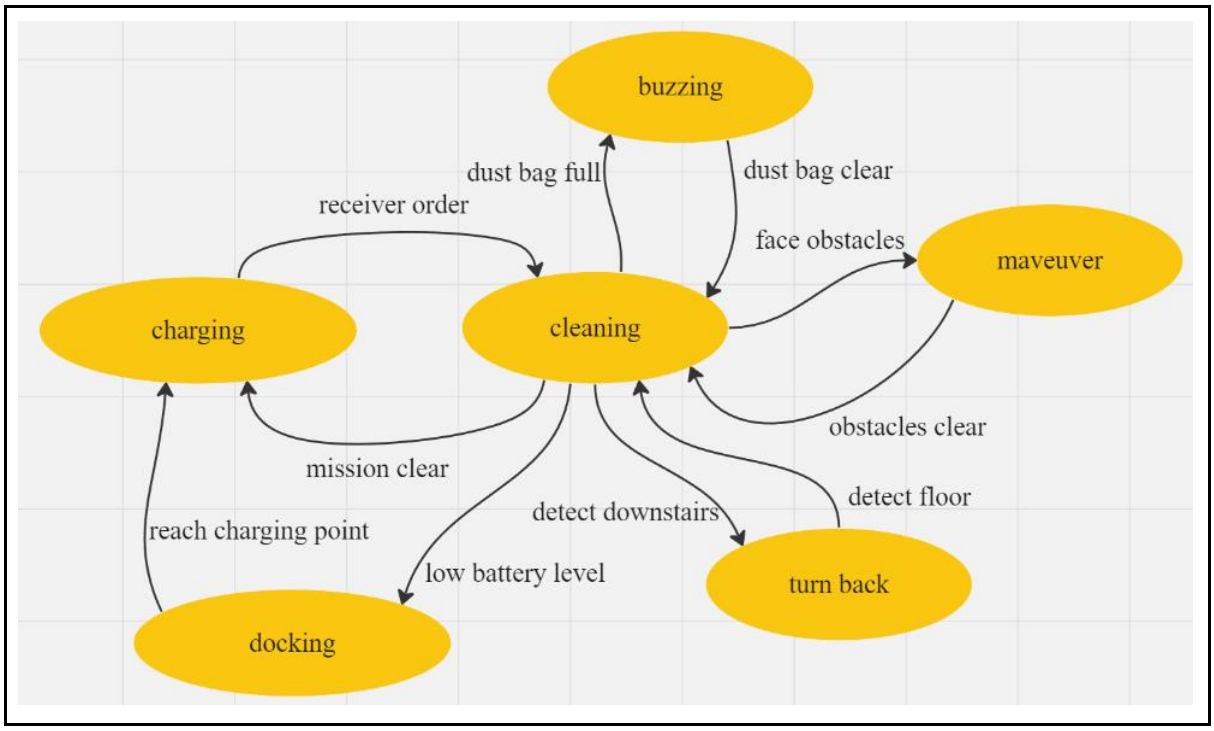


Fig 3.1.2. State machine diagram

3.1.3 Higher Level Design Functions

Navigation:

- Scanning and Memory Storage

  Our robot will use LiDAR (Light Detection and Ranging) device to scan the surrounding environment. LiDAR is a remote sensing technology that measures the distance to an object by emitting laser pulses and measuring the time it takes for them to bounce back. This method allows for precise distance measurements and object detection. From there, the robot will create a map of the environment and store the data in the memory. This data is used for the design of robot paths.

- Path Planning

  After consideration, we decided to apply three different algorithms to do the path planning. The first algorithm is the Frontier-Based Exploration Algorithm, which detects frontiers to locate unexplored places to navigate. The robot will move to the frontiers created in the occupancy grid or map using LiDAR. To move from a certain point to another, path planning and optimization must be employed. The second algorithm is the Hybrid Genetic Algorithm (HGA), which utilises the concepts of selection (fitness function), mutation, and evolution of chromosomes to select the best path to navigate to a frontier. After optimising the established path and considering obstacles, the third algorithm is the A* algorithm, and this is used to achieve point-to-point motion in the lowest-cost path using a heuristic function (best-first-search). One of the assumptions of the above algorithms is that the obstacles in the room are static. To detect moving obstacles, we consider integrating Kalman filters with the above path planning processes.

Wireless Charging:

The wireless charging mechanism is based on an inductive power transfer (IPT) system. The system uses curved coils and flexible ferrite sheets due to the curved design of the robot surface. Ferrite material is used to enhance the magnetic field between the primary and secondary IPT pads Additionally, power repeaters are implemented on the charging station to improve the range of wireless charging. Basically, they are used to direct the magnetic fields induced to the receiving coil. Therefore, a combination of these two features optimises the charging efficiency of the system.

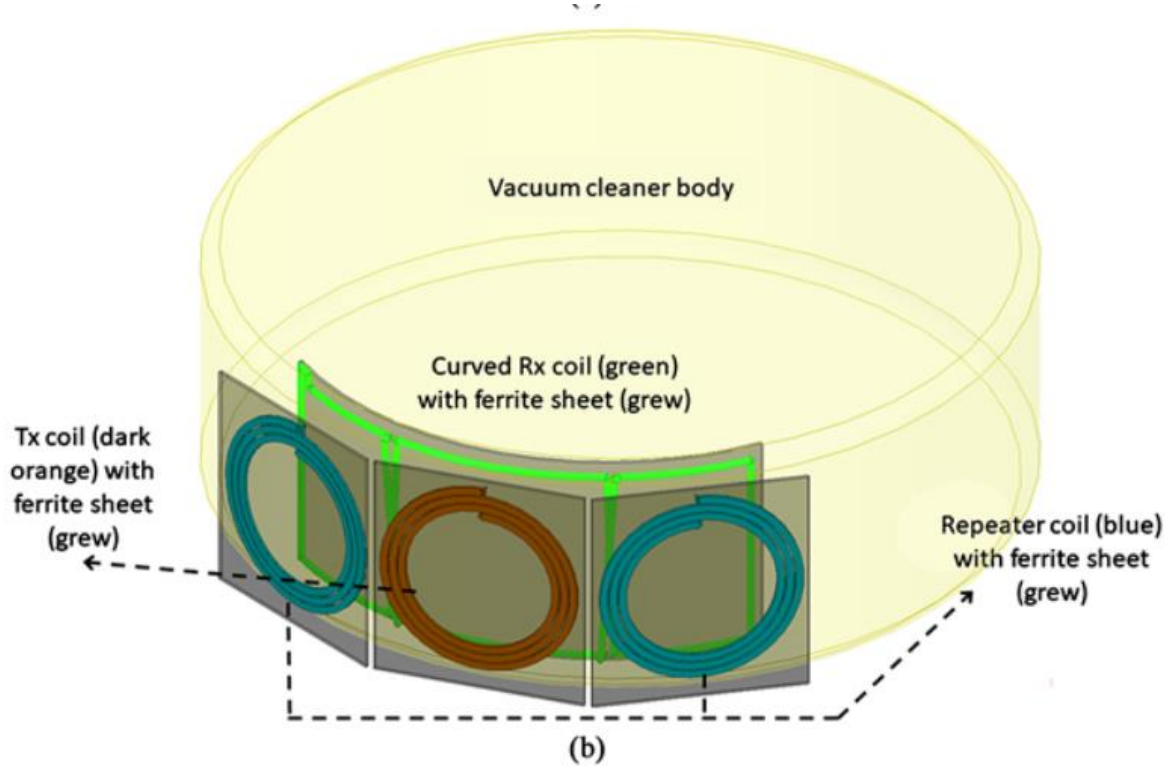


Fig 3.1.3.1. Inductive Power Transfer, placement of TX and RX coils

The charging dock design is as follows. The black area corresponds to where the TX coils are located.

Fig 3.1.3.2. Charging dock design

Suction Mechanism:

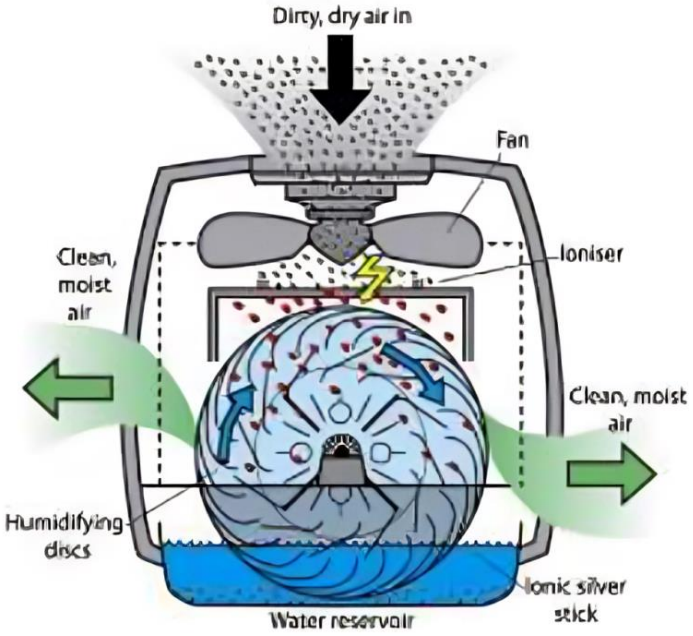


Fig 3.1.3.3. Filtering mechanism; HEPA filter

The High-Efficiency Particulate Air (HEPA) is crucial for trapping fine particles to prevent them from circulating back into the surrounding environment. As the vacuum cleaner operates, it pulls in air and debris from the floor. The larger particles are collected into the dust compartment, but the air carrying the smaller particles passes through the HEPA filter. The HEPA system is composed of a mat of fine meshes made of fiberglass. As the vacuum's motor activates the fan, a pressure drop is created, drawing ambient air along with dust and other pollutant particles into the device. Subsequently, an ionizer charges the airborne particles, enhancing the filtration process by allowing the particles to settle out of the air. Meanwhile, Rotating humidifying discs are designed to be partially submerged in water. As air moves through these discs, the contaminants will be physically trapped by the wet surfaces. This also ensures that the air that exits the purifier is clean.

The higher-level design addresses various limitations observed in the prototype. Specifically, the prototype's pathfinding capabilities are hindered by its constrained computing resources. Additionally, the use of a mBot in the prototype restricts opportunities for implementing additional functional enhancements. Consequently, this research showcases a higher-level design that integrates components sourced independently to enable diverse functional implementations. These enhancements include a more sophisticated pathfinding algorithm, human-machine interaction facilitated by IoT, wireless charging technology, and improved cleaning efficiency.

In terms of a more advanced pathfinding logic, the higher level design utilises various algorithms to navigate around a complex and dynamic home environment. To achieve full automation, the robot uses data points, also called ‘frontiers’, from LiDAR to position itself, while respective sensors assist to position and store velocity of the robot. Hybrid genetic algorithm and A* algorithms are utilised to generate and navigate paths among frontier points. The aforementioned data is fed to the Kalman filter in order to adjust the robot's motion. The navigation algorithms coupled with the state adjustment model enable the robot to avoid static and dynamic obstacles. To prevent the robot from falling from the floors, not only does the algorithm set boundaries during navigation, the robot also has three laser sensors put around the exterior, pointing at the ground. Laser sensors command the robot to promptly stop when the laser sensor detects a change in depth.

For efficient cleaning, the advanced design integrates a HEPA (High-Efficiency Particulate Air) filter to capture minute particles. Similarly to the prototype, a force sensor alerts users when the dust bag reaches capacity by activating an onboard buzzer. Two all-surface brushes are deployed to cover larger areas effectively.

Furthermore, the design features a wireless charging dock for the robot, facilitating automatic recharging when battery levels are low. The charging dock employs an inductive power transfer (IPT) system, with necessary computations executed to optimize energy exchange.

Moreover, the advanced design explores the potential for user manipulation of the robot's trajectory through IoT implementations. This entails adapting the WebSocket Server to enable real-time bidirectional communication, achieved via HTTP calls and requests between the host computer and Raspberry Pi.

3.3 HIGHER LEVEL DESIGN PSEUDO-CODE

3.3.1 Navigation and Path Planning

Here is an overall pseudocode for the integrated algorithms discussed in Section 3.1.3.

```
Initialize robot position, robot current direction of movement, map, LiDAR sensor, and Kalman filter

while there are unexplored areas in the environment:
    Use lidar to detect frontiers and update the frontier points.
    if there are frontier points:
        Use the hybrid genetic algorithm to generate paths for coverage between frontier points.
        Use A* algorithm to navigate the robot along the generated paths.
        while the robot is navigating along the current path: // the robot is trying to move to the frontiers
            Continuously monitor for new frontier detections.
            if new frontiers are detected:
                Update the frontier points and generate a new path covering all frontiers.
                Merge the new path with the current path to ensure coverage of both old and new
frontiers.
                Re-plan the robot's navigation using A* to follow the merged path.
            Measure robot position using sensors (IMU)  and update the Kalman filter:
                Predict state estimate and covariance matrix using motion model.
                if there are moving obstacles detected:
                    Include obstacle position and velocity measurements into the Kalman filter update.
                    Correct state estimate using Kalman gain to avoid obstacles.
                    Move the robot forward in the current direction while avoiding obstacles.
                else:
                    Update Kalman filter with robot position and velocity measurements.
            if an obstacle is detected: // zig-zag movement
                Determine whether to turn left or right based on the current direction of movement.
                Change the direction of movement to the opposite direction.
                Move the robot forward in the new direction.
             if robot try to move to a frontier outside of the room due to map leakage:
                Exclude frontier point from the list of frontiers and neglect it
        if the current exploration path is complete:
            Change the direction of movement (e.g., from forward to backward or vice versa).
            Continue exploring until all areas are covered and the map is closed.

If all areas are covered:
    Use HGA to generate a path from the current robot position back to the starting/charging point.
    Use A* algorithm to navigate the robot along the generated path.
    Move the robot along the path until it reaches the starting point.
```

Here is the flowchart of the above pseudocode to help visualize the logic behind it:

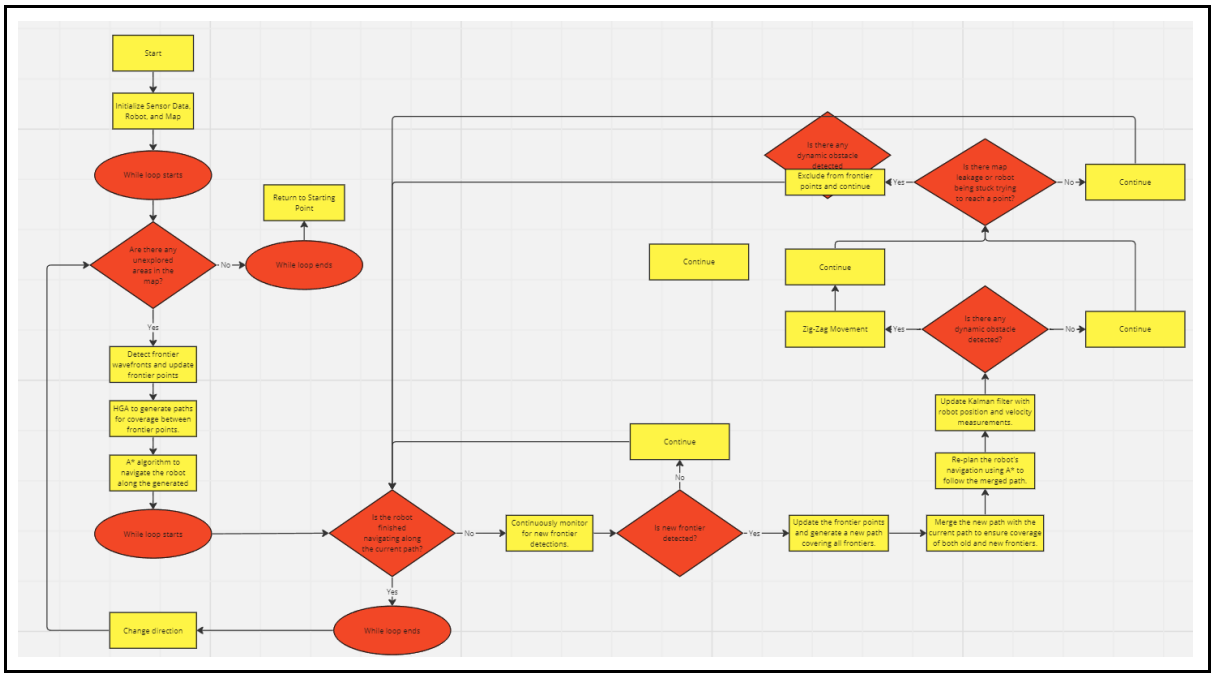
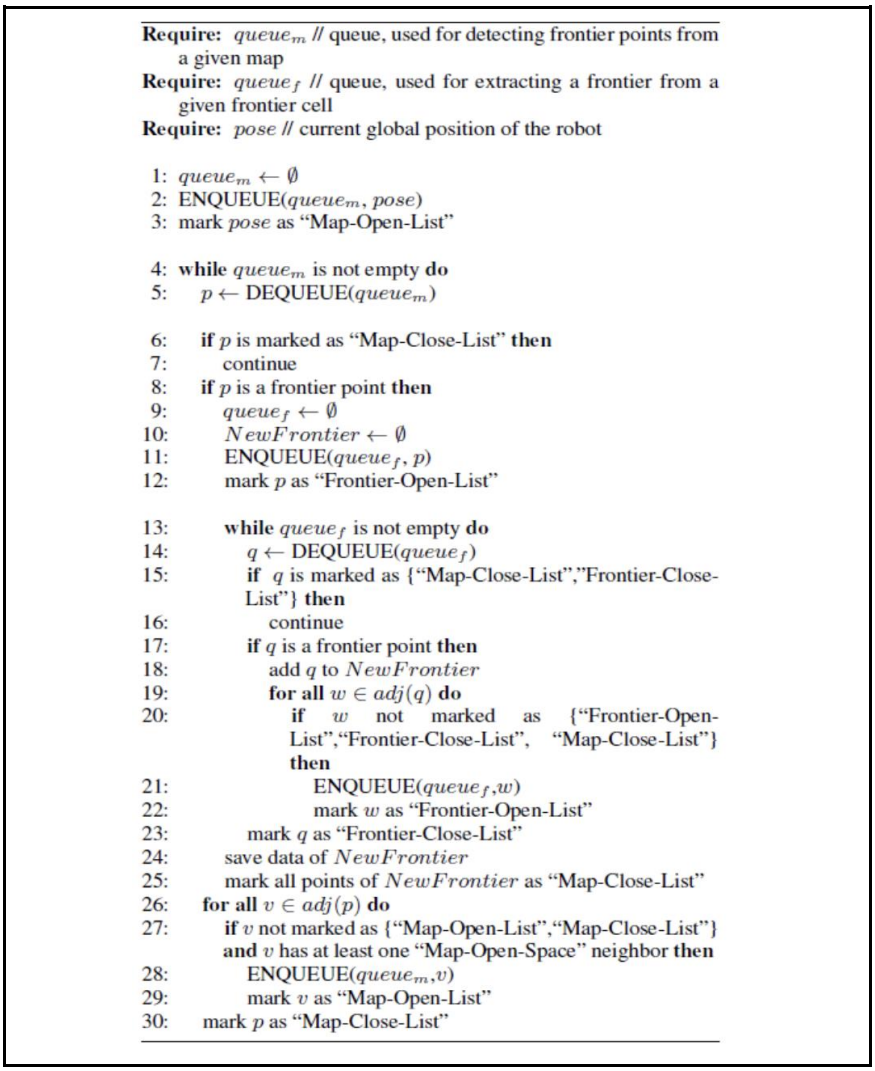


Fig 3.3.1. Navigation and Path Planning Flow Chart

While the concepts of the path planning implemented above are rigorously hard to understand, each algorithm will be discussed in following sections.

3.3.1.1 Path Establishment (Wavefront Frontier Detector Algorithm)

Topiwala et al. (2018) implements the Wavefront Frontier Detector Algorithm to find frontiers and then use SLAM and move base, which is a planning algorithm package to plan a path to move to the nearest frontier iteratively until the complete environment is mapped. Here is the pseudocode for the algorithm.



Require: $queue_m$ // queue, used for detecting frontier points from a given map
Require: $queue_f$ // queue, used for extracting a frontier from a given frontier cell
Require: $pose$ // current global position of the robot

1: $queue_m \leftarrow \emptyset$
2: ENQUEUE($queue_m$, $pose$)
3: mark $pose$ as "Map-Open-List"

4: **while** $queue_m$ is not empty **do**
5: $p \leftarrow$ DEQUEUE($queue_m$)

6: **if** p is marked as "Map-Close-List" **then**
7: continue
8: **if** p is a frontier point **then**
9: $queue_f \leftarrow \emptyset$
10: $NewFrontier \leftarrow \emptyset$
11: ENQUEUE($queue_f$, p)
12: mark p as "Frontier-Open-List"

13: **while** $queue_f$ is not empty **do**
14: $q \leftarrow$ DEQUEUE($queue_f$)
15: **if** q is marked as {"Map-Close-List","Frontier-Close-List"} **then**
16: continue
17: **if** q is a frontier point **then**
18: add q to $NewFrontier$
19: **for all** $w \in adj(q)$ **do**
20: **if** w not marked as {"Frontier-Open-List","Frontier-Close-List", "Map-Close-List"} **then**
21: ENQUEUE($queue_f$,w)
22: mark w as "Frontier-Open-List"
23: mark q as "Frontier-Close-List"
24: save data of $NewFrontier$
25: mark all points of $NewFrontier$ as "Map-Close-List"
26: **for all** $v \in adj(p)$ **do**
27: **if** v not marked as {"Map-Open-List","Map-Close-List"} **and** v has at least one "Map-Open-Space" neighbor **then**
28: ENQUEUE($queue_m$,v)
29: mark v as "Map-Open-List"
30: mark p as "Map-Close-List"

Fig 3.3.1.1. Wavefront frontier algorithm pseudocode

In the above pseudocode, first, a queue data structure is initialized to find all the frontier points in the occupancy grid. Points enqueued (resp. dequeued) to this list are marked as Map-Open-List (resp. Map-Closed-List). Only points that are not a part of Map-Open-List or Map-Close-List are considered to be added to the queue of frontier points. In addition to this, an additional constraint is added viz., each point being added to this queue must have at least one open-space neighbor (Line 27). This is the most crucial part of the algorithm as it ensures that only known regions are scanned.

At the start, the point in the occupancy grid corresponding to the current pose of the robot is enqueued into a queue data structure. Next, a breadth-first-search (BFS) is performed until a frontier point is encountered. Once such a point is encountered, a new queue is initialized to extract its frontier. Only

points that are not a part of the Map-Open-List are enqueued to this queue (Line 6). The frontier point obtained from the outer BFS is first enqueued to this queue. Now, a fresh BFS is performed to find all connected frontier points to extract its frontier.

All points being visited in the inner BFS are marked as Frontier-Open-List. Whenever a frontier point is encountered in this BFS, it is stored in a list containing all other frontier points in connection with their respective frontier. These points are marked as Frontier-Closed-List. A frontier point is added to a frontier list only if it is not a part of the Frontier-Open-List, Frontier-Close-List, or Map-Closed-List (Line 20). This ensures that each frontier point is assigned to a unique frontier. Using this scanning policy, it is guaranteed that all frontier points will be assigned to a unique frontier at the end of each algorithm run.

3.3.1.2 Path Coverage Optimization (Hybrid Genetic Algorithm)

A hybrid genetic algorithm (HGA), which is similar to the memetic algorithm (MA), combines the principles of genetic algorithms (GAs) with other optimization techniques or problem-solving methods to enhance performance and address specific challenges. Genetic algorithms are stochastic search algorithms inspired by the principles of natural selection and genetics, commonly used for optimization and search problems (Yaaximus, 2020). Singh et al. (2013) model the flowchart of the HGA as follows.

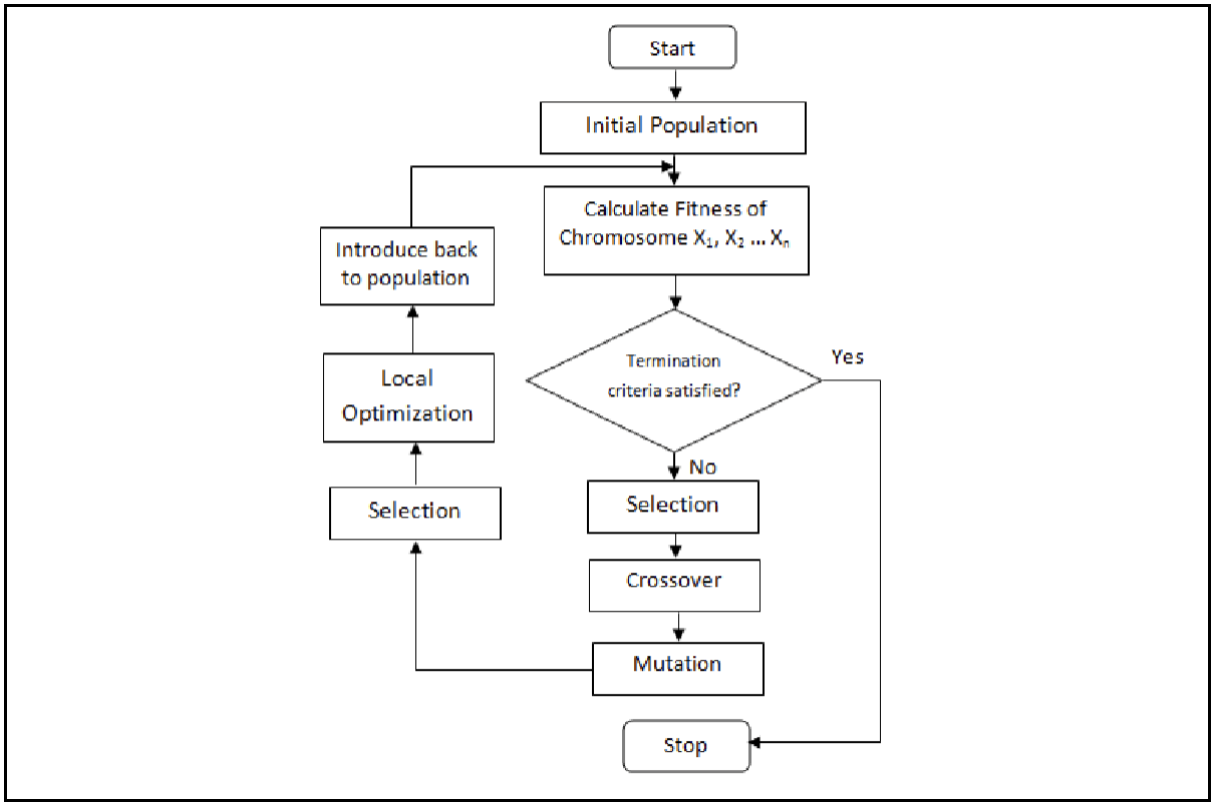


Fig 3.3.1.2. Flowchart of memetic algorithm

In our context, we use a random selection search to generate initial solutions from the starting node to the end node, looking from the occupancy map. We also use the total manhattan distance to calculate the fitness of each route. After selecting two chromosomes to crossover and produce a child, the child

mutates and we append it to the list of new generations. We then use the Hill local search algorithm to select the best chromosome. Combined with the efforts from Garg (2009), Gonzalez et al. (2020), and Yang (2015), we arrive at the pseudocode for the Hybrid Genetic Algorithm below.

```
Input: occupancy map data, s_start, s_end, pop_size
Output: optimal cleaning routes for the vacuum cleaner robot
1. INITIALIZATION
2: gen = 0, max_gen, cross_rate, mutation_rate
3: C = generate_random_routes(s_start, s_end, pop_size)
4: D =  generate_random_routes(s_start, s_end, pop_size)
5: C.union(D)
6: EXECUTION
7: while gen <= max_gen do  # Termination condition
8:      E = C.copy()
9.      total_distances = {}  # Dictionary to store manhattan distances for each route in E
10:     route_numbers = {e: i + 1 for i, e in enumerate(E)}
11.     for route in E do      # {1 : dist(E1), 2 : dist(E2)....}
12:           total_distances[route_numbers[route]] = total_manhattan_distance(route) #func = tmd
13:     sorted_routes = sorted(total_distances.keys(), key=lambda x: total_distances[x])
14:     new_generation = []
15:     for i in range(pop_size) and for j in range(i, pop_size) and i != j do
16:          mate1 = get_route_by_number(sorted_routes[i])
17:          mate2 = get_route_by_number(sorted_routes[j])
18:          if random.random() <= cross_rate then
19:               child = cross_over(mate1, mate2)
20:               if random.random() <= mutate_rate then
21:                    child = mutation(child)
22:               child = repair(child)
23:               new_generation.append(child)
24:     C.clear()
25:     C.update(new_generation)
26:     C.add(get_route_by_number(sorted_routes[0]))
27:     gen += 1
28: best_chromosome = C[0] # end while
29: for chromosome in C[1:] do
30:     if tmd(chromosome) < tmd(best_chromosome) then
31:          best_chromosome = chromosome
32: return best_chromosome
```

3.3.1.3 Point-to-Point Navigation Planning (A* Algorithm)

A* Algorithm is an informed search algorithm that efficiently finds the shortest path from a starting point to a goal location on a graph or grid while considering the cost of each step and a heuristic function that estimates the remaining cost to reach the goal. Sharma et al. (2012) provides the pseudocode of the A* algorithm as follows.

```
Input: A graph G(V,E) with source node start and goal node end.
Output: Least cost path from start to end.
Steps:
Initialise
        open_list = { start }                       /* Listof nodes to be traversed*/
        closed_list = { }                           /*List of already traversed nodes*/
        g(start) = 0                                /*Cost from source node  to a node*/
        h(start) = heuristic_function (start,end)   /*Estimated cost from node to goal node*/
        f(start) = g(start)+h(start)                /* Total cost from source to goal node*/

while open_list is not empty
        m = Node on top of open_list, with least f
        if m = = end
                return
        remove m from open _list
        add m to closed_list
        for each n in child(m)
                if n in closed_list
                        continue
                cost= g(m)+ distance(m,n)
                if n in open_list and cost < g (n)
                        remove n from open_list as new path is better
                if n in closed_list and cost < g(n)
                        remove n from closed_list
                if n not in open_list and n not in closed_list
                        add n to open_list
                        g(n) = cost
                        h(n) = heuristic_function (n,end)
                        f(n) = g(n)+h(n)
return failure
```

Fig 3.3.1.3. A search algorithm pseudocode*

In this algorithm, the robot starts from the current position and continues until reaching the determined position given by HGA. By the A* algorithm, we can only move the robot by one step and go to the adjacent units in a specific direction. When the robot's LiDAR rotates, the robot can evaluate the fitness function for all 4 directions (forward, right, backward, left). Then, the A-star algorithm selects the largest fitness (f-value) that belongs to a specific direction and the robot moves toward the adjacent cell

along this specific direction. The f-values for these directions are calculated using the following formula: $f(n) = g(n) + h(n)$.

According to the above equation, the h(n) is the cost-to-go, which assumes the fitness function value of the robot's current position in a specific direction. g(n) is the cost-thus-far, which is the cost from its current position to the next position. There are two lists in this algorithm; namely, Open and Close lists. All the acceptable directions of the robot, which have a specific fitness function value, are stored in the Open list and then sorted based on the Max-heap. When each direction is added to the Open list, the list is reordered based on the biggest f-value and therefore, the top of the list refers to the biggest f-value. The selected direction with the biggest f-value pops up from the Open list and is put in the Close list. Then, the algorithm selects a state from the neighbour of the current state of the robot and guides the robot to move to the state with the best fitness function value.

In the algorithm, zero is set for all states cost-thus-far (g(s)) to the goal first. In the next step, the current location of the robot is put in the Open list. Then, the algorithm executes several instructions within a specific time in a loop. Each time in the loop, the algorithm deletes the maximum f-value state from the Open list and stores it in the Close list. In this step, the algorithm obtains the possible actions for the robot to move toward its neighbour's (children's) states from deleted states. After that, if all the possible child states do not exist in the Open list then they will be stored in the Open List, and also their parent states stored in the other list (tree) is stored.

3.3.1.4 Dynamic Obstacle Detection and Avoidance (Kalman Filter Algorithm)

According to Amir (2023), the Kalman filter is a mathematical algorithm that uses a series of measurements observed over time, containing noise (random variations) and other inaccuracies, and produces estimates of unknown variables that tend to be more precise than those based on a single measurement alone. It is a type of recursive Bayesian filter that is widely used in control systems, navigation systems, and other applications where precise estimates of unknown variables are needed. Visually, the Kalman filter can be thought of as a process of estimating the state of a system at each time step, where the state is represented by a set of variables such as position, velocity, and acceleration. The process begins with an initial estimate of the state, and at each time step, the filter uses a combination of a model of the system's dynamics and the new measurement data to update the estimate of the state.

The Kalman filter algorithm consists of two main steps: prediction and correction. The prediction step uses the model of the system's dynamics to predict the state of the system at the next time step based on the current estimate of the state. The correction step then uses the new measurement data to update the estimate of the state, taking into account the prediction and the measurement noise. The key idea behind the Kalman filter is that it uses a combination of the model of the system's dynamics and the new measurement data to produce a more accurate estimate of the state of the system than would be possible using either the model or the measurements alone. The filter also takes into account the uncertainty (or noise) in both the model and the measurements and uses this information to adjust the estimate of the state over time. Below is the flowchart for the algorithm.

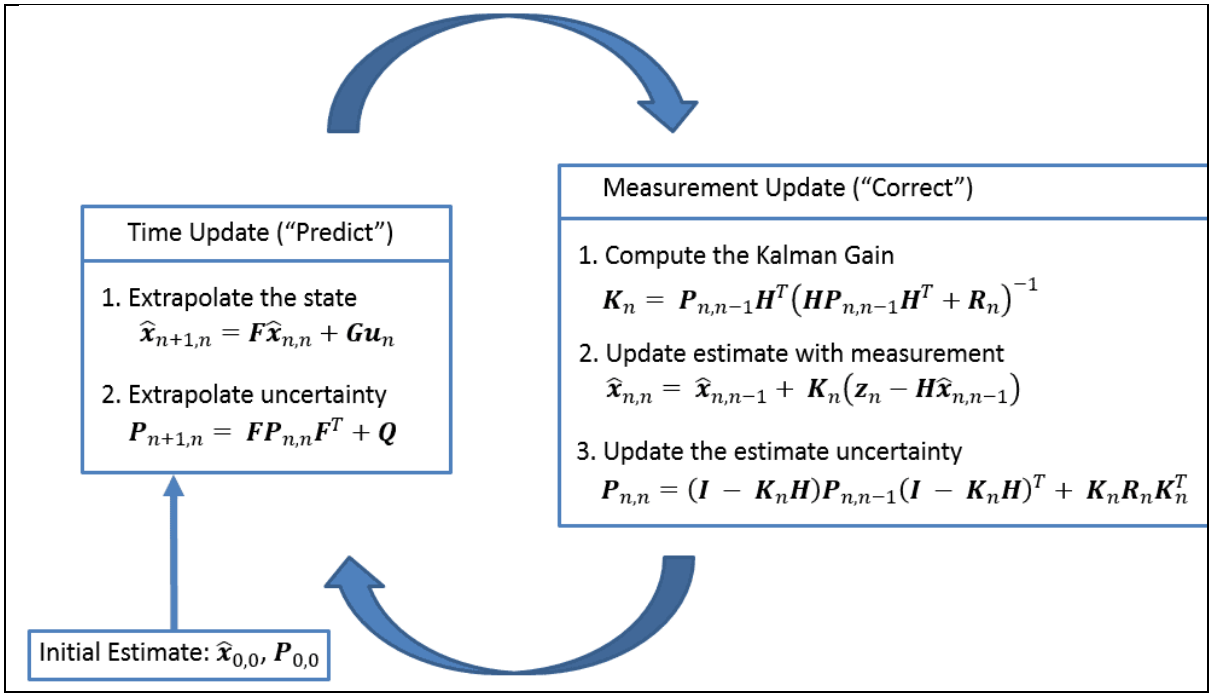


Fig 3.3.1.4. Flowchart of Kalman filter

Laaraiedh (2012) writes the Python code of the above process as follows. In our context, we shall define x as the robot's position and P as the robot's velocity.

```
def kf_predict(X, P, A, Q, B, U):
    X = dot(A, X) + dot(B, U)
    P = dot(A, dot(P, A.T)) + Q
    return(X,P)
def kf_update(X, P, Y, H, R):
    IM = dot(H, X)
    IS = R + dot(H, dot(P, H.T))
    K = dot(P, dot(H.T, inv(IS)))
    X = X + dot(K, (Y-IM))
    P = P - dot(K, dot(IS, K.T))
    LH = gauss_pdf(Y, IM, IS)
    return (X,P,K,IM,IS,LH)
def gauss_pdf(X, M, S):
    if M.shape()[1] == 1:
        DX = X - tile(M, X.shape()[1])
        E = 0.5 * sum(DX * (dot(inv(S), DX)), axis=0)
        E = E + 0.5 * M.shape()[0] * log(2 * pi) + 0.5 * log(det(S))
        P = exp(-E)
    elif X.shape()[1] == 1:
        DX = tile(X, M.shape()[1])- M
        E = 0.5 * sum(DX * (dot(inv(S), DX)), axis=0)
        E = E + 0.5 * M.shape()[0] * log(2 * pi) + 0.5 * log(det(S))
```

```
        P = exp(-E)
    else:
        DX = X-M
        E = 0.5 * dot(DX.T, dot(inv(S), DX))
        E = E + 0.5 * M.shape()[0] * log(2 * pi) + 0.5 * log(det(S))
        P = exp(-E)
    return (P[0],E[0])
```

In the above code, the following notations and their meanings are used:

X : The mean state estimate of the previous step (k −1).

P : The state covariance of the previous step (k −1).

A : The transition n x n matrix.

Q : The process noise covariance matrix.

B : The input effect matrix.

U : The control input.

K : the Kalman Gain matrix

IM : the Mean of predictive distribution of Y

IS : the Covariance or predictive mean of Y

LH : the Predictive probability (likelihood) of measurement which is computed using the Python function gauss_pdf.

3.3.2 WebSocket Server for Real-Time Communication

WebSocket is a computer communications protocol that provides simultaneous two-way communication channels over a single Transmission Control Protocol (TCP) connection. WebSocket is distinct from HTTP used to serve most webpages. Although they are different, RFC 6455 states that "WebSocket is designed to work over HTTP ports 443 and 80 as well as to support HTTP proxies and intermediaries", thus making it compatible with HTTP (Fette and Melkinov, 2011). To achieve compatibility, the WebSocket handshake uses the HTTP Upgrade header to change from the HTTP protocol to the WebSocket protocol.

The WebSocket protocol enables full-duplex interaction between a web browser (or other client application) and a web server with lower overhead than half-duplex alternatives such as HTTP polling, facilitating real-time data transfer from and to the server. This is made possible by providing a standardized way for the server to send content to the client without being first requested by the client and allowing messages to be passed back and forth while keeping the connection open. In this way, a two-way ongoing conversation can take place between the client and the server. The communications are usually done over TCP port number 443 (or 80 in the case of unsecured connections), which is beneficial for environments that block non-web Internet connections using a firewall. Additionally, WebSocket enables streams of messages on top of TCP. TCP alone deals with streams of bytes with no inherent concept of a message. The display of a flowchart of how WebSocket works is below.

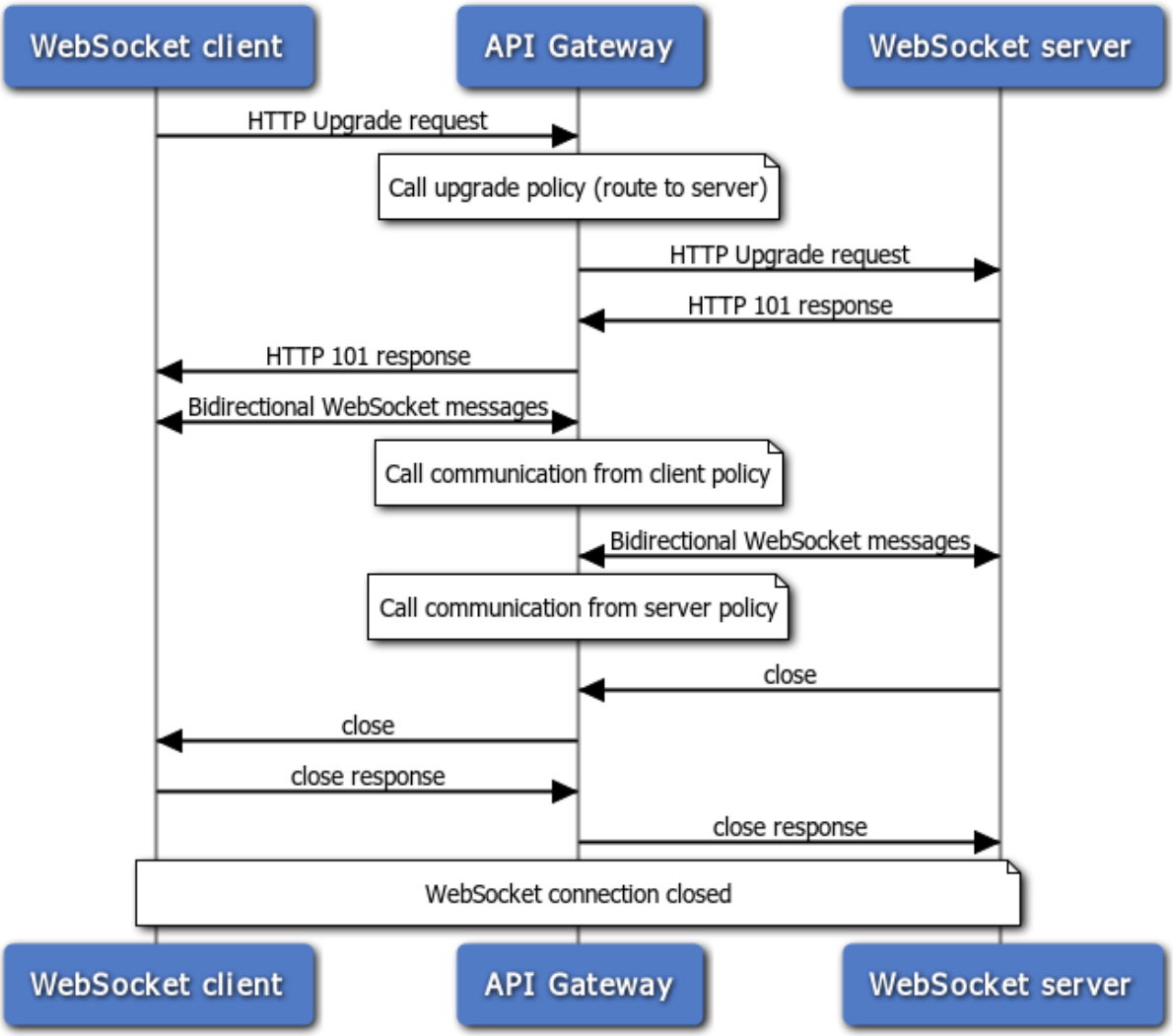


Fig 3.3.2. Flowchart of WebSocket mechanism

In our case, the robot's RPi would be the server and the mobile application would be the client. Since we are using RPi and Python to code, the programming language used would be Tornado. A sample code can be seen here.

3.3.2.1 HTTP Calls and Requests

Twinn (2022) provides the following flowchart for HTTP Calls and Requests within the host computer and Raspberry Pi.

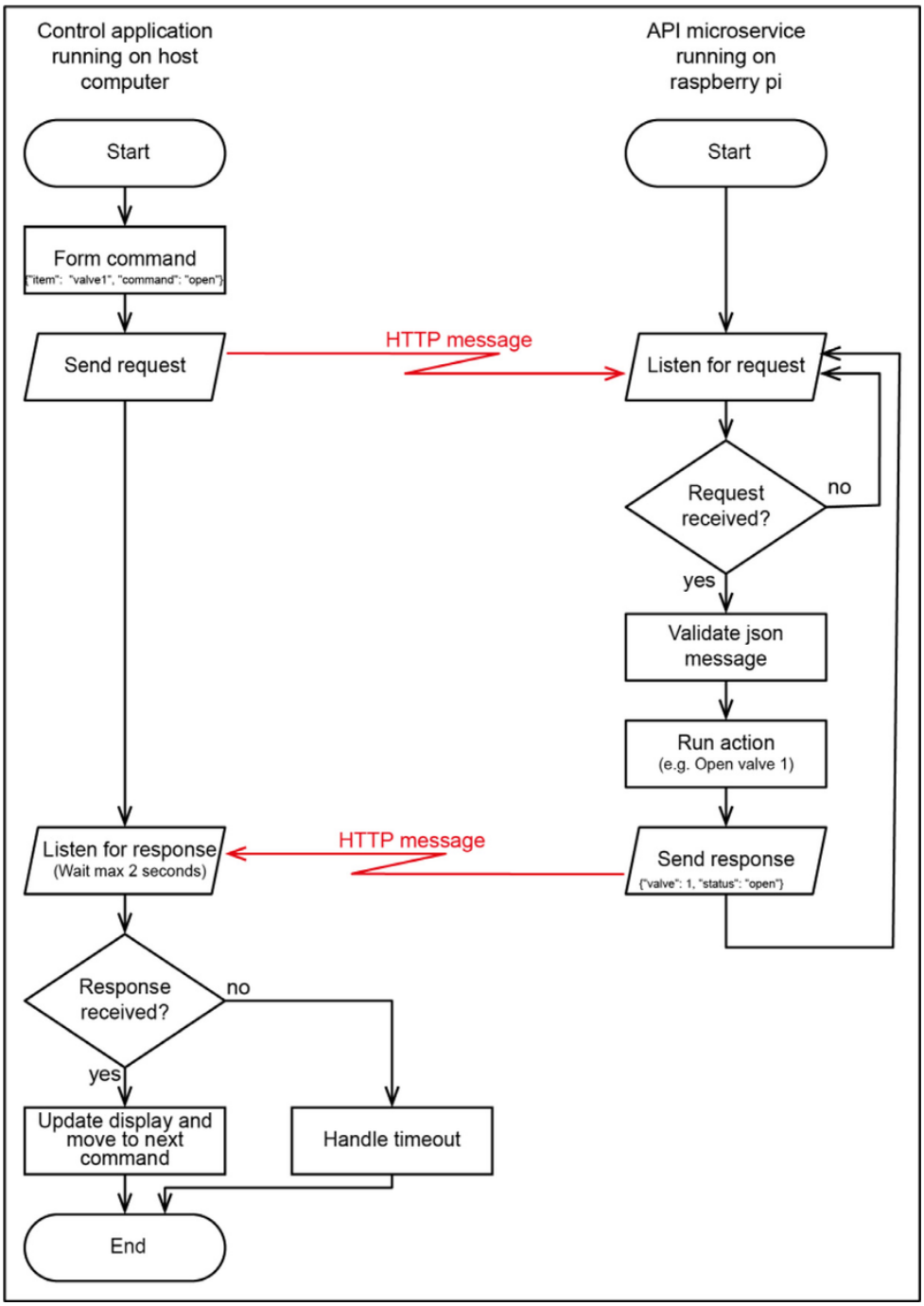


Fig 3.3.2.1. Flowchart of HTTP Calls and Requests between client and server

After the two sides have received an HTTP Call from each other, the WebSocket handshake occurs.

3.4.1 Mechanical Design

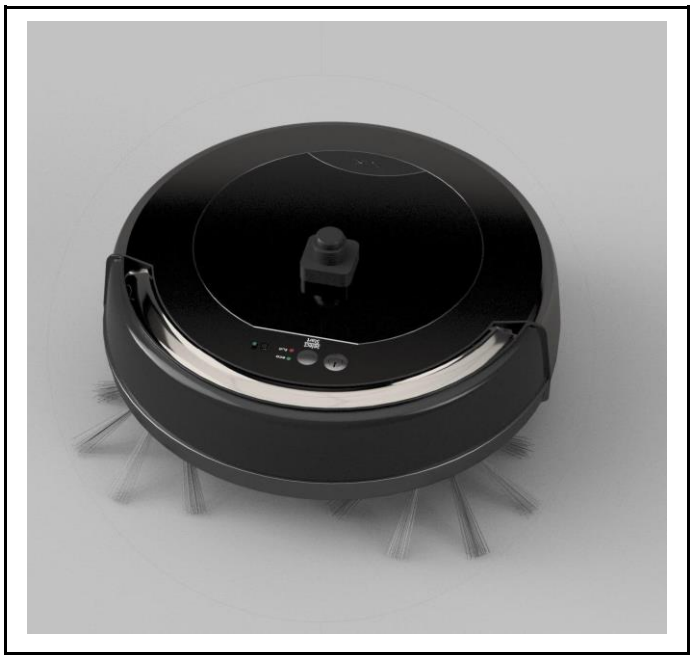


Fig 3.4.1.1 Higher Level Mechanical Design

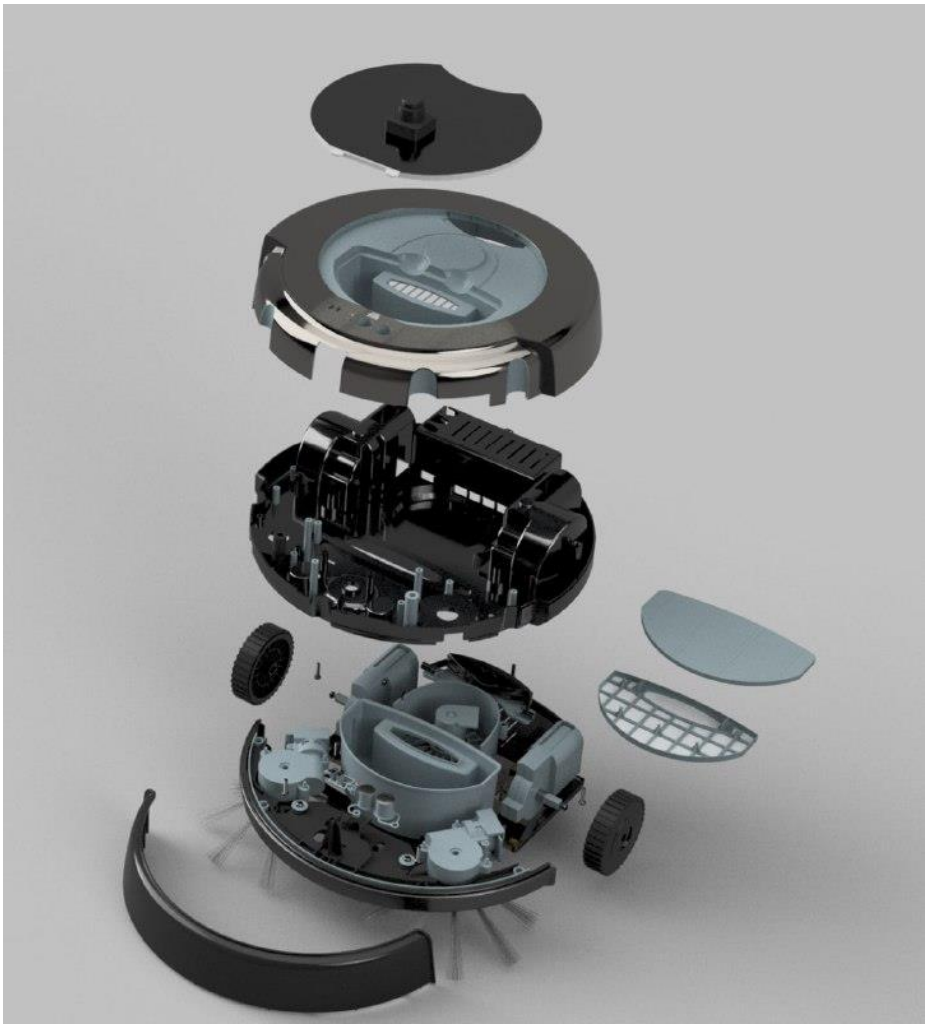


Fig 3.4.1.2 Higher Level Mechanical Design in Details

3D Drawing Reference:
CAD : https://grabcad.com/library/robotic-vacuum-cleaner-modified-roomba-1
Video Illustration : https://irobot.com.sg/product/irobot-roomba-s9-plus/

3.4.2 Electrical Circuit Schematics

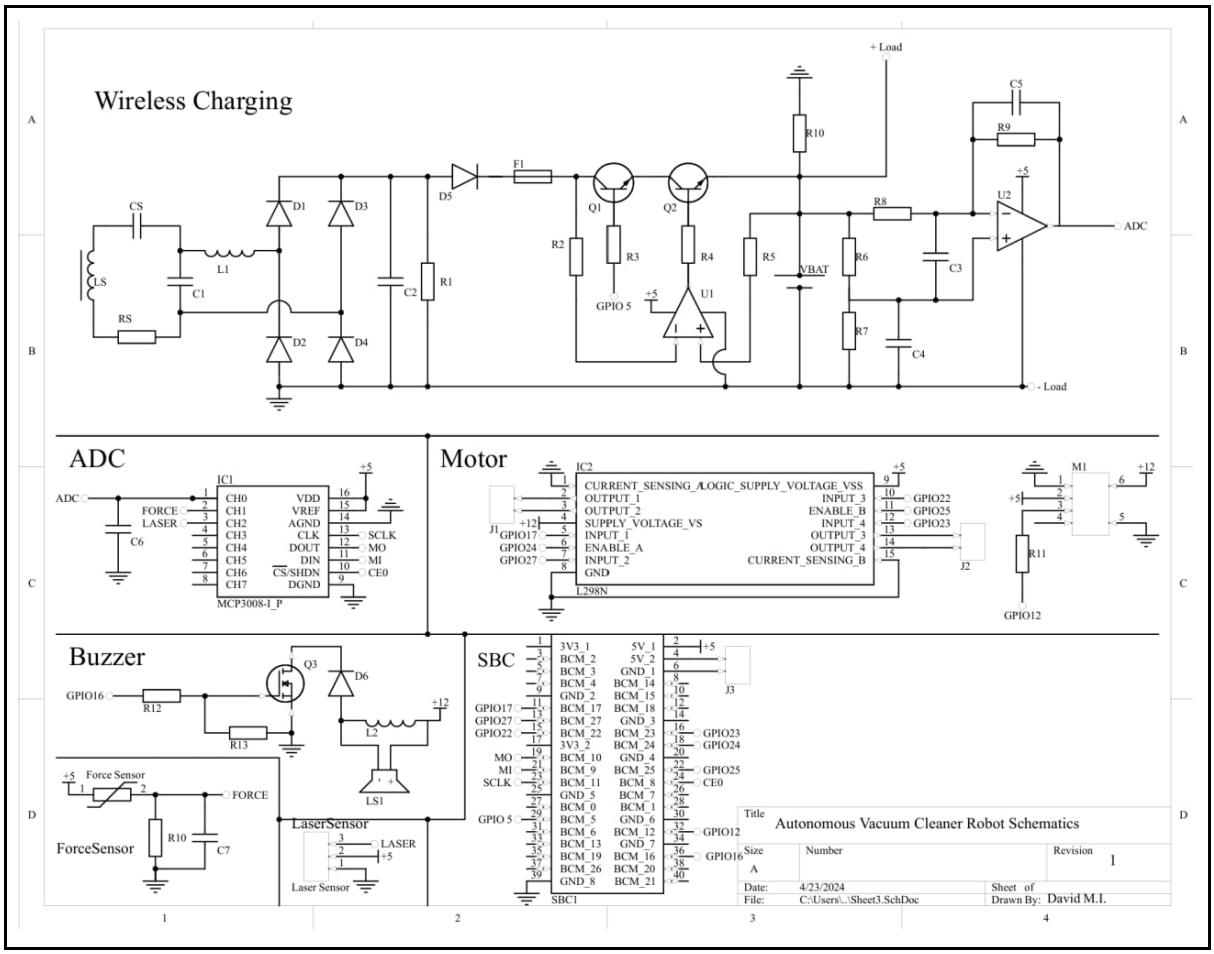


Fig 3.4.2.1. Robot Electrical Schematics

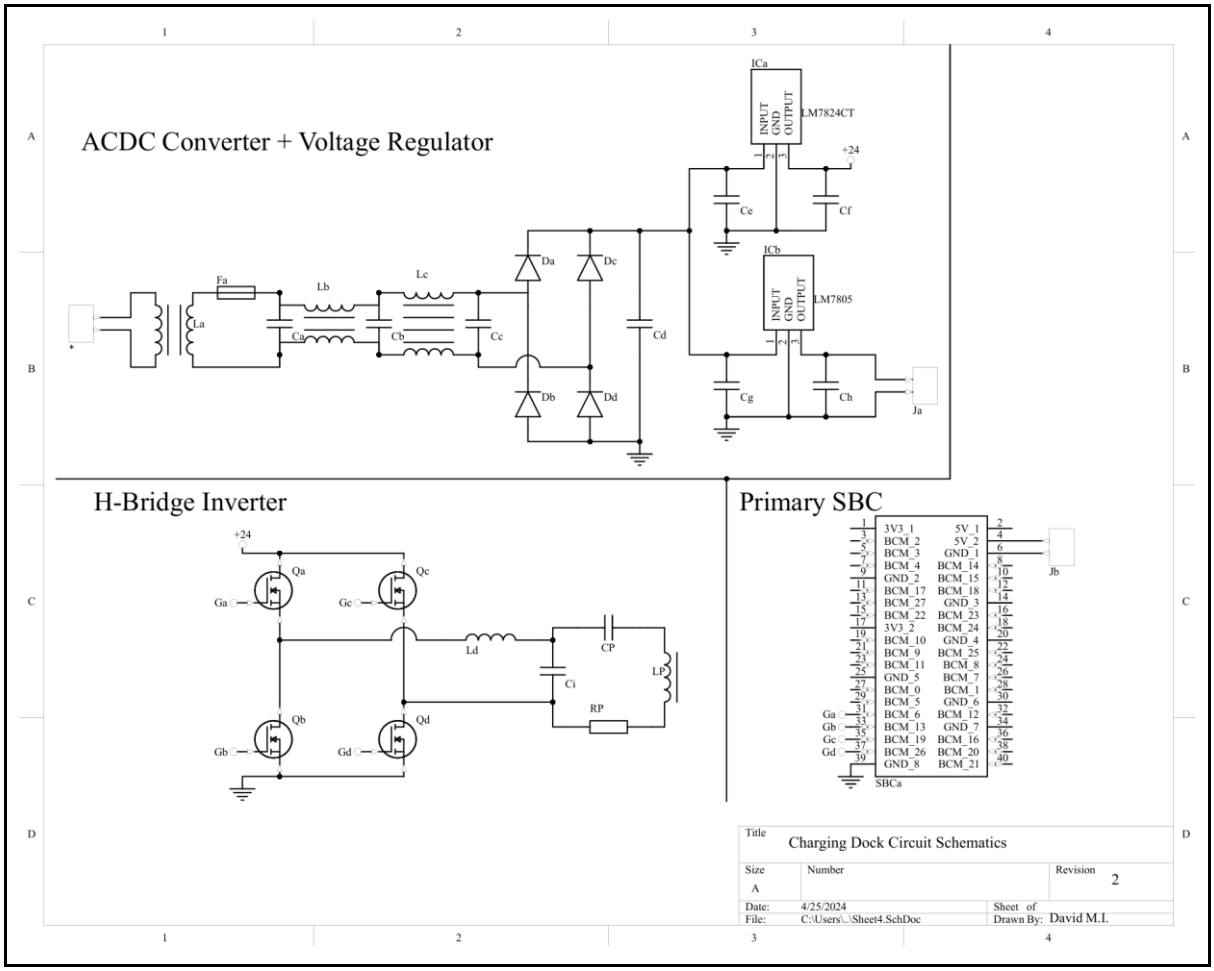


Fig 3.4.2.2. Charging Dock Electrical Schematics

3.5 SIZING CALCULATIONS

3.5.1 Power Consumption Table

Component	Model Number	Datasheet	Quantity	Voltage (V)	Current (A)	Power (W)
Raspberry Pi 4B	SBC006	Link	1	5.0	0.77	3.85
OpenCR 1.0	-	Link	1	14.4	0.25	3.60
Dynamixel-X Motor	-	Link	2	14.4	1.40	40.32
Motor Driver	L298N	Link	1	12.0	0.036	0.432
LiDAR SF20	-	Link	1	5.0	0.1	0.50
MCP3008 Analog to Digital Converter	MCP3008-I/P	Link	1	5.0	5 x 10^(-4)	0.0025
Force Resistive Sensor	SEN-09376	Link	1	5.0	5 x 10^(-4)	0.0025
Laser Sensor	KEYES	Link	3	3.3	0.025	0.2475
12V DC Motor RS555	RS-555SA-2755	Link	2	10.0	1.20	24.00
12V DC Universal Motor	HCDM2055	Link	1	12.0	6.7	80.4
12V DC Suction Motor Neato D Ser.	BCB1012UH-A	Link	1	12.0	3.45	41.4
12V DC Piezo Transducer	MCKPT-G1720-3922	Link	1	12.0	0.03	0.36
Total Power Required						195.115

Fig. 3.5.1 Higher Level Power Consumption Table

3.5.2 Sizing Calculation

3.5.2.1 Calculation for Battery

The following calculations follow steps in 2.7.2. According to *EdgeProp*, the median of home area in Singapore is 86.0 m^2, which is used to benchmark the robot's total working area. A 14.4V DC battery is utilised with a capacity of 5200 mAh. We want the robot to at least survive for 12 minutes of cleaning, which should be enough to navigate to the room and clean the room. The available energy to be used for the robot, assuming 80% efficiency and leaving 20% remainder for the capacity of the battery, is 0.8 * 0.8 * 14.4 * 5.2 = 47.923 Wh. The total time is therefore Available Energy / Required Power =

47.923 / 195.115 = **14.7** minutes, which surpasses what we expected. Theoretically, it should be longer than 15 minutes as we assume maximum current usage for the motors.

3.5.2.2 Calculation for Suction Motor

The rationale behind choosing a suction motor is that enough suction power should be considered. The measurement of suction power is Air Watts (*AW*), which has the following conversion to power (*W*).

$$P = AW \times 0.9983$$

According to vacuum cleaner company *Dyson*, 10 *AW* is the industry average for an automated vacuum cleaner. As such, it is assumed that the current design has a 10*AW*, its corresponding motor power requirement for the motor could be calculated as below.

$$P = 10\ AW \times 0.9983 = 9.983\ W$$

As the chosen suction motor has a maximum power of 41.4 W, the motor has enough suction power to complete the cleaning task.

3.5.2.3 Calculation for Wheel Motors

As suggested in 3.5.2.1, the robot is expected to complete 86.0 m^2 in 14.7 min, which indicates the expected average speed.

$$Expected\ Area\ Clearance\ Speed = \frac{86.0}{14.7 \times 60}$$
$$= 0.0975\ m^2/s \text{ (to 3 s.f.)}$$

This robot is designed to have a dimension of 50 * 50 *16 cm, the linear speed of the robot when assuming performing a perfect straight snake shape path is calculated below.

$$Expected\ Linear\ Speed = \frac{0.0975}{0.500}$$
$$= 0.0195\ m/s \text{ (to 3 s.f.)}$$

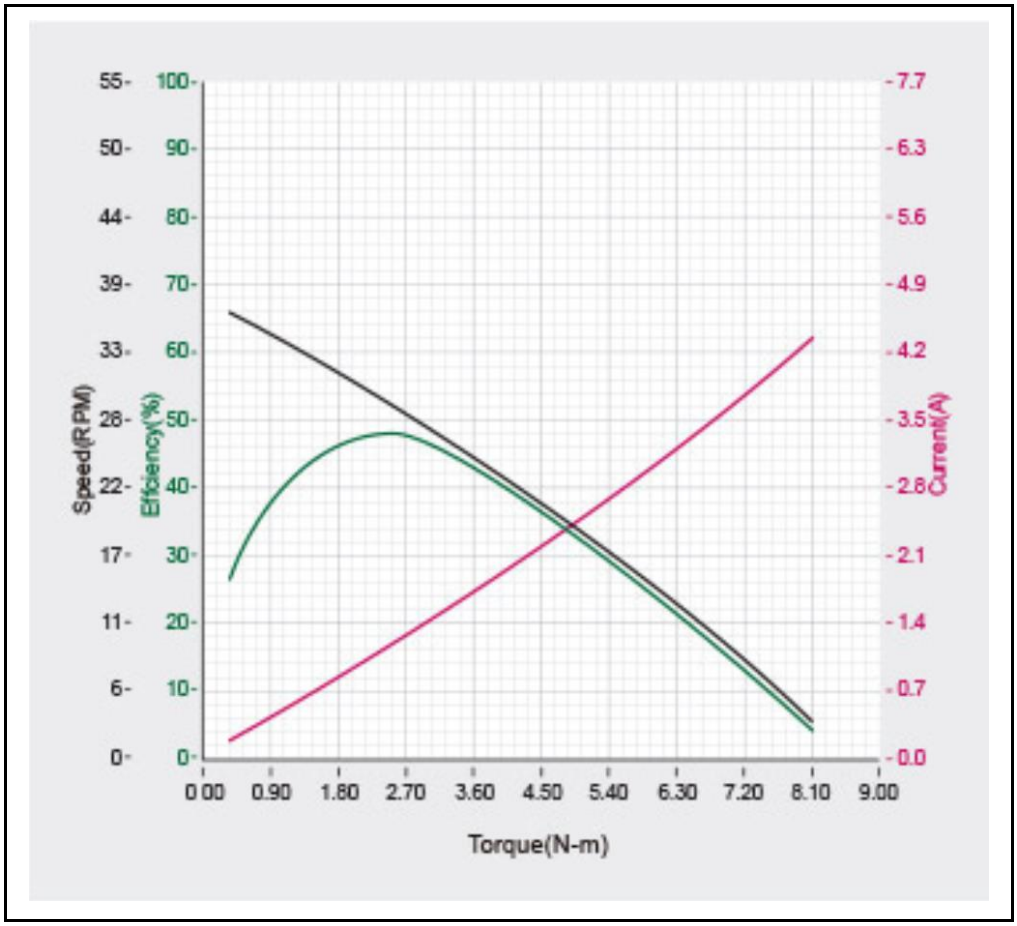


Fig 3.5.2.3 Dynamixel-X Motor Performance Graph

Referring to the Dynamixel-X Motor Performance Graph (*Fig 3.5.2.3)*, a wheel motor (Dynamixel-X Motor) has a maximum current of 1.40 A, and it runs at 20 RPM. Our choice of wheel is with a rough dimension of 4.50 * 4.50 * 2.00 cm, which indicates the circumference of the wheel can be calculated as below.

$$Circumference = 2 \times \pi \times 0.045 = 0.283\, m$$

As such, the theoretical linear speed can be calculated as below.

$$Theoretical\ Linear\ Speed = \frac{20 \times 0.283}{60} = 0.0943\, m/s$$

As shown above, the theoretical speed under maximum power condition surpasses the required speed, giving room for further speed adjustment to the motor.

3.5.2.4 Calculation for Wireless Charger

a. Charging Dock

We plan to step down the power grid voltage from 230VAC 50Hz to 30VAC using a toroidal transformer 1182K15. This secondary voltage would then be rectified by the Schottky diode DFLS230LQ to around 29V DC. With a voltage of 29V DC and a current of ~1.5 A, we will separate this voltage into 2 different voltage regulators. First, is to LM7824 to generate a stable 24V DC output to then be used as powering the primary or TX coil. The second is to LM7805 to generate a stable 5V DC output to then be used to power the Single Board Computer (SBC) or, in our case, RPi. Now, for the first filtering of the transformer's output voltage, we plan to use 2 low-pass LC filters connected in series with each other and followed by a capacitor to smoothen the power output and further reduce the voltage ripple. More specifically, Lb is a *differential mode choke* and Lc is a *common mode choke*. A fuse is installed to avoid overcurrent.

The set up for the TX (orange) and RX (blue) coils is as follows.

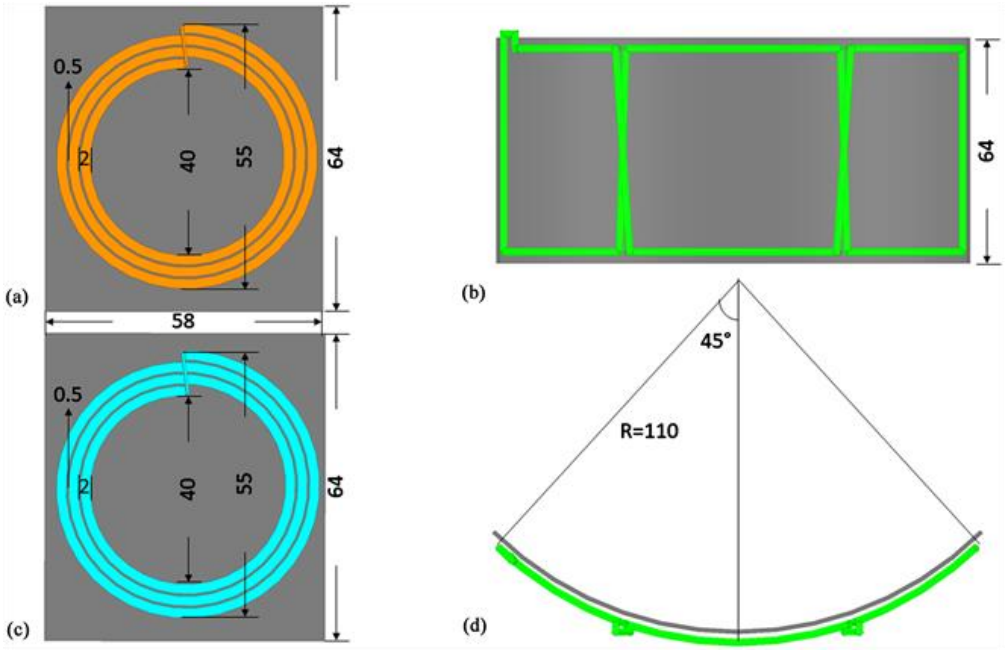


Fig 3.5.2.4.1. Dimensions of TX/RX coils and their placement

b. Robot

While the secondary coil voltage is lower than the primary coil voltage due to magnetic flux losses, we expect to see around 15V DC for charging the battery. This is made possible as we have 24V DC in the primary TX coils and we set the distance between the TX coils and the robot's RX coil to be ~2-3 cm, which is quite close so that most of the magnetic flux lines still come through the RX coil. See the figure below for illustration.

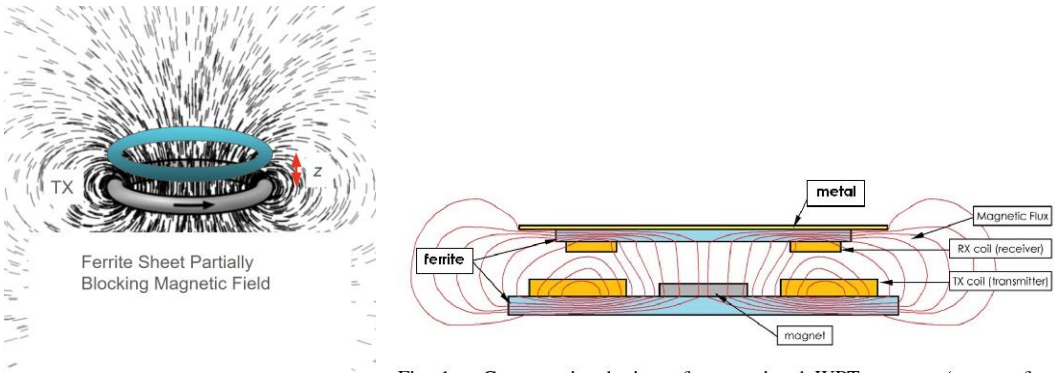


Fig 3.5.2.4.2. (Left) Illustration of low magnetic flux losses if distance z is small. (Right) Park (2015) illustrates how the ferrite sheet acts as a magnetic shield.

It initially covers most of the magnetic field line, but then due to electromagnetic induction, the secondary coil would also have current, which deviates the original magnetic field lines. Nevertheless, the secondary coil voltage there would reach 16V AC or even more. This voltage would then be rectified and reach around 15V DC after moving through the rectifier. Another Schottky diode is needed to prevent back current flowing from the battery to the secondary coil. We also implement an AND logic gate to let the coil charge the battery. The first input signal to the BJT is whether the rectified and filtered coil voltage (later called as cleaned voltage) is larger than the current battery voltage. If the coil voltage is somehow lower than the battery voltage, of course the battery would discharge rather than charge. We achieve this by implementing an operational amplifier with cleaned voltage connected to negative (-) pin and battery voltage connected to positive (+) pin. With battery voltage < cleaned voltage, the first condition is passed and the second condition arises, which is whether the battery is fully charged or not. To check this, we implement a voltage sensing amplifier to detect the voltage of the battery. The calculation is as follows.

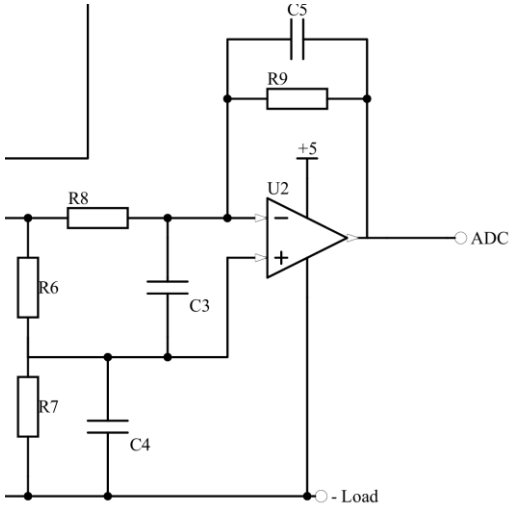


Fig 3.5.2.4.3. Second Condition Circuit

The output voltage is calculated as $V_{out} = \frac{R_9}{R_8}\left(\frac{R_7(R_8+R_9)}{R_9(R_6+R_7)} - 1\right)V_{battery}$. We want this voltage to be less than or equal to 5V as we want to read the voltage compared to 5V (VREF). If we set these resistor values such that Vt < Vout < 5 V for some threshold voltage (to be determined in the code), then the second condition is satisfied. The cleaned voltage would then charge the battery with a voltage of around 14.5 V DC.

It is noteworthy to mention several design limitations of the overall design processes and leave some final remarks.

LiDAR
In terms of navigation, LiDAR is utilised to enhance the overall navigation quality. Due to its intrinsic characteristics, LiDAR could only capture accurate and non-overlapping data points when the cleaner robot runs at a relatively low speed. Running at a low-speed poses issues such as the robot not being able to complete cleaning the entire parameter within its battery capacity.

To remedy this issue, the microcontroller OpenCR is introduced. Its advantage is its on-board IMU (Inertial Measurement Unit), which provides data such as spatial orientation and acceleration. It also hugely enhances LiDAR's data point cloud consistency and accuracy.

Motor Speed
The calculation in 3.5.2.3 is conducted without considering the motor's mechanical drag, resistance, back Electromagnetic Force and magnetic saturation. As such, further investigations are required which bring in factors such as the voltage constant (K_e), torque coefficient (K_t) and friction coefficient (K_f) to make a better runtime estimation.

Selection of SBC
After evaluating costs and performance of various Single-Board Computers, Raspberry 4B is chosen due to its competency and usability. However, it does not possess analog input pins to receive analog data from sensors. As such, an ADC (Analog to Digital Converter) MCP3008 is introduced which stands between the SBC and various sensors. Moreover, to ensure fast and accurate calculations for advanced algorithms mentioned in previous sections, a more state-of-the-art SBC such as Raspberry 5 should be considered.

Battery capacity
Integrating LiDAR and IMU can lead to a faster battery drain. Both technologies require a high level of power consumption. LiDAR emits laser pulses continuously to measure distances while IMUs need power for their sensors to track motion and orientation of the robot. This increased power consumption means that the device's battery may deplete faster than anticipated.

IMU inaccuracy
IMUs can be inaccurate due to drift and noise, particularly if the robot is operating for a long period of time. The small inaccuracies in measuring angular velocities and accelerations gradually causes the drift which is the error in the reported position or orientation of the robot. Noise, on the other hand, refers to random fluctuations that can obscure the real motion signals. These factors can degrade the navigation accuracy of the robot, making it less efficient or unable to operate by itself without human correction.

Full Dust Bag Warning Mechanism Consideration
It has been considered that the force sensor to warn the user about a full dust bag may not be entirely efficient. This is because during a scenario when the differently-shaped dusts accumulate unevenly and peak at one area, which will trigger the force sensor even when the bag is not full. A suggested

improvement is to utilise multiple laser sensors to measure if the height of the dust exceeds a certain threshold. Notably, several laser sensors measuring from different angles should be utilised to ensure the alarm is not triggered when the bag is not yet full.

Version Control Management

Version control with Git and GitHub was overlooked during the prototyping phase, yet it plays a crucial role in managing code changes. It's essential to leverage these tools to keep track of modifications, particularly when numerous detailed parameters are adjusted at various stages of development of the mBot. By maintaining an organized repository on GitHub, developers can effectively collaborate, track changes, and ensure the integrity and stability of the codebase throughout the prototyping process.

Mopping Function

One important feature of a vacuum cleaner not implemented is the mopping function. The higher level could carry a water tank, coupled with a solenoid valve which controls the dripping of the liquid from the tank. The charging stand could also add the function to add water each time the robot comes back to charge.

Components	Price	Quantity	Total cost
RaspberryPi 4B	$92.00 (Link)	1	$92.00
RaspberryPi 4B Case	$24.56 (Link)	1	$24.56
SanDisk Ultra microSDXC, SQUAB 64GB Card	$10.80 (Link)	1	$10.80
OpenCR 1.0	$290.84 (Link)	1	$290.84
Dynamixel-X Motor	$59.07 (Link)	2	$118.14
LiDAR SF20	$539.95 (Link)	1	$539.95
MCP3008 Analog to DC Converter	$4.46 (Link)	1	$4.46
Force Resistive Sensor SEN-09376	$17.88 (Link)	1	$17.88
Laser Sensor Keyes	$5.99 (Link)	3	$17.97
Motor Driver L298N	$1.75 (Link)	1	$1.75
12V DC Motor RS555	$25.36 (Link)	2	$50.72
12V DC Universal Motor HCDM2055	$42.14 (Link)	1	$42.14
12V DC Suction Motor Neato D Ser.	$41.94 (Link)	1	$41.94
5200mAh Xiaomi Robot Vacuum Cleaner Battery Pack	$52.93 (Link)	1	$52.93
Mechanical Part Estimation (Fan, Joints, Wheels, etc.)	$120.00	-	$120.00
Charger Dock	$75.00	1	$75.00
12V DC Piezo Transducer	$0.689 (Link)	1	$0.689
Neato Hepa Filter	$5.66 (Link)	1	$5.66
Total Cost			$1507.429

Table 3.7. Cost estimation breakdown

4. CONCLUSION

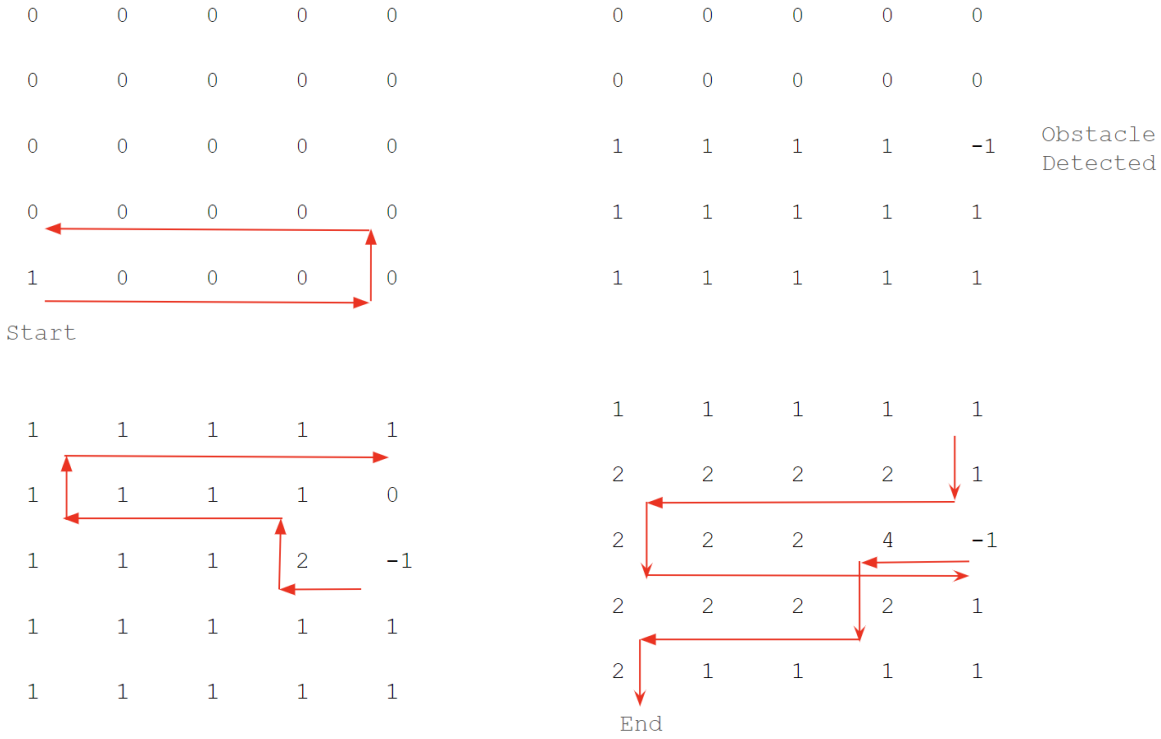
In conclusion, Project Saturn robotic vacuum cleaner represents an innovative solution of cleaning technology for residential uses. Through an iterative design and testing process, we have successfully integrated sophisticated technologies such as LiDAR and IMUs which effectively enhance the robot's navigational capabilities and operational efficiency. Furthermore, we incorporated complex algorithms for path planning and obstacle avoidance. In addition, we also deployed a custom-designed mechanical setup and power management system. These innovative solutions demonstrate our ability to utilise technologies to solve real life problems.

Despite facing challenges such as rapid battery depletion, IMU data drift, and the need for high processing power, we attempted to mitigate these issues through careful selection of components and continuous optimisation of system efficiency. Moreover, we conducted a detailed analysis of component selection, power consumption and system integration. This showcases our commitment to creating a reliable and user-friendly product. As we move on to the next phase of development, we aim to further enhance the Project Saturn platform. Our journey doesn't end here. This marks the starting point of a continuous evolution process towards creating a smarter, more efficient, and user-centric product.

5. APPENDIX

5.1 FIRST ATTEMPT TOWARD AUTONOMOUS NAVIGATION FOR PROTOTYPE

Initially, a back-and-forth snake algorithm to clean a room with a randomly placed obstacle is proposed. In order to achieve the zig-zag path, the occupancy mapping technique is deployed in order to memorise how many times the robot has been in a cell. The 50 cm x 50 cm area is divided into several cells with adjustable sizes. At the start, all cells that have not been passed by the robot are marked by 0. When the robot is on a cell and detects no obstacle, it will increment the value of the cell by 1. If the robot detects an obstacle, it will mark the value of the cell by -1. In the end, the grid is expected to be filled by -1, 0, and positive integers, or any possible combination of the three. To illustrate the mechanism, here is an example of a 5 x 5 map. Notably, the robot motion as well as the sensor integration are not specified in this section.



The algorithm is written in **Python** as follows:

```
def printOccupancyMap(occupancy_map):
    for row in occupancy_map:
        print(" ".join(f"{cell:2d}" for cell in row))

def cleanArea(occupancy_map):
    new_map = [row[:] for row in occupancy_map]
    row, col = len(new_map) - 1, 0  # Starting position of the
robot
    direction_x = 1 # 1 for moving right, -1 for moving left
    direction_y = 0 # 0 for moving horizontally, 1 for moving
upward, -1 for moving backward
    # Forward Phase
    while True:
        print(                          )
        printOccupancyMap(new_map)
        if row == 0 and col == len(new_map[0]) - 1:
            new_map[row][col] += 1
            row += 1
            direction_x *= -1
            break
        # Check if the current cell has an obstacle (-1) while
moving horizontally
        if new_map[row][col] == -1 and direction_y == 0:
            # If obstacle detected, move to left or right
            col -= direction_x
            new_map[row][col] += 1
            direction_y = 1
            row -= direction_y
            direction_x *= -1
       # Check if the current cell has an obstacle (-1) while
moving vertically
        if new_map[row][col] == -1 and direction_y == 1:
            # If obstacle detected, move down and go direction by
1 and then go up again
            col += direction_x
            new_map[row][col] += 1
        else:
            # If no obstacle, move to the right or left based on
direction
            new_map[row][col] += 1
            col += direction_x
            direction_y = 0
```

```
        # If the next cell is out of bounds, reverse the direction
and move up
        if col < 0 or col >= len(new_map[0]):
            direction_x *= -1
            direction_y = 1
            row -= direction_y
            col = max(0, min(len(new_map[0]) - 1, col))  # Ensure
col is within bounds
    # Return Phase
    while True:
        print(                          )
        printOccupancyMap(new_map)
        if row == len(new_map) - 1 and col == 0:
            new_map[len(new_map) - 1][0] += 1
            print(                            )
            printOccupancyMap(new_map)
            break
        # Check if the current cell has an obstacle (-1) while
moving horizontally
        if new_map[row][col] == -1 and direction_y == 0:
            # If obstacle detected, move to right or left
            col -= direction_x
            new_map[row][col] += 1
            direction_y = -1
            row -= direction_y
            direction_x *= -1
       # Check if the current cell has an obstacle (-1) while
     moving vertically
        if new_map[row][col] == -1 and direction_y == -1:
            # If obstacle detected, move up and go in horizontal
direction by 1 and then go down again
            col += direction_x
            new_map[row][col] += 1
        else:
            # If no obstacle, move to the right or left based on
direction
            new_map[row][col] += 1
            col += direction_x
            direction_y = 0

        # If the next cell is out of bounds, reverse the direction
and move down
        if (col < 0 or col >= len(new_map[0])):
            direction_x *= -1
            if row != len(new_map) - 1: # Ensure row is within
bounds
                direction_y = -1
```

```
            else:
                direction_y = 0
            row -= direction_y
            col = max(0, min(len(new_map[0]) - 1, col))  # Ensure
col is within bounds
```

5.2 2nd RENDITION OF CODING PROTOTYPE CODE

```
#include <MeMCore.h>
#include <SoftwareSerial.h>
#include <Wire.h>
#include <Orion.h>
#include <vector.h>
#include <cstdlib>

// Initiate Pins
MeRGBLed led1(PORT_3, SLOT1, 15);   /* parameter description: port,
slot, led number */
MeRGBLed led2(PORT_3, SLOT2, 15);
MeDCMotor motor3(M1);
MeDCMotor motor4(M2);
MeLineFollower lineFinder(PORT_2);
MeUltrasonicSensor ultraSensor(PORT_3);

// Initiate Constants
int lineSensorVal = 0;
int ultrasonicSensorVal = 0;
int tapeCount = 0; // Track how many times the mBot detects a black
tape
int positionX = 0; // Track the position of the mBot in x direction
unsigned long moveStartTime = 0;  // Variable to store the start time
of the x-direction movement
unsigned long xMoveDuration = 0; // Variable to see how long is it
moving in x direction
bool obstacleEncountered = false; // Initialize flag to false

// Declare the list (vector) of movement time in y direction globally

std::vector<int> Elements;

void setup()
{
    Elements yDuration;
    Serial.begin(9600);
    pinMode(8,OUTPUT);
    pinMode(12,INPUT);
```

```
    pinMode(LED_BUILTIN,OUTPUT);
}

void loop()// Main Running Code
{
    if (analogRead(A0)>560) // Force Sensor Board
    {
        stop();
        turnOnLED();
        delay(2000);
    }
    if (positionX == round(positionX) && checkAverageVsLastItem())
    //if either LHS and RHS is FALSE (0), lineFollow();both TRUE,
snake()
    {
        snake();
    }
    else
    {
        lineFollow();
    }
}

void turn()
{
    unsigned long turnStartTime = millis(); // Start timing the turn
    if (tapeCount % 2 == 1) //Turn Left if tapeCount is odd
    {
        motor3.run(100);
        motor4.run(100);
        delay(650);
        motor3.run(100);
        motor4.run(-100);
        delay(700);
        motor3.run(100);
        motor4.run(100);
    }
    else //Turn Right
    {
        motor3.run(-100);
        motor4.run(-100);
        delay(650);
        motor3.run(100);
        motor4.run(-100);
        delay(700);
        motor3.run(-100);
        motor4.run(-100);
    }
    while (millis() - turnStartTime < 1900)
    {
```

```
        lineSensorVal = lineFinder.readSensors();
            if (lineSensorVal == 0) // If tape is detected
            {
                xMoveDuration = millis() - turnStartTime - 1350;
                // Stop the turn
                motor3.stop();
                motor4.stop();
                // Update tape count
                tapeCount++;
                // Increment x-coordinate based on the duration
                positionX += xMoveDuration / 650;
                return;  // Exit the function
            }
    }
    delay(100);
}

void stop()
{
    motor3.run(0);
    motor4.run(0);
}

void forward()
{
    motor3.run(70);
    motor4.run(-70);
}

void backward()
{
    motor3.run(-80);
    motor4.run(80);
}

void turnOnLED()
{
  digitalWrite(LED_BUILTIN,HIGH);
  delay(1000);
  digitalWrite(LED_BUILTIN,LOW);
  delay(1000);
}

void snake()
{
    Elements yDuration;
    lineSensorVal = lineFinder.readSensors();
    ultrasonicSensorVal = ultraSensor.distanceCm();
    delay(10);
    forward();
```

```
    if (analogRead(A0) > 560)
    {
        stop();
        turnOnLED();
        delay(2000);
    }

    unsigned long forwardStartTime = millis(); // Start timing of
moving forward
    if (lineSensorVal == 0 && positionX == round(positionX) &&
checkAverageVsLastItem()) //If detect a tape and it is not the
horizontal tape
    {
        stop();
        delay(500);
        if (!obstacleEncountered && tapeCount % 2 == 0)
        {
            yDuration.push_back(millis()-forwardStartTime);
        }
        obstacleEncountered = false; // Reset the flag
        tapeCount++; //Increment Tape Count
        delay(500);
        turn();
        stop();
        delay(500);
    }
    else if (lineSensorVal == 0 && positionX != round(positionX) &&
(checkAverageVsLastItem() == false))
    {
        return; //Exit the function in order to start lineFollow()
function
    }
    if (ultrasonicSensorVal < 10)
    {
        stop();
        delay(500);
        avoidObstacle();
        obstacleEncountered = true; // Set the flag if obstacle
encountered
    }
}


// A function to turn with line checking to avoid the obstacle

bool turnWithLineCheck(int leftSpeed, int rightSpeed, int maxDuration)
{
    unsigned long startTime = millis();  // Store the start time of
the turn
```

```
    motor3.run(leftSpeed);
    motor4.run(rightSpeed);
    while (millis() - startTime < maxDuration) {
        delay(50);  // Delay for checking line sensor every 50
milliseconds
        lineSensorVal = lineFinder.readSensors();
        if (lineSensorVal == 0) {
            // Detected line, stop the turn and return true
            motor3.stop();
            motor4.stop();
            return true;
        }
    }
    // If no line is detected within the specified duration, stop the
turn and return false
    motor3.stop();
    motor4.stop();
    return false;
}

void avoidObstacle()
{
    if (analogRead(A0) > 560)
    {
        stop();
        turnOnLED();
        delay(2000);
    }
    if (tapeCount%2 == 0)
    {
        while (true)
        {
            // Move forward and check for line
            if (turnWithLineCheck(80,80,850)) {
                break;
            }
            // Turn left and check for line
            if (turnWithLineCheck(150, -150,600)) {
                break;
            }
            // Move backward and check for line
            if (turnWithLineCheck(-80,-80,850)) {
                break;
            }
            // Turn right and check for line
            if (turnWithLineCheck(120, -120,1300)) {
                break;
            }
            // Move backward and check for line
            if (turnWithLineCheck(-80,-80,850)) {
```

```
                break;
            }
            // Turn left and check for line
            if (turnWithLineCheck(80,-80,600)) {
                break;
            }
            // Move forward and check for line
            if (turnWithLineCheck(80,80,900)) {
                break;
            }
        }
    }
    else
    {
        while (true)
        {
            // Move backward and check for line
            if (turnWithLineCheck(-80,-80,850)) {
                break;
            }
            // Turn left and check for line
            if (turnWithLineCheck(150, -150,600)) {
                break;
            }
            // Move forward and check for line
            if (turnWithLineCheck(80,80,850)) {
                break;
            }
            // Turn right and check for line
            if (turnWithLineCheck(120, -120,1300)) {
                break;
            }
            // Move forward and check for line
            if (turnWithLineCheck(80,80,850)) {
                break;
            }
            // Turn left and check for line
            if (turnWithLineCheck(80,-80,600)) {
                break;
            }
            // Move backward and check for line
            if (turnWithLineCheck(-80,-80,900)) {
                break;
            }
        }
    }
}

bool checkAverageVsLastItem()
{
```

```
    Elements yDuration;
    if (yDuration.size() < 2) {
        // Not enough items to compare, return
        return true;
    }
    // Calculate the sum of the first n-1 items
    unsigned long sum = 0;
    for (int i = 0; i < yDuration.size() - 1; ++i) {
        sum += yDuration[i];
    }
    unsigned long average = sum / (yDuration.size() - 1); // Calculate
the average of the first n-1 items
    unsigned long lastItem = yDuration[yDuration.size() - 1]; // Get
the last item
    unsigned long threshold = 500;
    // Compare the average with the last item
    if (abs(average - lastItem) > threshold)
    {
        return false;
    }
    else
    {
        return true;
    }
}


void lineFollow()
{
    while (true)
    {
        int dock = digitalRead(12);
        if (dock == 0)
        {
            // Docking condition met, stop and break out of the loop
            stop();
            break;
        }

        lineSensorVal = lineFinder.readSensors();

        // Adjust direction based on sensor readings
        if (lineSensorVal == 0)
        {
            // Both sensors on the line, move forward
            motor3.run(100);
            motor4.run(100);
        }
        else if (lineSensorVal == 1)
        {
```

```
            // Left sensor off the line, right sensor on the line,
turn slightly to the right
            motor3.run(80);
            motor4.run(100);
        }
        else if (lineSensorVal == 2)
        {
            // Left sensor on the line, right sensor off the line,
turn slightly to the left
            motor3.run(100);
            motor4.run(80);
        }
        else
        {
            // Both sensors off the line, stop motors and start to
rotate to see for junctions
            motor3.stop();
            motor4.stop();
            // Turn right and check for line
            if (turnWithLineCheck(-150, 150,600))
            {
                forward();
            }
             // Turn 180 degree clock wise and check for line
            else if (turnWithLineCheck(150, -150,1200))
            {
                forward();
            }
            else
            {
                break;
            }
        }
        delay(100);
    }
}
```

6. REFERENCES

1. Improving path planning of autonomous vacuum cleaners ... (n.d.). http://www.diva-portal.org/smash/get/diva2:1213402/FULLTEXT01.pdf

2. Mohamed Amine Yakoubi, Highlights•Method for the path planning of cleaner robot for coverage region.•Genetic Algorithms approach.•To demonstrate the efficiency and feasibility of our approache and validate the results obtained, & AbstractThe vacuum cleaner robot should have a mechanism such as the artificial intelligence to solve the problem of cleaning the entire environment areas taking into account some factors such as the number of turns and the length of the trajectory. This . (2016, May 27). *The path planning of cleaner robot for coverage region using genetic algorithms*. Journal of Innovation in Digital Ecosystems. https://www.sciencedirect.com/science/article/pii/S2352664516300050

3. PandaMan. (2023, July 26). *A* pathfinding algorithm: Efficiently navigating the maze of possibilities*. Medium. https://panda-man.medium.com/a-pathfinding-algorithm-efficiently-navigating-the-maze-of-possibilities-8bb16f9cecbd

4. Li, A., Kacprzak, D., & Hu, A. P. (2022, January 1). *Wireless Charger Design of Robot Vacuum Cleaners with Power Repeaters for High Compatibility*. Journal of Electromagnetic Analysis and Applications.https://doi.org/10.4236/jemaa.2022.145005

5. *What kind of substance is FE3O4?*. TRUNNANO. (n.d.). https://www.nanotrun.com/article/what-kind-of-substance-is-fe3o4-i00582i1.html

6. Asafa, T. B., Afonja, T., Olaniyan, E., & Alade, H. (2018, December 1). *Development of a vacuum cleaner robot*. Alexandria Engineering Journal /Alexandria Engineering Journal. https://doi.org/10.1016/j.aej.2018.07.005

7. Radu, S. (n.d.). *Relation between function and form in vacuum cleaners design*. Journal of Industrial Design and Engineering Graphics. http://sorging.ro/jideg/index.php/jideg/article/view/211

8. T.B. Asafa, & AbstractModern households are becoming more automated thereby delivering convenience and reducing time spent on house chores. While vacuum cleaners have made

home cleaning easier. (2018, November 14). *Development of a vacuum cleaner robot*. Alexandria Engineering Journal. https://www.sciencedirect.com/science/article/pii/S1110016818300899?ref=pdf_download&fr=RR-2&rr=87a65fd1e8bf40d4

9. B. S. Gu, T. Dharmakeerthi, S. Kim, M. J. O'Sullivan and G. A. Covic, "Optimized Magnetic Core Layer in Inductive Power Transfer Pad for Electric Vehicle Charging," in IEEE Transactions on Power Electronics, vol. 38, no. 10, pp. 11964-11973, Oct. 2023, doi: 10.1109/TPEL.2023.3299959.

10. Moss, D. (2023, October 23). *The heartbeat of drone lidar: Why a fantastic Imu Matters*. LinkedIn. https://www.linkedin.com/pulse/heartbeat-drone-lidar-why-fantastic-imu-matters-david-moss-pypvc/

11. Admin, A. (2022, August 4). *AA battery: Everything you need to know*. Microbattery. https://www.microbattery.com/blog/post/battery-bios:-everything-you-need-to-know-about-the-aa-battery/

12. Pranav Inani, Anirudh Topiwala, & Abhishek Kathpal. (n.d.). *Frontier Based Exploration for Autonomous Robot*. arxiv.org. https://arxiv.org/ftp/arxiv/papers/1806/1806.03581.pdf

13. Poonam Garg. (n.d.). *A Comparison between Memetic algorithm and Genetic algorithm for the cryptanalysis of Simplified Data Encryption Standard algorithm* . Arxiv. https://arxiv.org/ftp/arxiv/papers/1004/1004.0574.pdf

14. Díez-González, J., Verde, P., Ferrero-Guillén, R., Álvarez, R., & Pérez, H. (2020, September 24). *Hybrid memetic algorithm for the node location problem in local positioning systems*. MDPI. https://www.mdpi.com/1424-8220/20/19/5475

15. Yaaximus. (n.d.). *Yaaximus/genetic-algorithm-path-planning*. GitHub. https://github.com/Yaaximus/genetic-algorithm-path-planning

16. Yang, Simon & Hu, Yanrong. *ROBOT PATH PLANNING IN UNSTRUCTURED ENVIRONMENTS USING A KNOWLEDGE-BASED GENETIC ALGORITHM*. researchgate. (n.d.).https://www.researchgate.net/publication/267550002_ROBOT_PATH_PLANNING_I

N_UNSTRUCTURED_ENVIRONMENTS_USING_A_KNOWLEDGE-BASED_GENETIC_ALGORITHM

17. (PDF) determining similarity in histological images using ... (n.d.-c). https://www.researchgate.net/publication/232085273_Determining_similarity_in_histological_images_using_graph-theoretic_description_and_matching_methods_for_content-based_image_retrieval_in_medical_diagnostics

18. A. B. (n.d.). *Online kalman filter tutorial*. Kalman Filter Tutorial. https://www.kalmanfilter.net/multiSummary.html

19. Mohamed LAARAIEDH. (n.d.). *Implementation of Kalman Filter with Python Language*. arxiv.org. https://arxiv.org/ftp/arxiv/papers/1204/1204.0375.pdf

20. Amir. (2023, January 18). *Kalman filter with python code*. Medium. https://medium.com/@ab.jannatpour/kalman-filter-with-python-code-98641017a2bd

21. Xie, X., Yan, Z., Zhang, Z., Qin, Y., Jin, H., & Xu, M. (2024, February 29). *Hybrid genetic ant colony optimization algorithm for full-coverage path planning of gardening pruning robots - intelligent service robotics*. SpringerLink. https://link.springer.com/article/10.1007/s11370-024-00525-6

22. Gharsalli, L. (2022, May 13). *Hybrid genetic algorithms*. IntechOpen. https://www.intechopen.com/chapters/81745

23. Shaveen Singh, Anish Chand, & Sunil Pranit Lal. (n.d.). Improving Spam Detection Using Neural Networks Trained by Memetic Algorithm. https://www.researchgate.net/publication/261468690_Improving_Spam_Detection_Using_Neural_Networks_Trained_by_Memetic_Algorithm

24. *RFC editor*. RFC Editor. (n.d.). https://www.rfc-editor.org/rfc/pdfrfc/rfc6455.txt.pdf

25. Gary Twinn. (n.d.). *Combining low-cost single board computers with open-source software to control noble gas extraction lines*. researchgate. https://www.researchgate.net/publication/366326832_Combining_low-cost_single_board_computers_with_open-source_software_to_control_noble_gas_extraction_lines

26. Chen, P.-H., Li, C., Dong, Z., & Priestley, M. (2022, November 7). *Inductive Power Transfer Battery Charger with IR-based closed-loop control*. MDPI. https://www.mdpi.com/1996-1073/15/21/8319

27. Giulia Di Capua, & N. Femia. (n.d.). *Optimal Coils and Control Matching in Wireless Power Transfer Dynamic Battery Chargers for Electric Vehicles*. researchgate. https://www.researchgate.net/publication/356449143_Optimal_Coils_and_Control_Matching_in_Wireless_Power_Transfer_Dynamic_Battery_Chargers_for_Electric_Vehicles
28. DATASHEET raspberry pi 4 model B. (n.d.-b). https://datasheets.raspberrypi.com/rpi4/raspberry-pi-4-datasheet.pdf

29. Name, Y. (n.d.). *Robotis e*. Manual. https://emanual.robotis.com/docs/en/parts/controller/opencr10/

30. Features. (n.d.-c). https://www.mouser.com/pdfDocs/XW540-T260-R_ProductDescription.pdf

31. L298N motor Driver.pdf. (n.d.-e). https://www.handsontec.com/dataspecs/L298N%20Motor%20Driver.pdf

32. SF20 Lidar Sensor - Mouser Electronics. (n.d.-g). https://www.mouser.com/datasheet/2/1142/SF20___LiDAR_Manual___Rev_12-2943502.pdf

33. Datasheet - Mouser Electronics. (n.d.-b). https://www.mouser.sg/datasheet/2/268/MCP3004_MCP3008_Data_Sheet_DS20001295-3081340.pdf

34. FSR 406 data sheet. (n.d.-e). https://cdn.sparkfun.com/assets/c/4/6/8/b/2010-10-26-DataSheet-FSR406-Layout2.pdf

35. *Keyes Brick Laser Head sensor module (pad hole) with PIN header board digital signal*. Elecbee Factory. (n.d.). https://www.elecbee.com/en-24901-Laser-Head-Sensor-Module-pad-hole-with-Pin-Header-Board-Digital-Signal?network=g&campaign=18719854073&adgroup=140153146062&creative=630917925115&keyword=&target=aud-384004394606%3Apla-296303633664&matchtype=&devicemodel=&placement=&feeditemid=&adpostition=&gad_source=1&gclid=CjwKCAjw5v2wBhBrEiwAXDDoJdYvXaOUvgLySSMbUidHTvaqVzYbuHGft0QgfZC2BcRFQPn2Bi3caBoCiSYQAvD_BwE

36. RS-555 SH/SA. (n.d.-i). https://s3-sa-east-1.amazonaws.com/robocore-lojavirtual/978/Motor-RS-555-SH_SA.pdf

37. Lazada.sg. (n.d.). https://www.lazada.sg/products/dc12v-85w-car-vacuum-cleaner-dc-brushless-motor-high-speed-vacuum-motor-3s-wireless-vacuum-cleaner-motor-i2986297597-s20487126630.html?from_gmc=1&fl_tag=1&exlaz=d_1%3Amm_150050845_51350205_2010350205%3A%3A12%3A19482288611%21%21%21%21%21c%21%2120487126630%215319083762&gad_source=1&gclid=CjwKCAjw5v2wBhBrEiwAXDDoJVR3VL5mUYFay7cLiOQjfOs6pmn_8zTG3aF-HltSbebI0ifrj8jnxBoCdB4QAvD_BwE

38. BCB1012UH. (n.d.). https://www.delta-fan.com/products/BCB1012UH.html

39. Farnell. (n.d.-d). https://www.farnell.com/datasheets/2891584.pdf

40. *Analysis: The evolution of condo sizes over the years*. edgeprop. (n.d.).

41. https://www.edgeprop.sg/property-news/analysis-evolution-condo-sizes-over-years

42. *Understanding vacuum cleaner specifications*. BestVacuum.com. (n.d.).

43. https://www.bestvacuum.com/pages/vacuum-cleaner-specifications

44. *How to choose the Best Robot Vacuum*. How to choose the best robot vacuum - Newsroom. (n.d.). https://www.dyson.com.sg/newsroom/how-to-choose-the-best-robot-vacuum#:~:text=In%20fact%2C%20the%20average%20suction,the%20market%20is%20only%2010AW

45. *Robotis - Dynamixel XW540-T260-R Servo*. Robosavvy - Robotics, CNC, 3D Print, Electronics, Smarthome. (n.d.).

46. https://robotis.co.uk/robotis-dynamixel-xw540-t260-r-servo.html

47. H. Park, J. Kwon. (n.d.). Effect of Air-Gap Between a Ferrite Plate and Metal Strips on Magnetic Shielding. https://www.semanticscholar.org/paper/Effect-of-Air-Gap-Between-a-Ferrite-Plate-and-Metal-Park-Kwon/7dc31900bcda6812aa170e568c1fb927de2488e9

48. *Raspberry Pi SBC006 4 model B motherboard, 4GB RAM*. Amazon.sg: Electronics. (n.d.). https://www.amazon.sg/Raspberry-Pi-SBC006-Model-Motherboard/dp/B07TC2BK1X/ref=pd_bxgy_d_sccl_1/358-6228971-6593711?pd_rd_w=5yGPR&content-id=amzn1.sym.53fedfa9-33a3-49ab-add1-b7fa499a9559&pf_rd_p=53fedfa9-33a3-49ab-add1-b7fa499a9559&pf_rd_r=DGP915K6G5G5AHGWSRHS&pd_rd_wg=p84Ks&pd_rd_r=02ddc534-e53f-48fa-b9cd-e5bf18316672&pd_rd_i=B07TC2BK1X&th=1

49. *Smraza case for Raspberry Pi 4 Model B, acrylic case with cooling fan, 4PCS heatsinks, 5V 3A USB-C power supply for Raspberry Pi 4B (RPI 4 board not included) - black and clear*. Amazon.sg: Electronics. (n.d.-b). https://www.amazon.sg/Smraza-Raspberry-Acrylic-Heatsinks-Included/dp/B07TTMQ4PH/ref=sr_1_1?adgrpid=95227220865&dib=eyJ2IjoiMSJ9.L7l_yiq47XisL_uINjohGa4yv2aH6l8aEb_Xl7mDMOm5p4RAjBnzjHSTVY0MO5zyvBcJTfa30PhDO7KV2uL8Ij1PSYTNP7ZgkKi6kgUQElwzYUEHhnu5OMgx0bdWj_908pru_9QgkrOhlgdHRc6Zmtj6JWc_754w2qPk4t-frKdrsLf8XA6xdQeiLqEuwNeHV8LfsXt7wlWMydGGqaKWzerQIMYIT0ND0z8inCba8w8IJWLYVN2pzYqZe_RyWP0REf3P4bIWHSSV2oWis5wRIWBEA3oqA8roshWa1M1My78.-c05vZByGNpWWD_z3BzLLwH5kDiACID8ScnDZSSbODw&dib_tag=se&hvadid=584016201307&hvdev=c&hvlocphy=9062542&hvnetw=g&hvqmt=e&hvrand=48176066647917981 28&hvtargid=kwd-

822444919824&hydadcr=12361_330665&keywords=raspberry%2Bpi%2B4b%2Bprice&qid=1712817689&sr=8-1&th=1

50. *SanDisk Ultra microSDXC, Squab 64GB, A1, C10, U1, UHS-I, 140MB/s R, 4x6, 10y*. Amazon.sg: Electronics. (n.d.-b). https://www.amazon.sg/SanDisk-Ultra-microSDXC-SQUAB-UHS-I/dp/B0BDRVFDKP/ref=asc_df_B0BDRVFDKP/?tag=googleshoppin-22&linkCode=df0&hvadid=606312833174&hvpos=&hvnetw=g&hvrand=8007271881160505449&hvpone=&hvptwo=&hvqmt=&hvdev=c&hvdvcmdl=&hvlocint=&hvlocphy=9062542&hvtargid=pla-1894667585757&psc=1&mcid=a256be73b929363b825070cc02eca517

51. Doyon, M., Sankaranarayanan, R., Enns, T., Enns, T., & T_Enns. (n.d.). *OpenCR1.0*. ROBOTIS. https://www.robotis.us/opencr1-0/

52. *XL-320 servo dynamixel X servo core motor 114 rpm 0.39nm torque for darwin robot*. eBay. (n.d.). https://www.ebay.com/itm/284578211603?chn=ps&norover=1&mkevt=1&mkrid=711-167022-050914-3&mkcid=2&itemid=284578211603&targetid=293946777986&device=c&mktype=pla&googleloc=9062542&poi=&campaignid=20440965829&mkgroupid=151793468789&rlsatarget=pla-293946777986&abcId=&merchantid=137750361&gad_source=1&gclid=EAIaIQobChMIjp2P6si5hQMV38s8Ah1KwQdbEAQYBiABEgIOHfD_BwE

53. SF20/C lightware lidar. (n.d.-l). https://www.mouser.sg/ProductDetail/LightWare-LiDAR/SF20-C?qs=iLbezkQI%252Bsj8XPNk6jtsvg%3D%3D&mgh=1&utm_id=17989067980&gad_source=1&gclid=EAIaIQobChMI_efSo8q5hQMV1KdmAh1mPA2JEAQYASABEgIBmvD_BwE

54. MCP3008 Analog to DC Converter (n.d.-i). https://www.mouser.sg/ProductDetail/Microchip-Technology/MCP3008-I-P?qs=AF%252BffTaPb30XZ0OdV6HdVg%3D%3D&mgh=1&utm_id=17989067980&gad_source=1&gclid=CjwKCAjw5v2wBhBrEiwAXDDoJQipFWkw5CJmPqxAPQEomKPL7o_kDQ99u0vALF4zhmSi4NACq7heCxoCuuUQAvD_BwE

55. Sen-09376 Sparkfun | mouser Singapore. (n.d.-l). https://www.mouser.sg/ProductDetail/SparkFun/SEN-09376?qs=WyAARYrbSnZm3k2OB4XwnA%3D%3D

56. *Keyes Brick Laser Head sensor module (pad hole) with PIN header board digital signal*. Elecbee Factory. (n.d.-a). https://www.elecbee.com/en-24901-Laser-Head-Sensor-Module-pad-hole-with-Pin-Header-Board-Digital-Signal?network=g&campaign=18719854073&adgroup=140153146062&creative=630917925115&keyword=&target=aud-384004394606%3Apla-296303633664&matchtype=&devicemodel=&placement=&feeditemid=&adpostition=&gad_source=1&gclid=CjwKCAjw5v2wBhBrEiwAXDDoJdYvXaOUvgLySSMbUidHTvaqVzYbuHGft0QgfZC2BcRFQPn2Bi3caBoCiSYQAvD_BwE

57. *2AMP 7V-30V L298N motor driver / stepper driver (2 channels)*. Cytron Technologies Singapore. (n.d.). https://sg.cytron.io/p-2amp-7v-30v-l298n-motor-driver-stepper-driver-2-channels?r=1

58. *Mellor electric brushed DC Motor, 24 V DC, 10 NCM, 3500 rpm*. RS. (n.d.). https://sg.rs-online.com/web/p/dc-motors/2483671?cm_mmc=SG-PLA-DS3A-_-google-_-PLA_SG_Pmax_LocalStock_0923-_--_-&matchtype=&&gad_source=1&gclid=EAIaIQobChMIzZei-Mq5hQMVmaRmAh2o7wV7EAQYAiABEgLt-_D_BwE&gclsrc=aw.ds

59. *12V DC Universal Motor HCDM2055*. Lazada.sg. (n.d.-a). https://www.lazada.sg/products/dc12v-85w-car-vacuum-cleaner-dc-brushless-motor-high-speed-vacuum-motor-3s-wireless-vacuum-cleaner-motor-i2986297597-s20487126630.html?from_gmc=1&fl_tag=1&exlaz=d_1%3Amm_150050845_51350205_2010350205%3A%3A12%3A19482288611%21%21%21%21%21c%21%2120487126630%215319083762&gad_source=1&gclid=CjwKCAjw5v2wBhBrEiwAXDDoJVR3VL5mUYFay7cLiOQjfOs6pmn_8zTG3aF-HltSbebI0ifrj8jnxBoCdB4QAvD_BwE

60. BCB1012UH Delta Electronics | mouser. (n.d.-b). https://www.mouser.com/ProductDetail/Delta-Electronics/BCB1012UH?qs=Tx9Tc1LIQ+FbjAzroIPHuQ==

61. *Original Xiaomi 1s/mi robot vacuum/SDJQR01RR/SDJQR02RR/SDJQR03RR parts of 5200mAh Li-Ion Battery: Shopee singapore*. shopee.sg. (n.d.). https://shopee.sg/product/526269342/16394943245?gad_source=1&gclid=EAIaIQobChMI6bjnu8u5hQMVNKVmAh3n6AFzEAQYASABEgLzCfD_BwE

62. *MCFT-27T-4.2AL-127 - piezo element*. element14. (n.d.). https://sg.element14.com/multicomp-pro/mcft-27t-4-2al-127/piezo-element/dp/1801061?gad_source=4&gclid=CjwKCAjw26KxBhBDEiwAu6KXt-Ddq5H0rDOnwc1aSE38oBvkcL3cmt5ynQjllFE90cJ3Y6_r0lYrNxoCXHQQAvD_BwE&mckv=_dc%7Cpcrid%7C%7Cpkw%7C%7Cpmt%7C%7Cslid%7C%7Cproduct%7C1801061%7Cpgrid%7C%7Cptaid%7C%7C&CMP=KNC-GSG-SHOPPING-PMAX-TOP-AOV-CATEGORY

63. *6 pack HEPA filter for Neato Botvac d7 d3 d4 d5 d6 D70 D75 D80 D85 ...* Fruugo. (n.d.). https://www.fruugo.sg/6-pack-hepa-filter-for-neato-botvac-d7-d3-d4-d5-d6-d70-d75-d80-d85-for-neato-botvac-75e-80-filters/p-207016052-440633169