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
2022

Preface

This project was done during the 1st semester of the study programme *Information Technologies* in the specialization of Innovative studies. We chose the topic *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 have had minimal experience in, as well as the project being the only one of its kind in the project market. Having had minimal experience with robotics, the topic drew us in and raised various questions - what parts to use for the robot, how it would look like, what programming languages to use, and where to begin with such an unfamiliar yet intriguing project.

December 21, 2022

Gabrielius Drungilas

Aistė Grigaliūnaitė

Nedas Janušauskas

Adomas Jonavičius

Contents

Abstract

Today 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. 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 humans, all with the use of stereo computer vision. 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, C++, bash, and OpenCV. The robot's hardware consisted of power components, ultrasonic distance sensors, USB cameras, stepper motors and stepper motor controllers for movement, a beeper for sound signals, a gyroscope and accelerometer to sense direction, and other miscellaneous components such as resistors, capacitors, and a remote controller. The precise movement will be calibrated using an electronic compass. The room mapping will be explained and examples of test results will be presented. The robot will be able to execute the following tasks: locate a human in a room, follow a human, find an object, and produce sound signals. The task will be given to the robot by the user using a remote controller. The expected result is an autonomous robot with the described functionality.

Keywords: Autonomous Robot, Pathfinding, Room Mapping, Stereo Computer Vision, 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 žmones - visą tai įvykdoma naudojant stereo kompiuterinį matymą. Šiame darbe paaiškinamos visos tokiam robotui įgyvendinti naudojamos technologijos, įskaitant jo techninės ir programinės įrangos diagramas. Roboto funkcionalumas buvo įgyvendintas naudojant python, C++, bash ir OpenCV. Roboto techninę įrangą sudarė energijos šaltinių komponentai, ultragarsiniai atstumo jutikliai, USB kameros, žingsniniai varikliai ir žingsninių variklių valdikliai judėjimui, garsinis signalizatorius, giroskopas ir akselerometras krypčiai nustatyti ir kiti įvairūs komponentai, tokie kaip rezistoriai, kondensatoriai ir nuotolinio valdymo pultelis. Tikslus judėjimas bus kalibruojamas naudojant elektroninį kompasą. Bus paaiškintas patalpų žemėlapio kūrimas ir pateikti bandymų rezultatų pavyzdžiai. Robotas galės atlikti tokias užduotis: rasti žmogų patalpoje, sekti žmogų, surasti objektą ir skleisti garso signalus. Užduotį robotui duos vartotojas, naudodamas nuotolinio valdymo pultelį. Tikėtinas rezultatas – autonominis robotas su aprašytu funkcionalumu.

Raktiniai žodžiai: Autonominis Robotas, Kelio Ieškojimas, Kambario Žemėlapio Kūrimas, Stereo Kompiuterinė Rega, Kliūčių Vengimas

Introduction

The goal of this project was to develop an autonomous robot companion with the ability to map a room, detect and avoid obstacles, and recognize humans with the use of stereo computer vision. The start of the document will include the overview of the project, followed by the functional and non-functional requirements, as well as other implementation details. The main task of the robot is to detect and avoid obstacles using cameras and distance sensors. The robot will use a Raspberry Pi, stepper motors, multiple ultrasound distance sensors and two cameras. The precise movement will be assisted by an accelerometer / gyroscope module. The room mapping will be explained and examples of test results will be presented. The robot's hardware details and implementation will be presented in a diagram and part list. The robot's mapping algorithm will be explained in detail. The robot will use stereo computer vision to recognize and follow humans. The robot will be able to execute the following tasks: find a human in a room, follow a human, find an object, and produce sound signals. The task will be given to the robot by the user using a remote controller. Similar concepts and robot kits have been analyzed to get a better understanding of the best way to implement this project.

1 Analysis

To realize the robot companion project, we researched robots and robot kits available online, as well as referencing online videos on small robots. There were two major types of robots - those using arduinos, and those using a raspberry pi. 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 and PiCar-S v2 from Sunfounder, and FreeNove 4WD 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, a raspberry pi, lithium ion batteries and other power components. From our analysis, we decided to keep all the main components from other robots, but focusing more on camera's obstacle detection rather than relying on sensors. Instead, we will use sensors as an aide to our cameras. Our reasons for this decision were ultrasound sensors being unpredictable, as well as them being loud. Furthermore, we chose a different type of motor than other robots because the robots we researched were able to move outdoors, while our robot is meant for indoor environments.

2 Overview

2.1 Project overview

The goal of the project is to design an autonomous robot, which would be able to detect, avoid and follow objects in an indoor environment. The rover will be able to roam around the area freely and remember the layout. Ultrasound sensors will be used to detect objects and map the environment. A micro-controller is used to control the motors and receive input from the sensors.

2.2 Hardware diagram

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

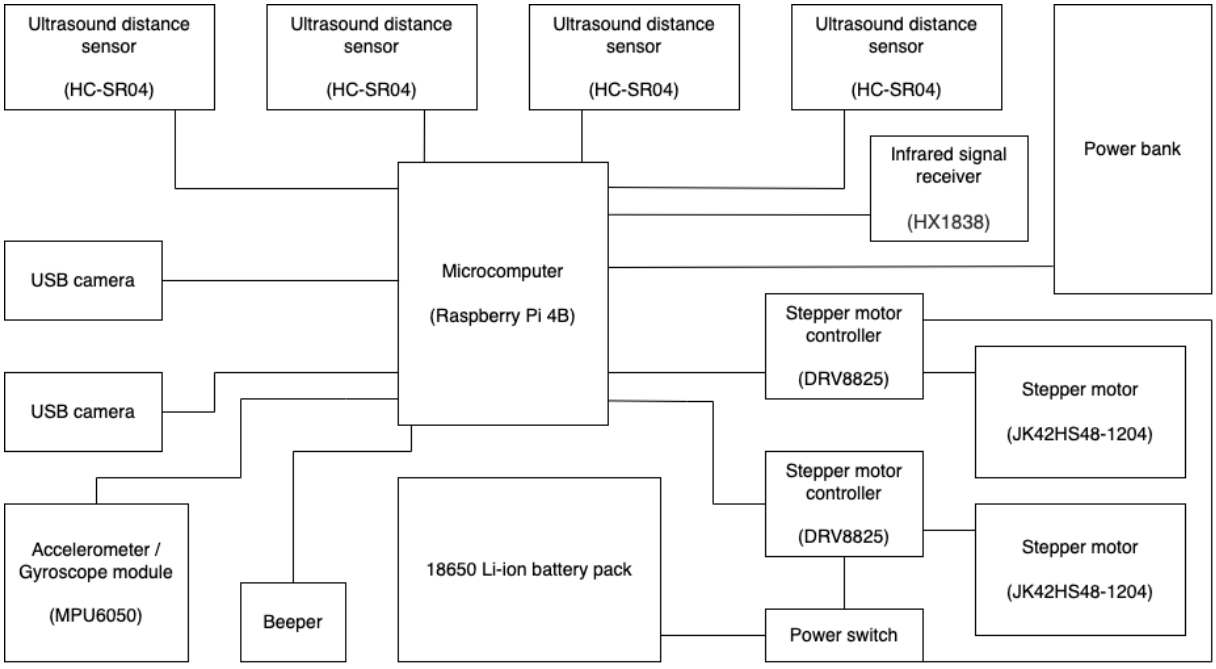

Figure 1. Hardware diagram

2.3 Context diagram

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

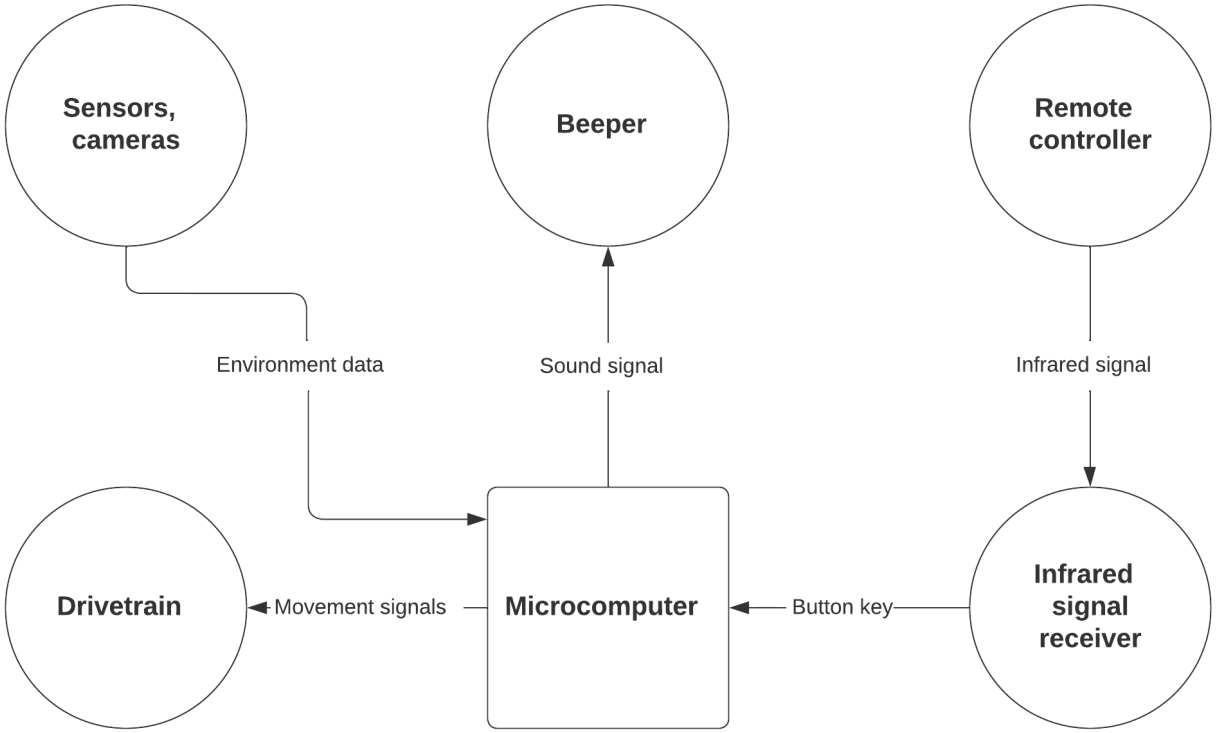

Figure 2. Context diagram

2.4 UML deployment diagram

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

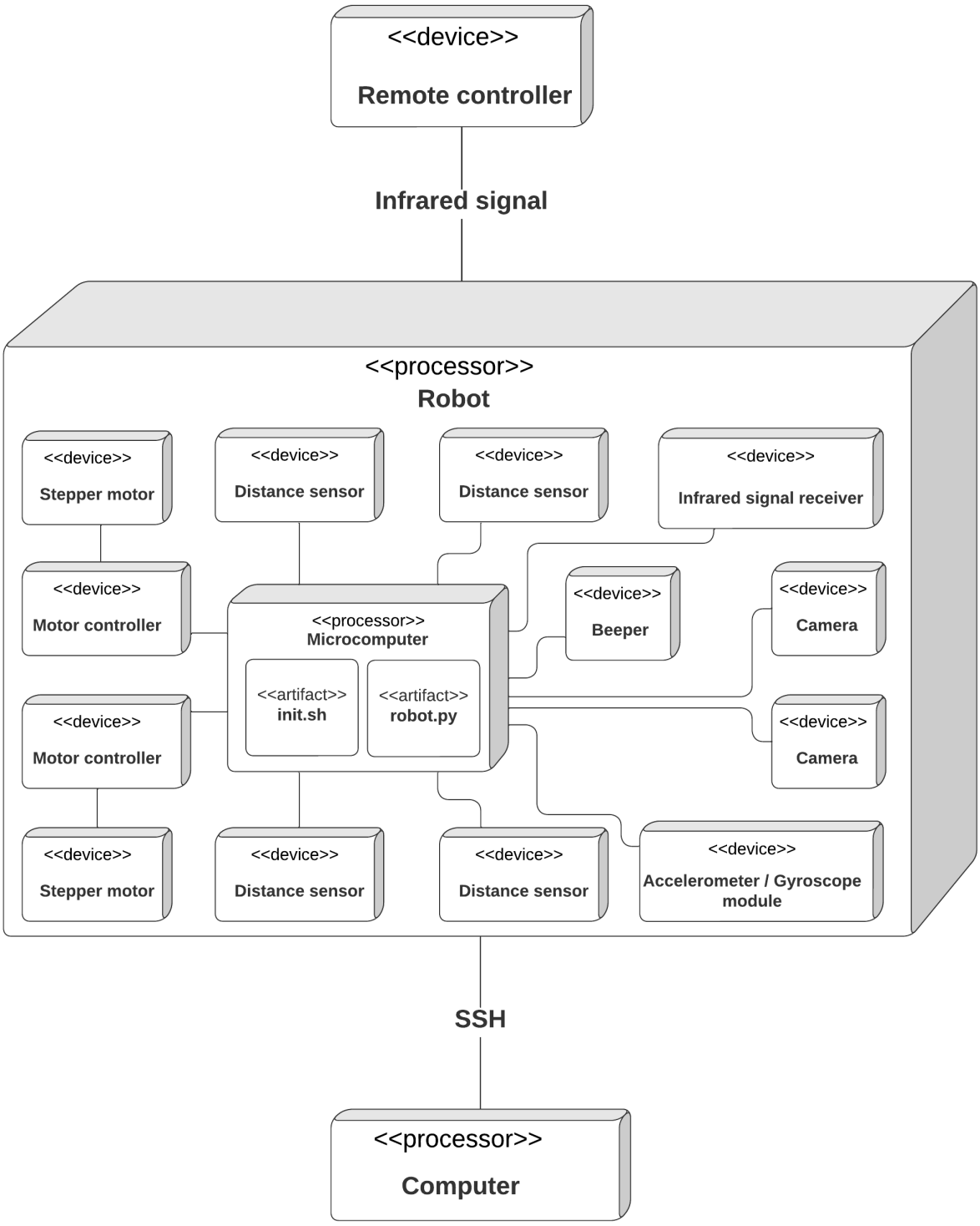

Figure 3. Deployment diagram

2.5 Hardware components

The robot will be constructed using the following hardware:

- Raspberry Pi 4 Model B (4GB RAM version)
- (4) HC-SR04 ultrasound distance sensors
- (2) USB cameras
- (2) NEMA 17 stepper motors
- (2) DRV8825 stepper motor controllers
- MPU6050 accelerator/gyroscope module
- HX1838 infrared signal receiver
- Remote controller
- Beeper
- 32GB MicroSD card
- Power switch
- Power bank
- 18650 battery holder
- (4) 18650 Li-ion batteries
- (4) 1kΩ resistors
- (4) 2kΩ resistors
- 100 μF capacitor

2.6 Software

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

2.6.1 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.

2.6.2 Software used for development

- Git and GitLab - version control.
- Webots - robot simulation environment.
- VSCode - development environment.

2.6.3 Miscellaneous software

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

3 Deliverable internals

3.1 Structural aspects

The main components of the robot are the Raspberry Pi, cameras and sensors, motors, infrared signal sensor and the beeper. 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. The beeper will be used to warn the user of errors and events, and the Raspberry Pi will be used to execute the required code.

The Raspberry Pi will run on Linux OS, and BASH will be used to run any required scripts on startup. The code for the robot will be written using python, OpenCV libraries will be used for computer vision.

3.2 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.

3.2.1 Basic robot control

A remote controller will send an infrared signal to the infrared signal receiver. This will be the main way that user is going to interact with robot. The user will be able to send signals to start different tasks like roaming freely, finding a human, 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.

3.2.2 Internal logic of USB cameras

A "camera" class is be created as an extension of the thread class to be able to keep cameras working together at the same time. A separate thread is be 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, human, 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.

3.2.3 Path finding

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

3.2.4 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 cameras.
4. Reading data from IMU.
5. Reading infrared signal data from the remote controller.

4 Functional Requirements

4.1 High Priority

- The robot will have tracks and will be able to move around the room, which includes moving forwards, backwards, and turning left and right.
- The robot will decide how to move by itself, with a user-given task.
- The robot will be able to detect obstacles (objects in the robot's path – furniture, walls, etc.) that are in front of it and behind.
- The robot will make decisions on where to move when an obstacle is detected.
- The robot can be turned off with a power switch.

4.2 Medium Priority

- The robot will remember the layout of the room.
- The robot will be able to inform the user about navigation errors with sound signals.

4.3 Low Priority

- The robot will be able to recognise and follow the user defined objects.

5 Non-Functional requirements

5.1 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.

6 Algorithm

6.1 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.

6.2 Pseudo code

```
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
```

6.3 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)

6.4 Real data

6.4.1 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

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
4	3	4	5	6	7
3	2	3	4	5	6
S	1	2	3	4	5

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)	Position	(9, 5)
is_obstacle	False	is_obstacle	False
is_visited	False	is_visited	False
is_start	False	is_start	False
is_end	False	is_end	True
distance	13	distance	14
unknown	False	unknown	False

6.4.2 Stored data

Final path as a list of tiles in .json:
["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 obj used to calculate the distances in .json:
{ "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 , ... }

7 Testing

7.1 Testing methods

The robot will be tested using two methods:

7.1.1 Manually in an indoor environment

This is a reliable way to test the functionality of the code, especially when testing features that cannot be tested using a simulation, such as computer vision. Manual testing will be our preferred method.

7.1.2 Using simulation software "Webots"

This makes testing more accessible by sharing the code in a simulated environment. With these simulations we will test small features like movement or path finding.

7.2 Motor testing

7.2.1 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.

7.2.2 Precondition

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

7.2.3 Assumption

All components are wired together correctly. Batteries that are used for powering the computer, motor controller and stepper motor are fully charged.

7.2.4 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 controller.
4. Connect to the computer (Raspberry Pi 4B) using the SSH protocol.
5. Locate and execute the testing script using the python3 command.
6. Check if the motor's speed is constant.
7. Check if the motor can turn in both directions.
8. Check if the motor shaft rotated the same amount of times as specified in the script.
9. Check if the motor doesn't make any strange noises.

7.2.5 Expected results

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

7.3 Drivetrain testing

7.3.1 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.

7.3.2 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.

7.3.3 Assumption

All components are wired together correctly. Batteries that are used for powering the computer, motor controller and stepper motor are fully charged.

7.3.4 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.
4. Connect to the computer (Raspberry Pi 4B) using the SSH protocol.
5. Locate and execute the testing script using the python3 command.
6. Check if the robot drives in a straight line with no deviations.
7. Check if the robot can drive forward and backward.

7.3.5 Expected results

The robot can drive in a straight line forward and backward with no deviations.

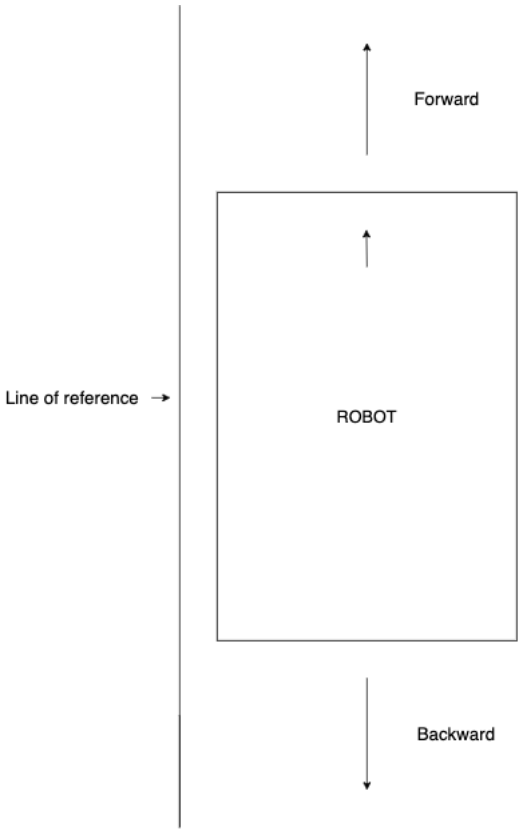

Figure 4. Drivetrain test example

7.4 Precise turning testing

7.4.1 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.

7.4.2 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.

7.4.3 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.

7.4.4 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.
5. Connect to the computer (Raspberry Pi 4B) using the SSH protocol.
6. Locate and execute the testing script using the python3 command.
7. Check if the robot turned the exact amount of degrees as specified in the script.

7.4.5 Expected results

The robot should be able to execute precise turns even if the tracks are slipping.

7.5 Distance sensor testing

7.5.1 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.

7.5.2 Precondition

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

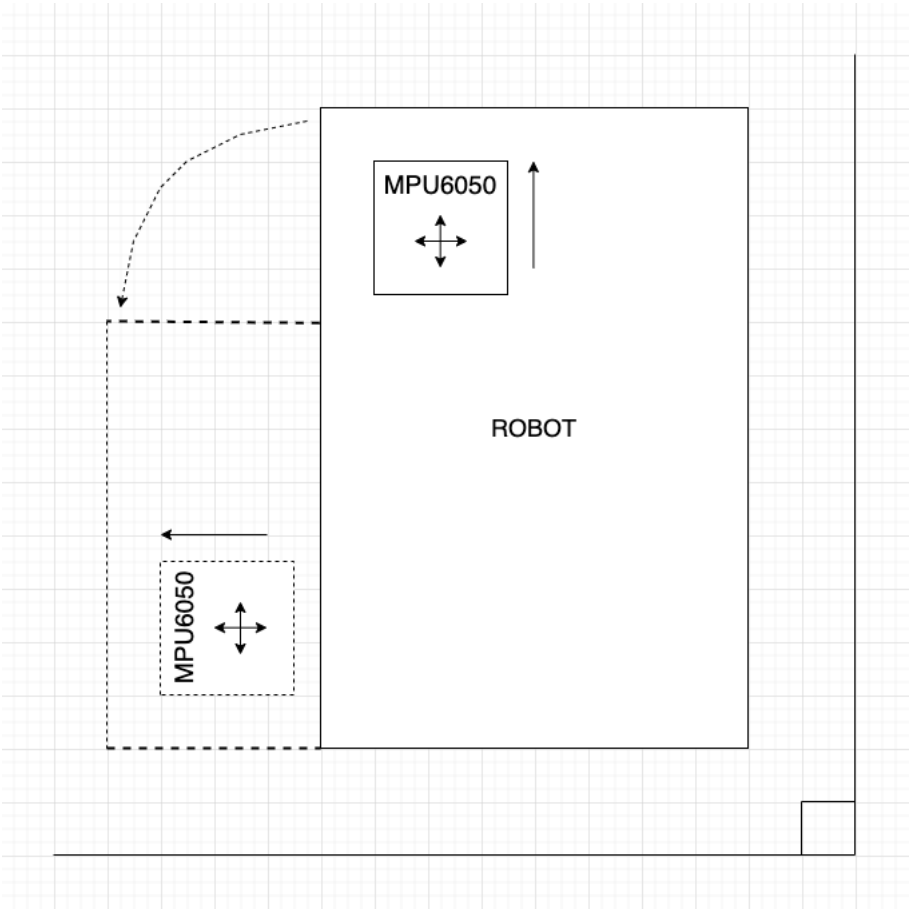

Figure 5. 90° left turn test example

7.5.3 Assumption

All components are wired together correctly and the power bank of the computer (Raspberry Pi 4B) is fully charged.

7.5.4 Test steps

1. Turn on the computer (Raspberry Pi 4B).
2. Connect to the computer (Raspberry Pi 4B) using the SSH protocol.
3. Locate and execute the testing script using the python3 command.
4. Check the ultrasound distance sensor's values, which are routinely printed out into the console.
5. Compare printed distances to the actual distances.
6. Check printed distances for irregularities.

7.5.5 Expected results

The ultrasound distance sensors are expected to routinely print out distances that are accurate to real life, with no more than a few centimeters of room for error.

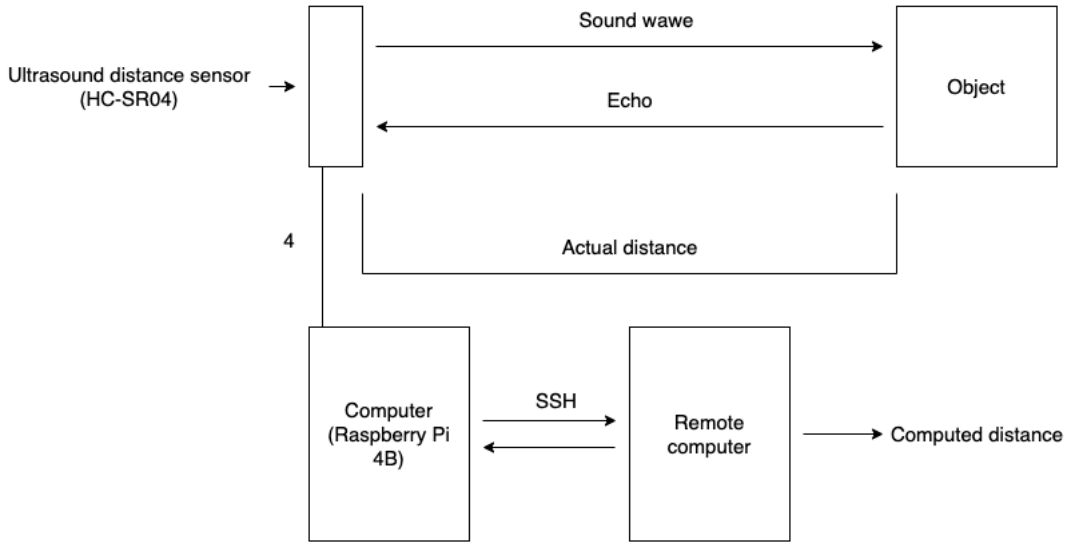

Figure 6. Ultrasound distance sensor testing example

7.6 Camera testing

7.6.1 Description

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

7.6.2 Precondition

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

7.6.3 Assumption

All components are wired together correctly and the power bank of the computer (Raspberry Pi 4B) is fully charged.

7.6.4 Test steps

1. Turn on the computer (Raspberry Pi 4B).
2. Connect to the computer (Raspberry Pi 4B) using Remote Desktop Protocol.
3. Check if both cameras are detected by the computer (Raspberry Pi 4B).
4. Record a short video with both cameras.
5. Ensure both cameras are level and the videos are of expected quality, without obvious defects.

7.6.5 Expected results

The USB cameras are expected to produce decent image quality, have nearly identical test videos, be level with each other, and produce videos without any visible abnormalities.

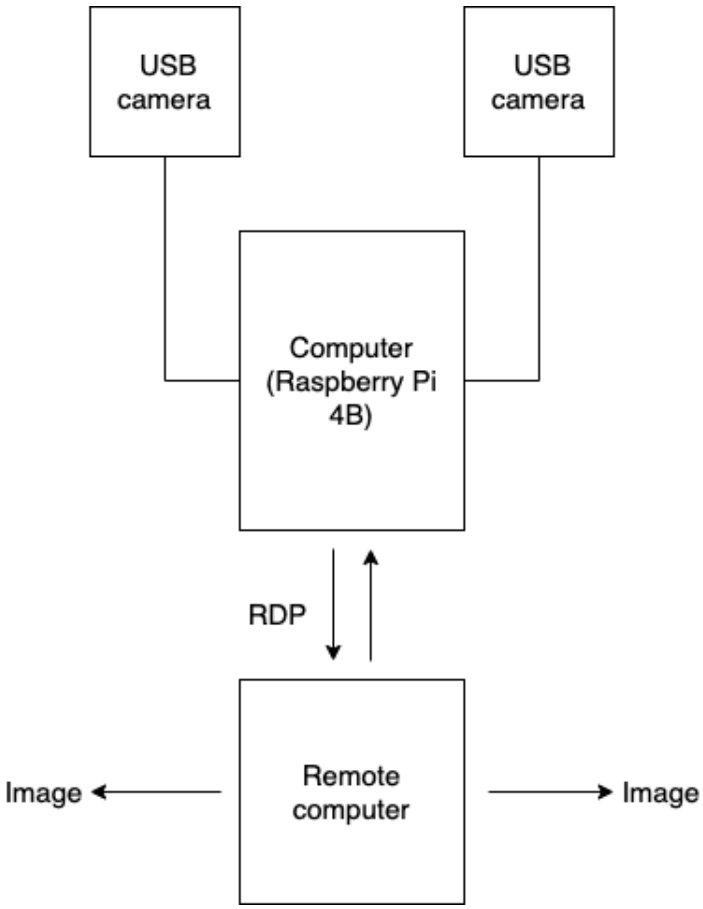

Figure 7. Camera testing example

7.7 Object detection testing

7.7.1 Description

The purpose of this test is to ensure the proper functionality of the object detection library and functions. The results of testing must be evaluated by a person physically.

7.7.2 Precondition

The two USB cameras and the computer (Raspberry Pi 4B) must be connected together correctly according to the manuals. The OpenCV library is loaded and used.

7.7.3 Assumption

All components are wired together correctly and the power bank of the computer (Raspberry Pi 4B) is fully charged.

7.7.4 Test steps

1. Turn on the computer (Raspberry Pi 4B).
2. Connect to the computer (Raspberry Pi 4B) using Remote Desktop Protocol.
3. Locate and execute the testing script using the python3 command.

4. Ensure that cameras can detect and recognize objects.

7.7.5 Expected results

The USB cameras working together with the OpenCV library are expected to recognize objects with accuracy.

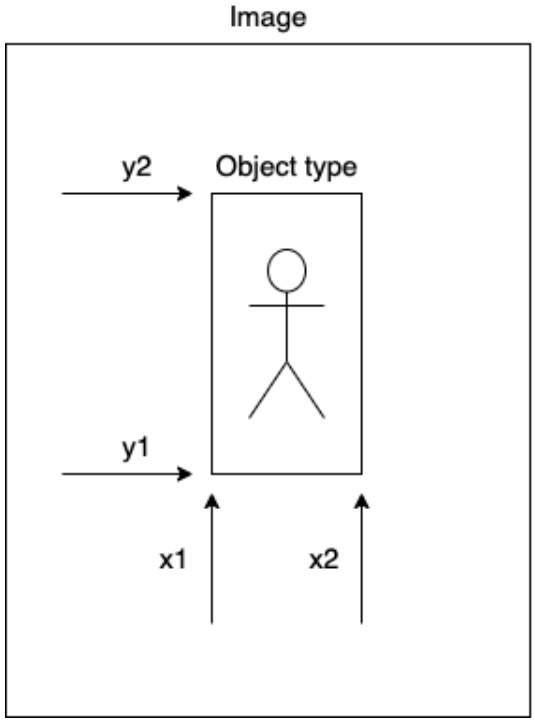

Figure 8. Object detection testing example

7.8 Accelerometer / Gyroscope testing

7.8.1 Description

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

7.8.2 Precondition

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

7.8.3 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.

7.8.4 Test steps

1. Turn on the computer (Raspberry Pi 4B).

2. Connect to the computer (Raspberry Pi 4B) using the SSH protocol.
3. Locate and execute the testing script using the python3 command.
4. Move the robot around and test if the printed-out angles of the axis are correct.

7.8.5 Expected results

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

7.9 Beeper testing

7.9.1 Description

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

7.9.2 Precondition

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

7.9.3 Assumption

All components are wired together correctly and the power bank of the computer (Raspberry Pi 4B) is fully charged.

7.9.4 Test steps

1. Turn on the computer (Raspberry Pi 4B).
2. Connect to the computer (Raspberry Pi 4B) using the SSH protocol.
3. Locate and execute the testing script using the python3 command.
4. Listen for the sound signal of the expected length and beeping repetitions.

7.9.5 Expected results

The beeper is expected to make clear and audible beeps.

7.10 Mapping system testing

7.10.1 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.

7.10.2 Precondition

The four ultrasound distance sensors(HC-SR04), 2 USB cameras, 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.

7.10.3 Assumption

All components are wired together correctly. Batteries that are used for powering the computer, motor controller and stepper motor are fully charged.

7.10.4 Test steps

1. Place the robot on the ground.
2. Turn on the computer (Raspberry Pi 4B).
3. Turn on the motor controllers.
4. Connect to the computer (Raspberry Pi 4B) using the SSH protocol.
5. Locate and execute the testing script using the python3 command.
6. After letting the robot roam around the room, check if the printed-out map corresponds to the actual layout of the room.

7.10.5 Expected results

The robot correctly maps out and updates the room layout.

Conclusions and Recommendations

To conclude ...