VILNIUS UNIVERSITY
FACULTY OF MATHEMATICS AND INFORMATICS
INSTITUTE OF COMPUTER SCIENCE
INFORMATION TECHNOLOGIES STUDY PROGRAM

Problem-Based Project

Rover

Done by:
Gabrielius Drungilas
Aistė Grigaliūnaitė
Nedas Janušauskas
Adomas Jonavičius

Supervisor:
dr. Linas Bukauskas

Vilnius
2023

Preface

This project was done during the 3rd semester of the study program *Information Technologies* in the specialization of Innovative studies. We chose the topic *Autonomous Robot Companion* suggested during the project market on the first lecture of the subject ‘‘Problem-Based Project’’. The topic sounded very interesting as robotics was a topic we had minimal experience in, as well as the project being the only one of its kind in the project market. Having minimal experience with robotics, the topic drew us in and raised various questions - what parts to use for the robot, what it would look like, what programming languages to use, the function of the robot and the logic behind it, and where to begin with such an unfamiliar yet intriguing project.

January 12, 2023

Gabrielius Drungilas

Aistė Grigaliūnaitė

Nedas Janušauskas

Adomas Jonavičius

Contents

Preface 2

Abstract 6

Santrauka 7

Introduction 8

1 Analysis 9

2 Overview 10
2.1 Project overview . . . 10
2.2 Context diagram . . . 10

3 System architecture 11
3.1 Hardware system . . . 11
3.1.1 Hardware diagram . . . 11
3.1.2 UML deployment diagram . . . 12
3.1.3 Hardware components . . . 13
3.2 Software system . . . 13
3.2.1 Software used on the Raspberry Pi . . . 13
3.2.2 Software used for development . . . 14
3.2.3 Miscellaneous software . . . 14

4 Deliverable internals 14
4.1 Structural aspects . . . 14
4.2 Dynamic aspects . . . 14
4.2.1 Basic robot control . . . 14
4.2.2 Internal logic of USB cameras . . . 14
4.2.3 Pathfinding . . . 15
4.2.4 Multiprocessing . . . 15

5 Functional Requirements 15

6 Non-Functional requirements 16

7 User Interface 17
7.1 Turning On The Robot . . . 17
7.2 Remote Controls . . . 17
7.2.1 Main Buttons . . . 17
7.2.2 Autonomous Mode Buttons . . . 18
7.2.3 Manual Mode Buttons . . . 18

8 Pathfinding algorithm 19
8.1 Description . . . 19
8.2 Pseudo code . . . 19
8.3 Example data . . . 21

8.4 Real data 21
8.4.1 Real world example 21
8.4.2 Stored data 24

9 Mapping algorithm 25

10 Testing 26
10.1 Motor Test 26
10.1.1 Description 26
10.1.2 Precondition 26
10.1.3 Assumption 26
10.1.4 Test steps 27
10.1.5 Expected results 27
10.2 Drivetrain Test 28
10.2.1 Description 28
10.2.2 Precondition 28
10.2.3 Assumption 28
10.2.4 Test steps 28
10.2.5 Expected results 28
10.3 Precise Turning Test 29
10.3.1 Description 29
10.3.2 Precondition 29
10.3.3 Assumption 29
10.3.4 Test steps 29
10.3.5 Expected results 30
10.4 Distance Sensor Test 31
10.4.1 Description 31
10.4.2 Precondition 31
10.4.3 Assumption 31
10.4.4 Test steps 31
10.4.5 Expected results 31
10.5 Object Recognition Test 32
10.5.1 Description 32
10.5.2 Precondition 32
10.5.3 Assumption 32
10.5.4 Test steps 32
10.5.5 Expected results 33
10.6 Accelerometer / Gyroscope (IMU) Test 33
10.6.1 Description 33
10.6.2 Precondition 33
10.6.3 Assumption 33
10.6.4 Test steps 34
10.6.5 Expected results 34
10.7 Buzzer Test 34
10.7.1 Description 34
10.7.2 Precondition 34

10.7.3 Assumption 34
10.7.4 Test steps 34
10.7.5 Expected results 34
10.8 Mapping System Test 35
10.8.1 Description 35
10.8.2 Precondition 35
10.8.3 Assumption 35
10.8.4 Test steps 35
10.8.5 Expected results 35
10.9 Script Loading On Boot Test 36
10.9.1 Description 36
10.9.2 Precondition 36
10.9.3 Assumption 36
10.9.4 Test steps 36
10.9.5 Expected results 36

11 Related work 37

Conclusions 38

References 39

Abstract

In this day and age the world is full of autonomous robots that help with various tasks - cleaning a home, mowing the lawn, transporting goods from one place to another, and more. The market for autonomous robots has been steadily rising for the past decade, indicating the current and future popularity of autonomous robot companions. This work intends to introduce an autonomous robot companion with the ability to map a room, detect and avoid obstacles, and recognize objects, all with the use of sensors and a camera. This work explains all technologies used to implement such a robot, including diagrams of its hardware and software. The robot's functionality was implemented using python, bash, and the OpenCV library. The robot's hardware consists of a microcomputer, a camera, ultrasound distance sensors, a gyroscope and accelerometer module, stepper motors, a remote controller, a piezo buzzer, and other various electronic components. The room mapping will be explained and examples of test results will be presented. The robot will be able to execute the following tasks: "explore the room", "find an object", "return to home", and "play a song". The user will give the robot the task using a remote controller. The expected result is an autonomous robot with the ability to roam and map a room, as well as execute tasks provided by the user.

Keywords: Autonomous Robot, Pathfinding, Room Mapping, Object recognition, Obstacle Avoidance

Santrauka

Rover

Šiandieniniame pasaulyje gausu autonominių robotų, kurie atlieka įvairias užduotis – valo grindis, pjauna žolę, transportuoja prekes iš vienos vietos į kitą. Per pastarąjį dešimtmetį autonominių robotų rinka stabiliai auga, tai rodo dabartinį ir būsimą autonominių robotų populiarumą. Šiuo darbu ketinama pristatyti autonominį robotą, gebantį sukurti patalpos žemėlapį, aptikti ir išvengti kliūtis bei atpažinti objektus. Šiame darbe paaiškinamos visos tokiam robotui įgyvendinti naudojamos technologijos, įskaitant jo techninės ir programinės įrangos diagramas. Roboto funkcionalumas buvo įgyvendintas naudojant python, bash ir OpenCV. Roboto techninę įrangą sudaro mikrokompiuteris, kamera, ultragarsiniai atstumo jutikliai, giroskopo ir akselerometro modulis, žingsniniai motorai, nuotolinio valdymo pultelis, garsinis signalizatorius ir kiti įvairūs elektroniniai komponentai. Šiame dokumente bus paaiškintas patalpų žemėlapio kūrimas ir pateikti bandymų rezultatų pavyzdžiai. Robotas galės atlikti tokias užduotis: "tyrinėti patalpą", "rasti objektą", "grįžti namo", "sugroti dainą". Užduotį robotui duos naudotojas, naudodamas nuotolinio valdymo pultelį. Norimas rezultatas - autonominis robotas gebantis tyrinėti patalpą ir sukurti jos žemėlapį, bei įvykdyti vartotojo pateiktas užduotis.

Raktiniai žodžiai: Autonominis Robotas, Kelio Ieškojimas, Patalpos Žemėlapio Kūrimas, Objektų Atpažinimas, Kliūčių Vengimas

Introduction

The goal of this project was to develop an autonomous robot companion with the ability to map the room, detect and avoid obstacles, recognize objects with the use of a camera, execute tasks provided by the user, and produce sound signals. Similar concepts and robot kits have been analyzed to get a better understanding of the best way to implement this project. The main task of the robot is to be able to explore the room with minimal error using a limited obstacle detection system. The robot will use a Raspberry Pi microcomputer, multiple ultrasound distance sensors, a camera, power components, and various other components. The robot's turning will be assisted by an accelerometer/gyroscope module to minimize errors from the slippage of tracks. The robot will execute a number of the available tasks using the OpenCV library. The start of this document will include the market analysis, the overview of the project, followed by the implementation details, algorithm explanation, as well as testing.

1 Analysis

To realize the robot companion project, we researched robots and robot kits available online, as well as online videos on small robot projects. There were two major types of robots - those using Arduino, and those using Raspberry Pi microcomputers. We decided on utilizing a Raspberry Pi because we were more comfortable using Python than C++. There were numerous options for robots, namely, robot kits PiCar-X[5] and PiCar-S v2[4] from Sunfounder, and FreeNove 4WD[1] Smart Car. From our analysis, we noticed all of the robots and robot kits had many shared components: cameras, ultrasound sensors, wheels or tracks, motors, Raspberry Pi, Lithium-ion batteries, and other power components. From our analysis, we decided to keep some of the main components from other robots in the market. One of which are ultrasound distance sensors. They will be the main tool for detecting obstacles in the way of the robot. Of course, there are better alternatives in the market like LiDAR sensors or stereo computer vision systems, but they are quite expensive. For computer vision, there were various academic papers that utilized the OpenCV library[2][3]. As we decided to utilize computer vision, we decided to use the OpenCV library due to its popularity and amount of available resources regarding the library. Because it is the first robot that we built, we decided to keep the concept simple and build up from that. Furthermore, we chose a different type of motor than in most other robots, a stepper motor, because we wanted to have more control and reliability of movement.

2 Overview

2.1 Project overview

The goal of the project is to design an autonomous robot for indoor environment use. The "Rover" will be able to roam around the area freely and remember the layout while executing the user-given tasks. It will be controlled by a microcomputer, which will execute the Python scripts and interact with other electronic components (sensors, controllers, etc...) to obtain all the necessary data for proper execution.

2.2 Context diagram

The robot's context diagram is represented in the following:

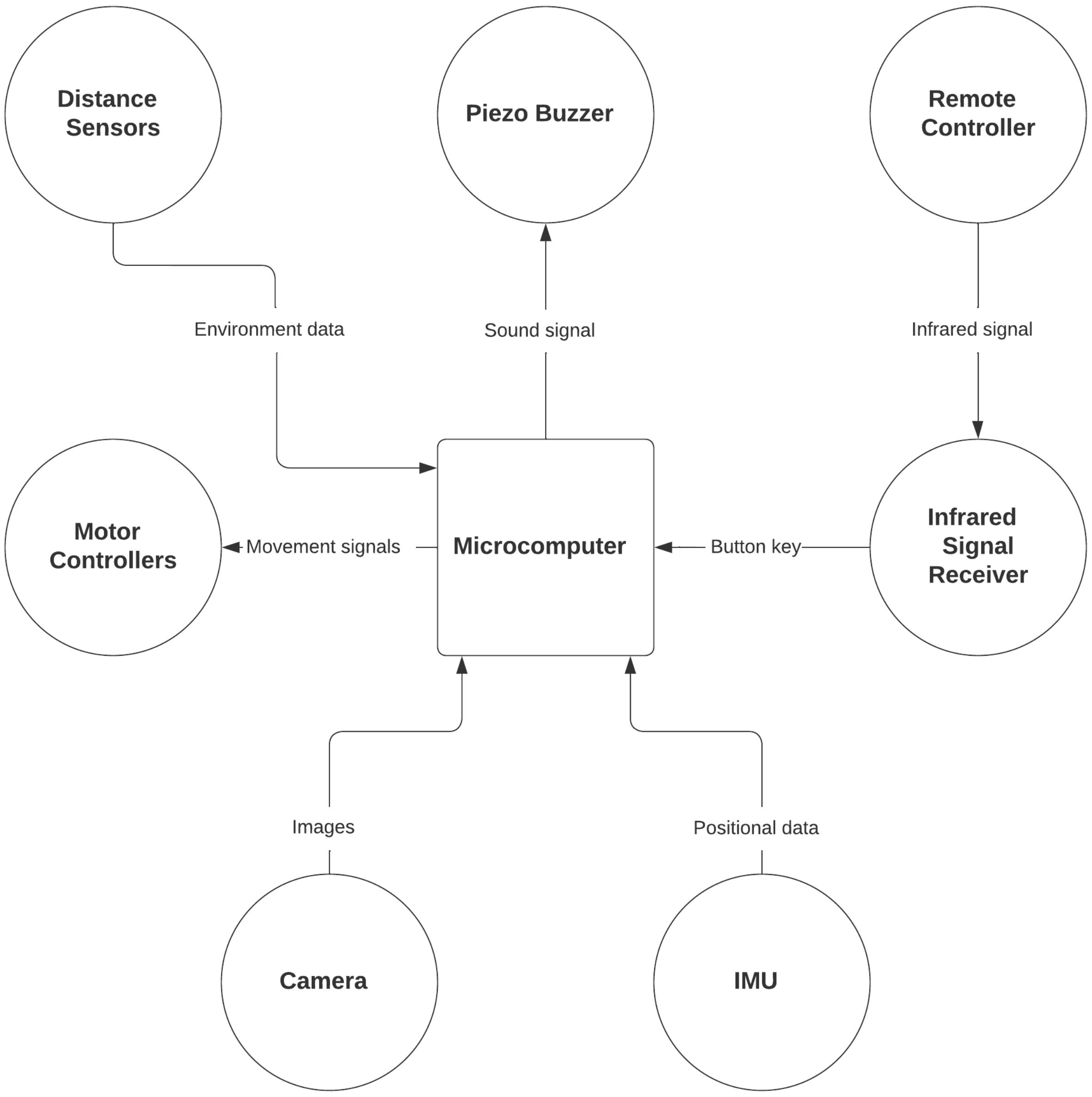


Figure 1. Context diagram

3 System architecture

3.1 Hardware system

3.1.1 Hardware diagram

The robot’s hardware is represented in the following diagram:

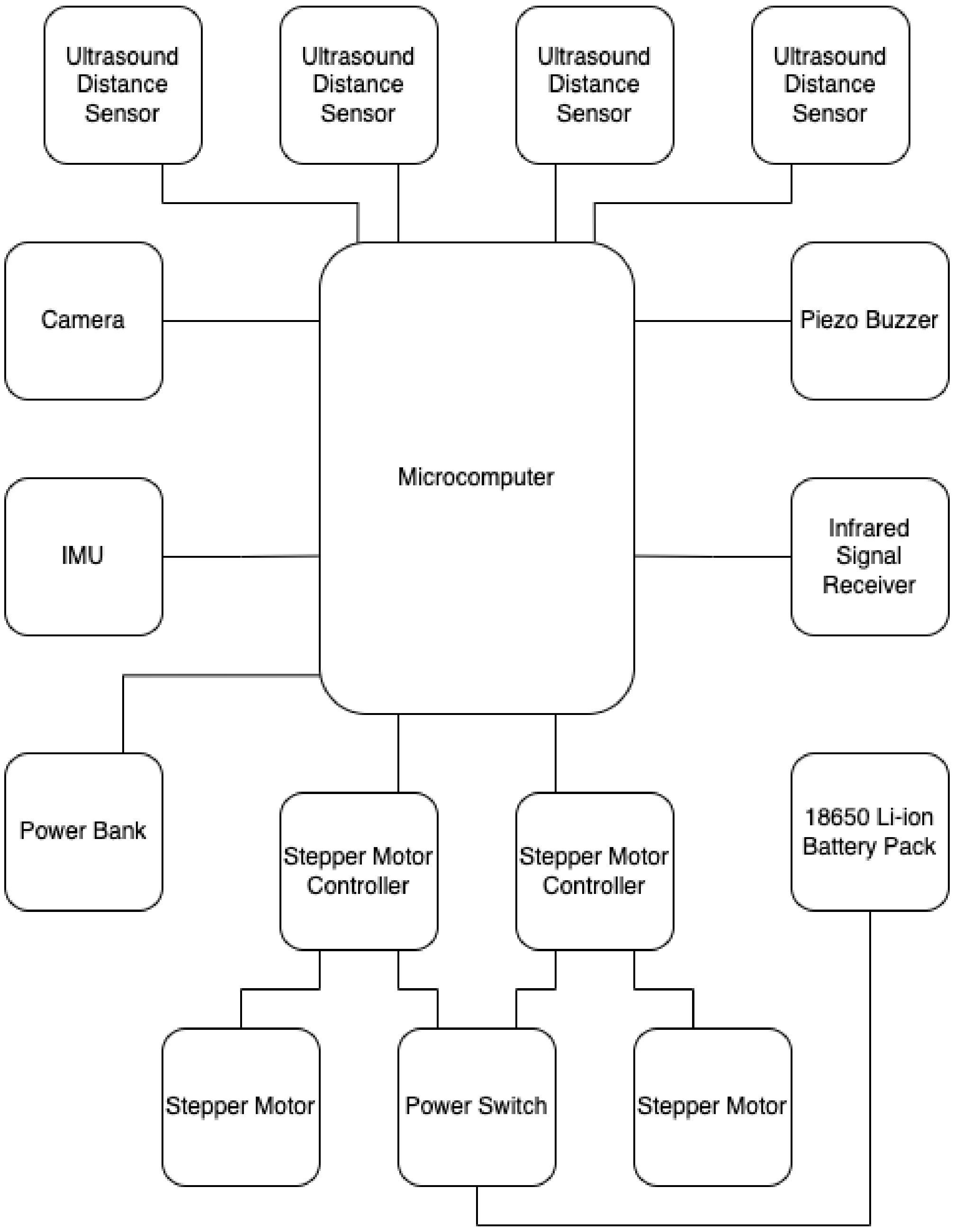


Figure 2. Hardware diagram

3.1.2 UML deployment diagram

The robot's UML deployment diagram is represented in the following:

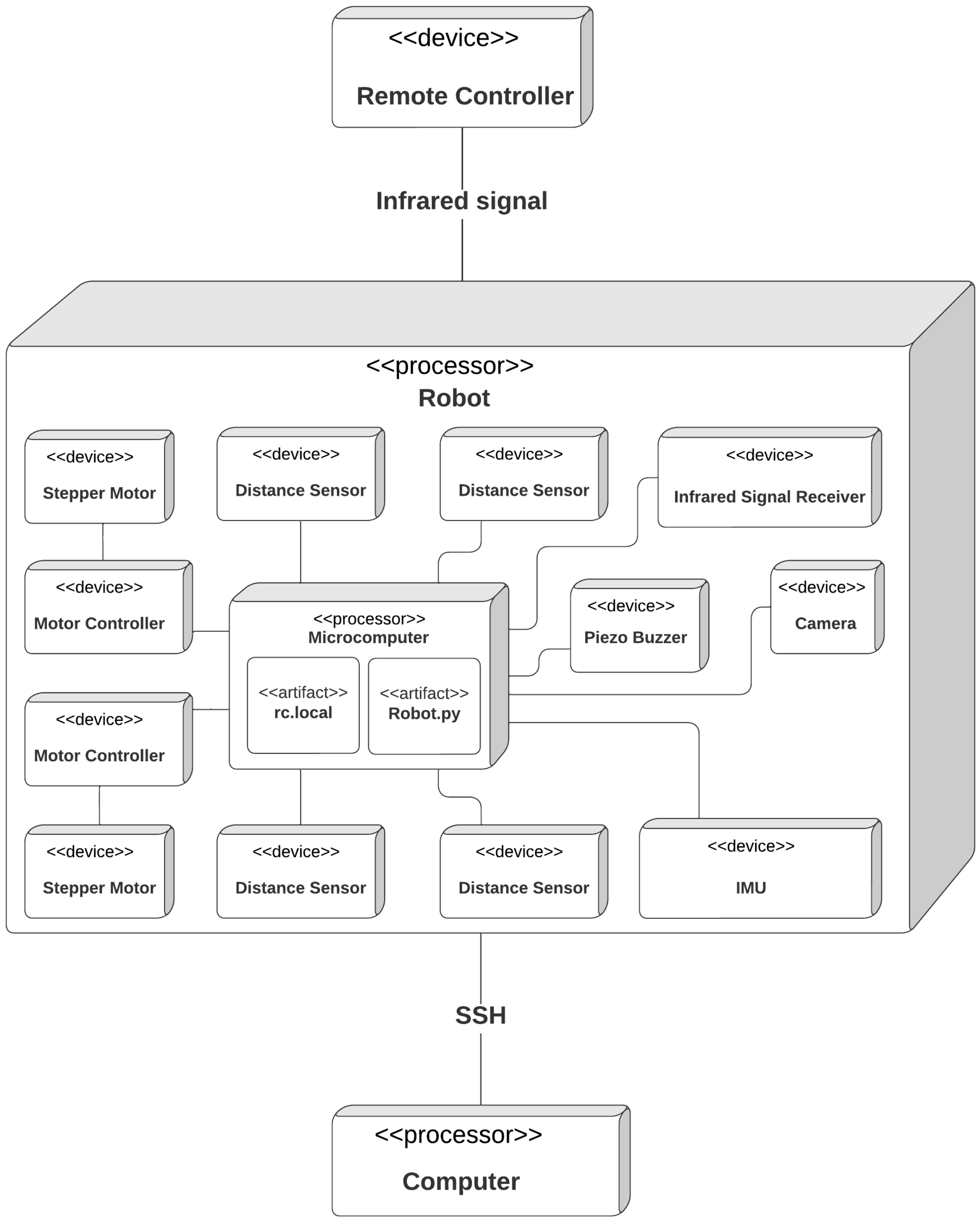


Figure 3. Deployment diagram

3.1.3 Hardware components

The robot will be constructed using the following hardware: [Figure 2]

- Raspberry Pi 4 Model B 4GB. Used for main computing.
- (4) HC-SR04 ultrasound distance sensors. Used for obstacle detection.
- (1) USB camera. Used for object recognition.
- (2) NEMA 17 stepper motors. Used for moving the tracks.
- (2) DRV8825 stepper motor controllers. Used for controlling and powering the stepper motors.
- MPU6050 accelerator/gyroscope module (IMU). Used for tracking the robot's position when turning.
- HX1838 infrared signal receiver. Used for receiving infrared signals from the remote controller.
- Remote controller. Used for giving commands to the robot.
- Piezo buzzer. Used for notifying the user about the robot's status.
- 32GB MicroSD. Used for storing the OS.
- Power switch. Used for powering on and off motor controllers and stepper motors.
- Power bank. Used for powering Raspberry Pi microcomputer.
- 18650 battery holder. Used for storing and combining batteries into one package.
- (4) 18650 Li-ion batteries. Used for powering motor controllers and stepper motors.
- (4) 1kΩ Resistors. Used for signal voltage reduction.
- (4) 2kΩ Resistors. Used for signal voltage reduction.
- 100 μF Capacitor. Used for voltage spike prevention.

3.2 Software system

The following technologies will be used to build the robot and maintain the required code:

3.2.1 Software used on the Raspberry Pi

- Linux OS - OS that will be run on the Raspberry Pi.
- Python - code for the robot's functionality.
- C++ - code for hardware tests.
- OpenCV - libraries used for computer vision.
- Bash - used to execute any scripts required on startup.

3.2.2 Software used for development

- Git and GitLab - version control.
- VSCode - development environment.

3.2.3 Miscellaneous software

- AutoCAD - 3D modeling.
- 3D Printer - manufacturing of robot body parts.

4 Deliverable internals

4.1 Structural aspects

The main components of the robot are the Raspberry Pi, camera, ultrasound distance sensors, motors, infrared signal sensor, and the piezo buzzer. The cameras and sensors will be used to detect the robot's surroundings, the motors will be used to move the robot. The infrared signal sensor will be used to read the signals from the remote controller. The beeper will be used to notify the user of errors or events, and the Raspberry Pi will be used to execute the required code.

Linux OS is installed on Raspberry Pi. BASH is used to run the main script on startup. The robot's software was written using python3.11.

4.2 Dynamic aspects

In order to accomplish the desired functionality, the robot's components will send and receive data from the Raspberry Pi. The infrared signal receiver, when it receives a signal, will add tasks to the robot's task queue. The stepper motors will make it possible for the robot to keep track of its location in the room, as well as move a specified distance.

4.2.1 Basic robot control

A remote controller will send an infrared signal to the infrared signal receiver. This will be the main way that the user is going to interact with the robot. The user will be able to send signals to start different tasks like roaming freely, finding a bottle, returning to starting position, stopping, using beeper, and similar tasks. Tasks received by remote controller can be added to the task list and will be executed sequentially.

4.2.2 Internal logic of USB cameras

A "camera" class is created as an extension of the thread class to be able to keep cameras working together at the same time. A separate thread is created that will compare the view of the different cameras and calculate the disparity between different areas of the picture. Then, the output of the disparity calculation is taken, the path ahead is analyzed, it is decided whether it is clear, and the result is used to update the 2D array representing the known map of the surrounding area. Then, the next move the robot should make to get to the goal is calculated.

The known map will hold all data about the surrounding area: each element in the array will represent the state associated with a particular area. The element will be an object with various state flags and additional information. It can hold states like: unknown, current "Rover" position, obstacle, destination, starting position, and any additional useful information that could impact the decision-making of the rover. These elements will be held in a dictionary and new elements will be added if needed to represent new location.

4.2.3 Pathfinding

The path will be calculated using lightning algorithm to find the shortest viable path. If such a path does not exist, the bot will analyze if a path is still possible, possibly through unexplored area, then test it.

4.2.4 Multiprocessing

There will be 5 processes running simultaneously each responsible for:

1. Movement/decision making.
2. Reading data from ultrasound distance sensors.
3. Reading data from the camera.
4. Reading data from IMU.
5. Reading infrared signal data from the remote controller.

5 Functional Requirements

- The robot has tracks and is able to move around the room, including moving forwards, backwards, and turning left and right.
- Turning left and right with less than 1.5 degree error (on solid ground).
- The robot decides how to move by itself, based on a user-given task.
- The robot can detect obstacles (objects in its path - furniture, walls, etc.) in front of and behind it.
- The robot makes decisions on where to move when an obstacle is detected.
- The robot can be turned on/off using a power switch.
- The robot stores the layout of the room.
- The robot informs the user about events, navigation, or other errors using sound signals.

6 Non-Functional requirements

Reliability: The robot will be operational until turned off unless it runs out of power. The robot will have an expected battery life of 2 hours.

Usability: The robot can be manually controlled with a remote controller. The user is able to give commands for the robot to move forward, backward, turn left, right and stop.

Performance: Robot can compute the shortest path to the destination in order to reduce service time and energy consumption

7 User Interface

7.1 Turning On The Robot

To turn on the Robot:

1. Plug in the USB C cable into Raspberry Pi.
2. Turn on the power switch which is located on top of the robot.
3. After hearing a short beep, press button (4), which will load the main script.
4. After hearing 3 short beeps the Rover is ready to be used.

7.2 Remote Controls

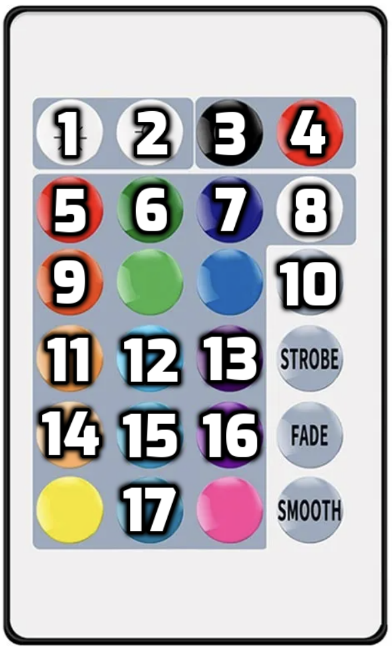


Figure 4. Remote Controller

7.2.1 Main Buttons

Button (1): Autonomous Mode

Button (2): Manual Mode

Button (3): OFF

Button (4): ON

Button (8): Clear The Map

7.2.2 Autonomous Mode Buttons

Button (5): Explore

Button (6): Find a Bottle

Button (7): Return to Home

Button (9): Play a Song

Button (10): Clear Task Queue

7.2.3 Manual Mode Buttons

Button (11): Left Micro-turn

Button (12): Forward

Button (13): Right Micro-turn

Button (14): Left

Button (15): Stop

Button (16): Right

Button (17): Backward

8 Pathfinding algorithm

8.1 Description

The code implements a pathfinding algorithm to find the shortest path between a start and end point on a map. The __calculate_distances function recursively calculates the distances from the start tile to all other reachable tiles on the map. The get_shortest_path function uses this to trace the path from the end tile back to the start tile by repeatedly finding the lowest distance neighbor.

Additional functions get_neighbours and _lowest_distance_neighbour are created to support this algorithm.

8.2 Pseudo code

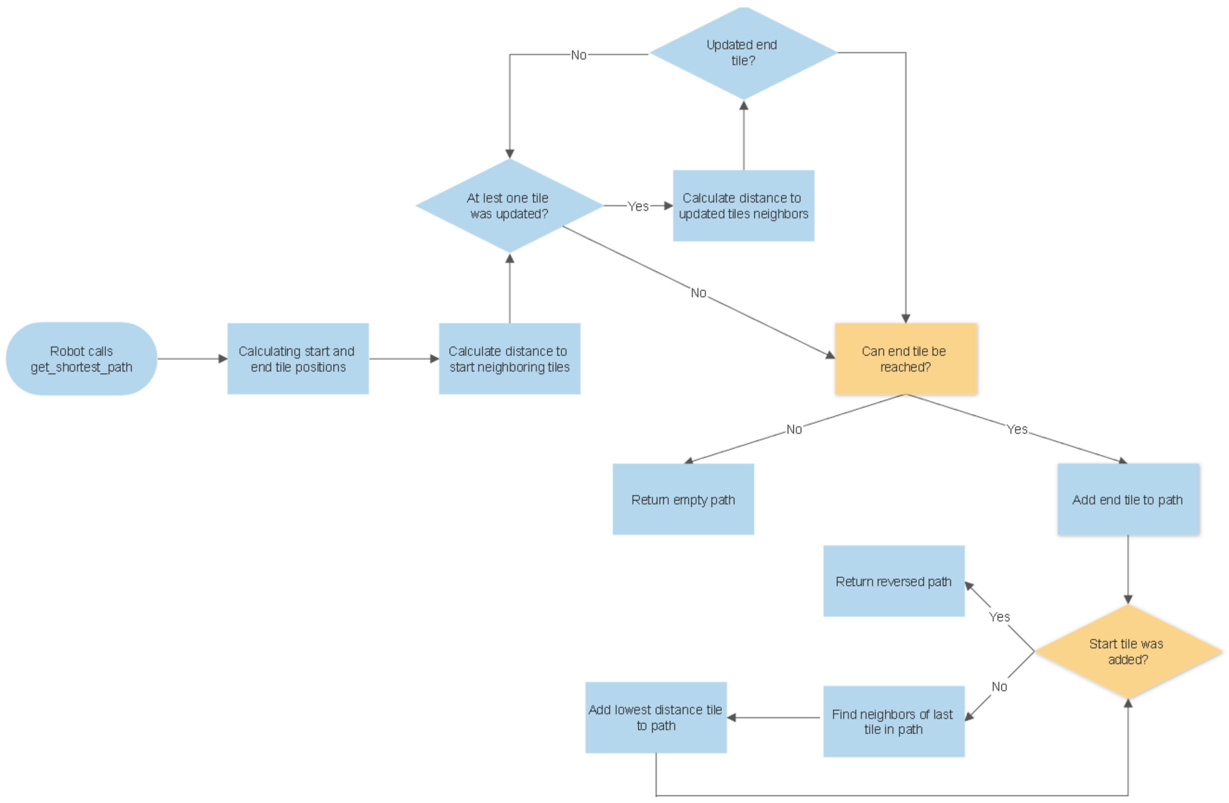

Updated end tile?
No
At lest one tile was updated?
Yes
Calculate distance to updated tiles neighbors
No
Robot calls get_shortest_path
Calculating start and end tile positions
Calculate distance to start neighboring tiles
Can end tile be reached?
No
Yes
Return empty path
Add end tile to path
Return reversed path
Yes
Start tile was added?
No
Add lowest distance tile to path
Find neighbors of last tile in path

Figure 5. Flow chart of pathfinding algorithm

```
global Map map
global start = [0,0], end = [0,0]
#  returns a list of tiles that are neighbours of the given tile
Function  get_neighbours(Tile tile) returns List[Tile]:
    neighbours = []
    if (tile.x, tile.y + 1) exists in map:
        neighbours.append(map[tile.x, tile.y + 1])
    if (tile.x, tile.y - 1) exists in map:
        neighbours.append(map[tile.x, tile.y - 1])
    if (tile.x + 1, tile.y) exists in map:
        neighbours.append(map[tile.x + 1, tile.y])
    if (tile.x - 1, tile.y) exists in map:
        neighbours.append(map[tile.x - 1, tile.y])
    return neighbours
```

```
#  recursively calculates the distance from the start to rest of the tiles
Function  __calculate_distances(Tile tile, int n=0) returns None:
    neighbours = get_neighbours(tile)
    for i in range(len(neighbours)):
        if neighbours[i] is an obstacle or is unknown:
            continue
        elif neighbours[i].distance is None or neighbours[i].distance > n + 1:
            neighbours[i].distance = n + 1
            if neighbours[i] is the end:
                break
              calculate_distances(neighbours[i], n + 1)
```

```
#  returns the neighbour with the lowest distance to start
Function  get_lowest_distance_neighbour(Tile tile) returns Tile:
    neighbours = get_neighbours(tile)
    lowest_distance = None
    lowest_distance_neighbour = None
    for neighbour in neighbours:
        if neighbour is an obstacle or is unknown:
            continue
        elif lowest_distance is None or neighbour.distance < lowest_distance:
            lowest_distance = neighbour.distance
            lowest_distance_neighbour = neighbour
    return lowest_distance_neighbour
```

```
#  returns the shortest path from start to end or an empty list if no path exists
Function  get_shortest_path() returns List[Tile]:
    path = []
    if start is not set or end is not set:
        return path
    start_tile = map[start]
    end_tile = map[end]
    start_tile.distance = 0
    __calculate_distances(start_tile)
    if end_tile.distance is None:
        return path
    cur_tile = end_tile
    for i in range(cur_tile.distance):
        path.append(cur_tile)
        cur_tile = get_lowest_distance_neighbour(cur_tile)
    return reversed path
```

8.3 Example data

S - start, E - end, # - obstacle, N – distance is None, number – distance from the start
Start = [0,1]
End = [4,3]
2D array representing map before calculating distances to start:

N	S	N	N	N
N	#	N	#	N
N	N	N	N	N
N	N	N	N	E

2D array representing map after calculating distances to start:

1	S	1	2	3
2	#	2	#	4
3	4	3	4	5
4	5	4	5	E

Final path extracted from calculated distances:
Path: End - (4,3), (3,3), (2,3), (2,2), (2,1), (2,0), start - (1,0)
Reversing path to get steps for robot:
(1,0), (2,0), (2,1), (2,2), (2,3), (3,3), (4,3)

8.4 Real data

8.4.1 Real world example

S - start, E - end, # - obstacle, N – distance is None, number – distance from the start
2D arrays representing map during iterations:

Starting map:

N	N	N	N	N	E
N	N	N	N	N	N
N	N	N	N	N	N
N	N	N	N	N	N
N	N	N	N	N	N
N	N	N	N	N	N
#	#	#	#	N	N
N	N	N	N	N	N
N	N	N	N	N	N
S	N	N	N	N	N

Map after the first recursive iteration:

17	16	15	14	13	E
16	15	14	13	12	13
15	14	13	12	11	12
14	13	12	11	10	11
13	12	11	10	9	10
12	11	10	9	8	9
#	#	#	#	7	8
4	3	4	5	6	7
3	2	3	4	5	6
S	1	2	3	4	5

Map after the second(in this case last) recursive iteration:

17	16	15	14	13	E
16	15	14	13	12	13
15	14	13	12	11	12
14	13	12	11	10	11
13	12	11	10	9	10
12	11	10	9	8	9
#	#	#	#	7	8
2	3	4	5	6	7
1	2	3	4	5	6
S	1	2	3	4	5

Tile list of tiles representing the final path:

Position	(1, 0)
is_obstacle	False
is_visited	False
is_start	False
is_end	False
distance	1
unknown	False

Position	(2, 0)
is_obstacle	False
is_visited	False
is_start	False
is_end	False
distance	2
unknown	False

Position	(2, 1)
is_obstacle	False
is_visited	False
is_start	False
is_end	False
distance	3
unknown	False

Position	(2, 2)
is_obstacle	False
is_visited	False
is_start	False
is_end	False
distance	4
unknown	False

Position	(2, 3)
is_obstacle	False
is_visited	False
is_start	False
is_end	False
distance	5
unknown	False

Position	(2, 4)
is_obstacle	False
is_visited	False
is_start	False
is_end	False
distance	6
unknown	False

Position	(3, 4)
is_obstacle	False
is_visited	False
is_start	False
is_end	False
distance	7
unknown	False

Position	(4, 4)
is_obstacle	False
is_visited	False
is_start	False
is_end	False
distance	8
unknown	False

Position	(5, 4)
is_obstacle	False
is_visited	False
is_start	False
is_end	False
distance	9
unknown	False

Position	(6, 4)
is_obstacle	False
is_visited	False
is_start	False
is_end	False
distance	10
unknown	False

Position	(7, 4)
is_obstacle	False
is_visited	False
is_start	False
is_end	False
distance	11
unknown	False

Position	(8, 4)
is_obstacle	False
is_visited	False
is_start	False
is_end	False
distance	12
unknown	False

Position	(9, 4)
is_obstacle	False
is_visited	False
is_start	False
is_end	False
distance	13
unknown	False

Position	(9, 5)
is_obstacle	False
is_visited	False
is_start	False
is_end	True
distance	14
unknown	False

8.4.2 Stored data

Final path as a list of tiles in .json format:

["distance": 1, "is_end": false, "is_obstacle": false, "is_start": false, "is_visited": false, "unknown": false, "x": 1, "y": 0 ,

"distance": 2, "is_end": false, "is_obstacle": false, "is_start": false, "is_visited": false, "unknown": false, "x": 2, "y": 0 , ...]

Map object used to calculate the distances in .json format:

{ "00": "distance": 0, "is_end": false, "is_obstacle": false, "is_start": true, "is_visited": false, "unknown": false, "x": 0, "y": 0 ,

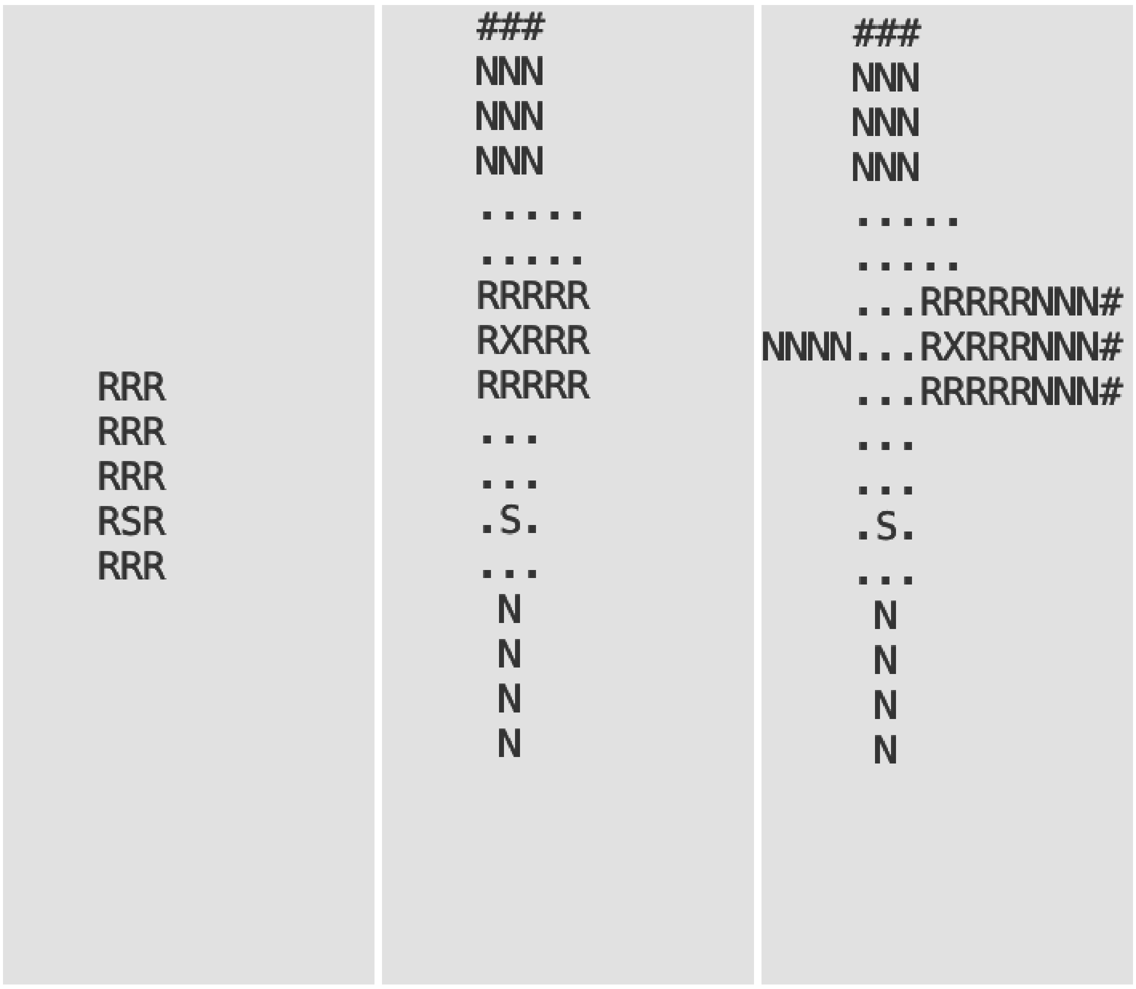
"01": "distance": 1, "is_end": false, "is_obstacle": false, "is_start": false, "is_visited": false, "unknown": false, "x": 0, "y": 1 ,

"02": "distance": 2, "is_end": false, "is_obstacle": false, "is_start": false, "is_visited": false, "unknown": false, "x": 0, "y": 2 , ... }

9 Mapping algorithm

The mapping algorithm uses the sensor data from the robot to build a map of the environment. The algorithm starts by initializing an empty map, and then as the robot moves and collects sensor data, the algorithm updates the map by adding new information or updating existing information. The resulting map can be used for the navigation and localization of the robot. Additionally, the map can be used for path planning, obstacle avoidance, and other tasks. One tile on the map represents a 6cm * 6cm area.

S: Start	N: Not visited
$\#$: Obstacle	R: Robot
• : Visited area	

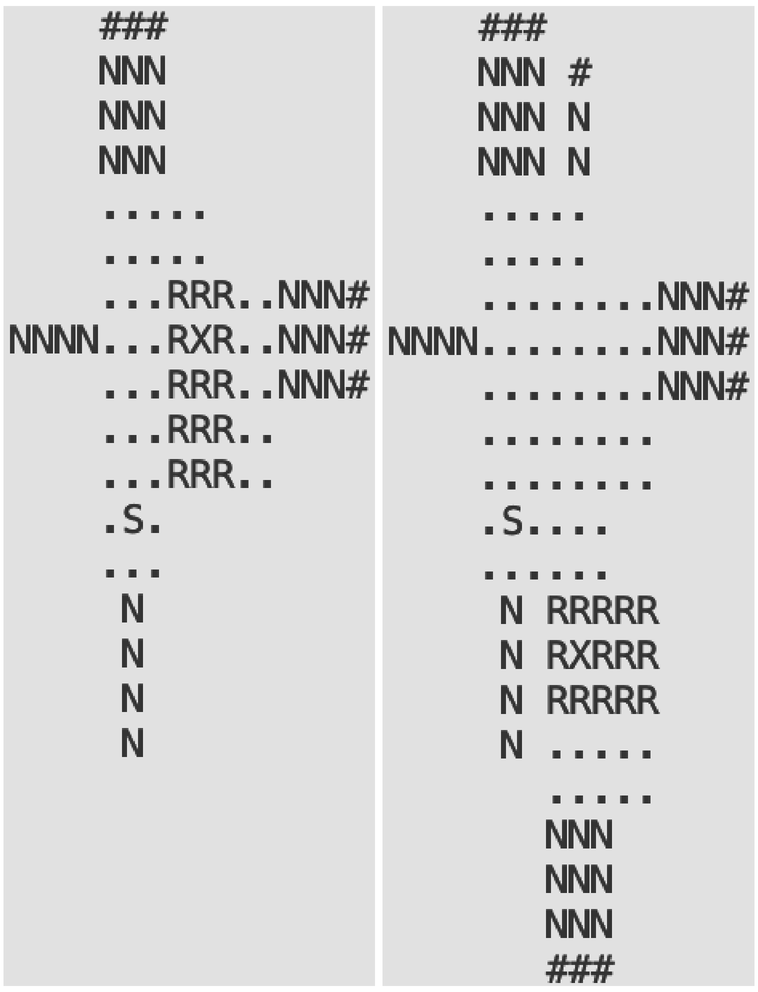


```
RRR
RRR
RRR
RSR
RRR
```

```
###
NNN
NNN
NNN
.....
.....
RRRRR
RXRRR
RRRRR
...
...
.S.
...
 N
 N
 N
 N
```

```
    ###
    NNN
    NNN
    NNN
    .....
    .....
    ...RRRRRNNN#
NNNN...RXRRRNNN#
    ...RRRRRNNN#
    ...
    ...
    .S.
    ...
     N
     N
     N
     N
```

Figure 6. Map as robot starts exploring



```
    ###
    NNN
    NNN
    NNN
    .....
    .....
    ...RRR..NNN#
NNNN...RXR..NNN#
    ...RRR..NNN#
    ...RRR..
    ...RRR..
    .S.
    ...
     N
     N
     N
     N
```

```
    ###
    NNN #
    NNN N
    NNN N
    .....
    .....
    ........NNN#
NNNN........NNN#
    ........NNN#
    ........
    ........
    .S....
    ......
     N RRRRR
     N RXRRR
     N RRRRR
     N .....
       .....
       NNN
       NNN
       NNN
       ###
```

Figure 7. Updated map

10 Testing

Manual testing was the most reliable way to test the functionality of the code, especially when testing features that cannot be tested using a simulation, such as object recognition.

10.1 Motor Test

10.1.1 Description

This test is used to check if the written software correctly controls the speed, direction, and precision of the stepper motor. The results of testing must be evaluated by a person physically.

10.1.2 Precondition

The stepper motor (JK42HS48-1204), motor controller (DRV8825), and computer (Raspberry Pi 4B) must be connected together correctly according to the manuals.

10.1.3 Assumption

All components are wired together correctly. Batteries that are used for powering the computer, motor controller, and stepper motor are fully charged. The robot and a remote computer are connected to the same network. The login credentials are known.

10.1.4 Test steps

The following steps are taken to carry out the test:

1. Put something (paper clip, masking tape, etc.) on a motor shaft to have a track of the motor's rotation.
2. Turn on the computer (Raspberry Pi 4B).
3. Turn on the motor controllers using a power switch on a robot.
4. Connect to the computer (Raspberry Pi 4B) using the SSH protocol.
5. Execute the script using "python3.11 ∽/Rover/tests/Motor_test/Motor_test.py".
6. Follow the steps specified in a terminal
7. Check if the motor rotates in a clockwise direction and the same amount of rotations as specified in the terminal.
8. Check if the motor rotates in a counter-clockwise direction and the same amount of rotations as specified in the terminal.
9. Check if the motor rotates in a clockwise direction during the high-speed test, doesn't make any concerning sounds, and maintains a stable speed.
10. Check if the motor rotates in a counter-clockwise direction during the high-speed test, doesn't make any concerning sounds, and maintains a stable speed.

10.1.5 Expected results

The motor functions as expected: the speed is stable, it can rotate in both directions and it can turn the specified amount of rotations precisely.

10.2 Drivetrain Test

10.2.1 Description

This test is used to check if the written software correctly controls the robot's drivetrain. The results of testing must be evaluated by a person physically.

10.2.2 Precondition

The stepper motor (JK42HS48-1204), motor controller (DRV8825) and computer (Raspberry Pi 4B) must be connected together correctly according to the manuals. The body parts of the robot must be correctly connected with the screws tightened. Both tracks of the robot must be tightened.

10.2.3 Assumption

All components are wired together correctly. Batteries that are used for powering the computer, motor controller, and stepper motor are fully charged. The robot and a remote computer are connected to the same network. The login credentials are known.

10.2.4 Test steps

The following steps are taken to carry out the test:

1. Put the robot on a surface that has a straight line to reference from or make a reference line yourself. The line should be parallel to the robot's tracks.
2. Turn on the computer (Raspberry Pi 4B).
3. Turn on the motor controllers using a power switch on a robot.
4. Connect to the computer (Raspberry Pi 4B) using the SSH protocol.
5. Execute the script using "python3.11 ∽/Rover/tests/Drivetrain_test/Drivetrain_test.py".
6. Follow the steps specified in a terminal
7. Check if the robot drives in a straight line with no deviations.
8. Check if the robot can drive forward and backward.
9. Check if the robot traveled the distance as specified in the script.

10.2.5 Expected results

The robot can drive in a straight line forward and backward with no deviations, it drives forwards and backward the distance specified while running the test script.

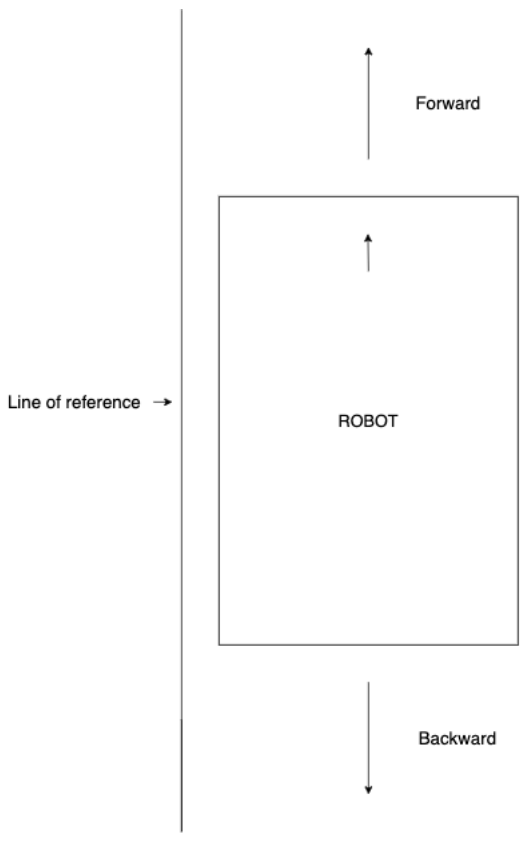


Figure 8. Drivetrain test example

10.3 Precise Turning Test

10.3.1 Description

This test is used to check if the robot can turn precisely using the accelerometer/gyroscope module (MPU6050) data. The results of testing must be evaluated by a person physically.

10.3.2 Precondition

The stepper motor (JK42HS48-1204), motor controller (DRV8825), accelerometer/gyroscope module (MPU6050) module and computer (Raspberry Pi 4B) must be connected together correctly according to the manuals. The body parts of the robot must be correctly connected with the screws tightened. Both tracks of the robot must be tightened.

10.3.3 Assumption

All components are wired together correctly. Batteries that are used for powering the computer, motor controller and stepper motor are fully charged. The accelerometer/gyroscope module (MPU6050) is calibrated correctly.

10.3.4 Test steps

1. Put the checkered paper on a plain, level surface and use masking tape to hold it down.
2. Place the robot on that paper, so the tracks are parallel to the lines on the paper.

3. Turn on the computer (Raspberry Pi 4B).
4. Turn on the motor controllers using a power switch on a robot.
5. Using the remote controller press the "ON" button, to continue the script boot.
6. Wait for 3 beeps that indicate the robot is ready to be used.
7. Using the remote controller turn on the manual mode.
8. Using the remote controller press the "ON" button.
9. Try pressing "left" or "right" button.
10. Check if the robot rotated 90 degrees (+-1° error is acceptable) in the specified direction.

10.3.5 Expected results

The robot can successfully execute 90° turns within the margin of error.

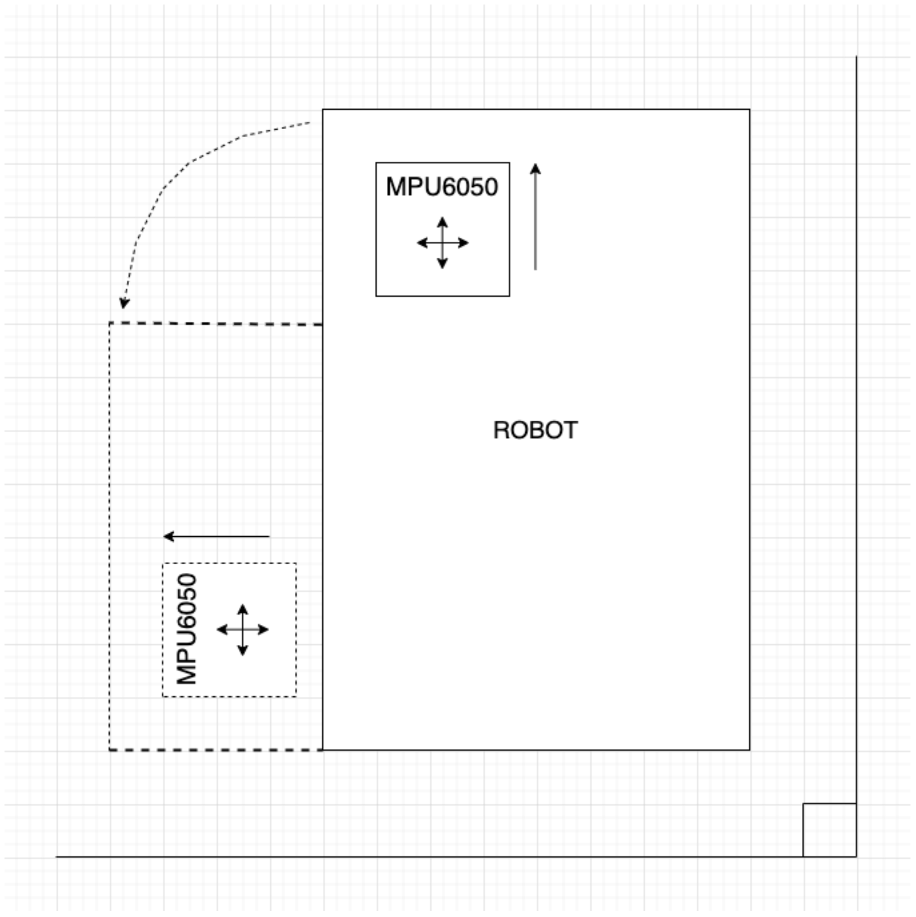


Figure 9. 90° left turn test example

10.4 Distance Sensor Test

10.4.1 Description

The purpose of this test is to ensure the proper functionality of the ultrasound distance sensors that are mounted on the robot. The results of testing must be evaluated by a person physically.

10.4.2 Precondition

The four ultrasound distance sensors(HC-SR04) and the computer (Raspberry Pi 4B) must be connected together correctly according to the manuals.

10.4.3 Assumption

All components are wired together correctly and the power bank of the computer (Raspberry Pi 4B) is fully charged. The robot and a remote computer are connected to the same network. The login credentials are known.

10.4.4 Test steps

1. Turn on the computer (Raspberry Pi 4B).
2. Connect to the computer (Raspberry Pi 4B) using the SSH protocol.
3. Execute the script using "python3.11 ~/Rover/tests/Distance_Sensor_test/Distance_Sensor_test.py".
4. Follow the steps specified in a script
5. Compare printed distances to the actual distances.

10.4.5 Expected results

The ultrasound distance sensor returns the distance which corresponds to the actual distance.

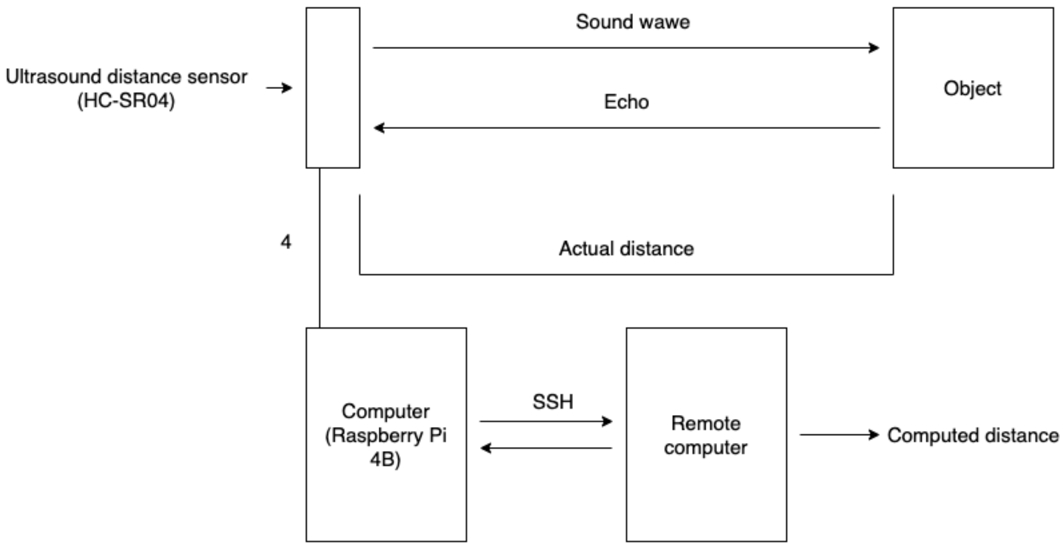


Figure 10. Ultrasound distance sensor testing example

10.5 Object Recognition Test

10.5.1 Description

The purpose of this test is to ensure the proper functionality of the USB camera that is mounted on the robot and the correct use of OpenCV module. The results of testing must be evaluated by a person physically.

10.5.2 Precondition

The USB camera and the computer (Raspberry Pi 4B) must be connected together correctly according to the manuals.

10.5.3 Assumption

All components are wired together correctly and the power bank of the computer (Raspberry Pi 4B) is fully charged. The robot and a remote computer are connected to the same network. The login credentials are known.

10.5.4 Test steps

1. Turn on the computer (Raspberry Pi 4B).
2. Connect to the computer (Raspberry Pi 4B) using the SSH protocol.
3. Execute the script using
   "python3.11 ~/Rover/tests/Computer_Vision_test/Computer_Vision_test.py".
4. Try pointing the robot to the objects that are recognizable by the script (classes of recognizable objects are printed in the terminal)
5. Check if the object classes are printed in the terminal as you are pointing the robot to recognizable objects.

10.5.5 Expected results

The object classes are printed out correctly, which means that the USB camera and OpenCV module are working correctly.

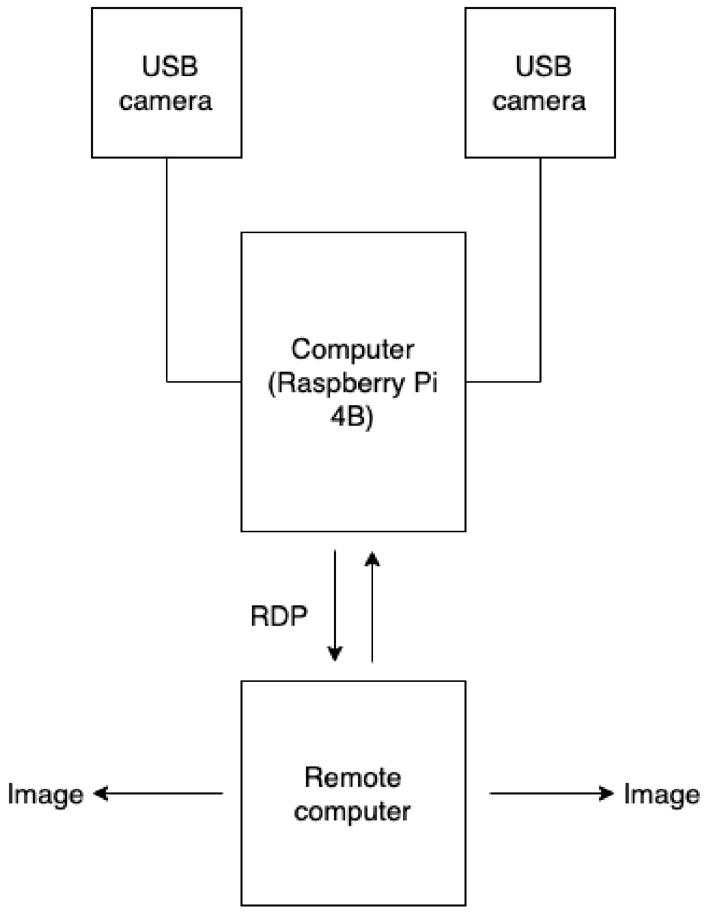


Figure 11. Camera testing example

10.6 Accelerometer / Gyroscope (IMU) Test

10.6.1 Description

The purpose of this test is to ensure that the accelerometer/gyroscope (IMU) module (MPU6050) is functioning correctly. The results of testing must be evaluated by a person physically.

10.6.2 Precondition

The accelerometer/gyroscope module (MPU6050) and the computer (Raspberry Pi 4B) must be connected together correctly according to the manuals.

10.6.3 Assumption

All components are wired together correctly and the power bank of the computer (Raspberry Pi 4B) is fully charged. The accelerometer/gyroscope module (MPU6050) is calibrated correctly. The robot and a remote computer are connected to the same network. The login credentials are known.

10.6.4 Test steps

1. Turn on the computer (Raspberry Pi 4B).
2. Connect to the computer (Raspberry Pi 4B) using the SSH protocol.
3. Execute the script using "python3.11 ~/Rover/tests/IMU_test/IMU_test.py".
4. Follow the steps specified in a script
5. Move the robot around and test if the printed-out angles of the axis are correct.

10.6.5 Expected results

The printed-out roll, pitch, and yaw values should represent actual values with little to no error.

10.7 Buzzer Test

10.7.1 Description

The purpose of this test is to ensure the proper functionality of the piezo buzzer. The results of testing must be evaluated by a person physically.

10.7.2 Precondition

The piezo buzzer (KY-012) and the computer (Raspberry Pi 4B) must be connected together correctly according to the manuals.

10.7.3 Assumption

All components are wired together correctly and the power bank of the computer (Raspberry Pi 4B) is fully charged. The robot and a remote computer are connected to the same network. The login credentials are known.

10.7.4 Test steps

1. Turn on the computer (Raspberry Pi 4B).
2. Locate and execute the testing script using the python3 command.
3. Execute the script using "python3.11 ~/Rover/tests/Buzzer_test/Buzzer_test.py".
4. Follow the steps specified in a terminal.
5. Check if you pass all the tests.

10.7.5 Expected results

All tests are passed, which means that the piezo buzzer is functioning properly.

10.8 Mapping System Test

10.8.1 Description

The purpose of this test is to ensure the proper functionality of the mapping system. The results of testing must be evaluated by a person physically.

10.8.2 Precondition

The four ultrasound distance sensors(HC-SR04), stepper motors (JK42HS48-1204) with their controllers (DRV8825), accelerometer/gyroscope module (MOU6050) and the computer (Raspberry Pi 4B) must be connected together correctly according to the manuals.

10.8.3 Assumption

All components are wired together correctly. Batteries that are used for powering the computer, motor controller, and stepper motor are fully charged. The robot and a remote computer are connected to the same network. The login credentials are known.

10.8.4 Test steps

1. Place the robot on the ground.
2. Turn on the computer (Raspberry Pi 4B).
3. Turn on the motor controllers using a power switch on a robot.
4. Execute the main script using "python3.11 ~/Rover/dev/Rover.py".
5. Using the remote controller press the "ON" button, to continue the script boot.
6. Wait for 3 beeps that indicate the robot is ready to be used.
7. Using the remote controller add "Explore to the queue"
8. Using the remote controller press the "start" button
9. After letting the robot roam around the room, press the "stop" button and check if the printed-out map corresponds to the actual layout of the room.

10.8.5 Expected results

The robot correctly maps out and updates the room layout.

10.9 Script Loading On Boot Test

10.9.1 Description

The purpose of this test is to ensure the proper functionality of the system booting process.

10.9.2 Precondition

All hardware components are connected together accordingly to the manuals.

10.9.3 Assumption

All components are wired together correctly. Batteries are fully charged. The robot and a remote computer are connected to the same network. The login credentials are known.

10.9.4 Test steps

1. Turn on the computer (Raspberry Pi 4B).
2. Wait for a beep after the green light on the microcomputer stops blinking.
3. After a beep, using the remote controller press the "ON" button, to continue the script boot.
4. Wait for 3 beeps that indicate the robot is ready to be used.

10.9.5 Expected results

The script will correctly load on system boot.

11 Related work

The decision to use the OpenCV library for object recognition was supported by a few scientific papers where the use of OpenCV in object recognition was used. A 2010 article named *"Research of Driver Eye Features Detection Algorithm Based on OpenCV"*[2] and a 2019 article named *"Monocular Vision Based UAV Target Detection and Ranging System Implemented on OpenCV and Tensor Flow"*[3] both indicate the widespread application for object recognition using the OpenCV library.

Conclusions

To conclude, Rover is an autonomous robot companion designed to roam and map room, detect and avoid obstacles, and execute tasks given by its user. These functionalities were implemented using ultrasound sensors, cameras, power components, python, and OpenCV for computer vision. While we managed to successfully implement the majority of the functional and non-functional requirements we set for the robot, there are still areas where improvement is possible, namely the implementation of additional tasks like ’follow a human’, code readability and conciseness, and freer movement and control of the robot when in manual mode, for example, the ability to turn 45°. The following improvements would provide the user a more enjoyable experience and would be more feature-rich and polished.
Future development of Rover includes freer movement for the robot when in manual mode, like turning 45°, and additional tasks for the robot to execute.

References

[1] FreeNove. 4WD Smart Car, sept 2019.

[2] S. Hu, Z. Fang, J. Tang, H. Xu, and Y. Sun. Research of driver eye features detection algorithm based on opencv. In *2010 Second WRI Global Congress on Intelligent Systems (GCIS 2010)*, volume 3, pages 348--351, Los Alamitos, CA, USA, dec 2010. IEEE Computer Society.

[3] Z. Jiang, Y. Liu, B. Wu, and Q. Zhu. Monocular vision based uav target detection and ranging system implemented on opencv and tensor flow. In *2019 18th International Symposium on Distributed Computing and Applications for Business Engineering and Science (DCABES)*, pages 88--91, Los Alamitos, CA, USA, nov 2019. IEEE Computer Society.

[4] Sunfounder. PiCar-S v2, 2019.

[5] Sunfounder. PiCar-X, 2020.