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

Problem-Based Project

24-Hour Task

Done by:
Gabrielius Drungilas
Aistė Grigaliūnaitė
Nedas Janušauskas
Adomas Jonavičius

Supervisor:
dr. Linas Bukauskas

Vilnius
2022

Contents

1 Algorithm

1.1 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.

1.2 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
```

1.3 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)

1.4 Real data

1.4.1 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

1.4.2 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 , ... }

2 Testing

2.1 Motor testing

2.1.1 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.

2.1.2 Precondition

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

2.1.3 Assumption

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

2.1.4 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 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.

2.1.5 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.

2.2 Drivetrain testing

2.2.1 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.

2.2.2 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.

2.2.3 Assumption

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

2.2.4 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.
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.

2.2.5 Expected results

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

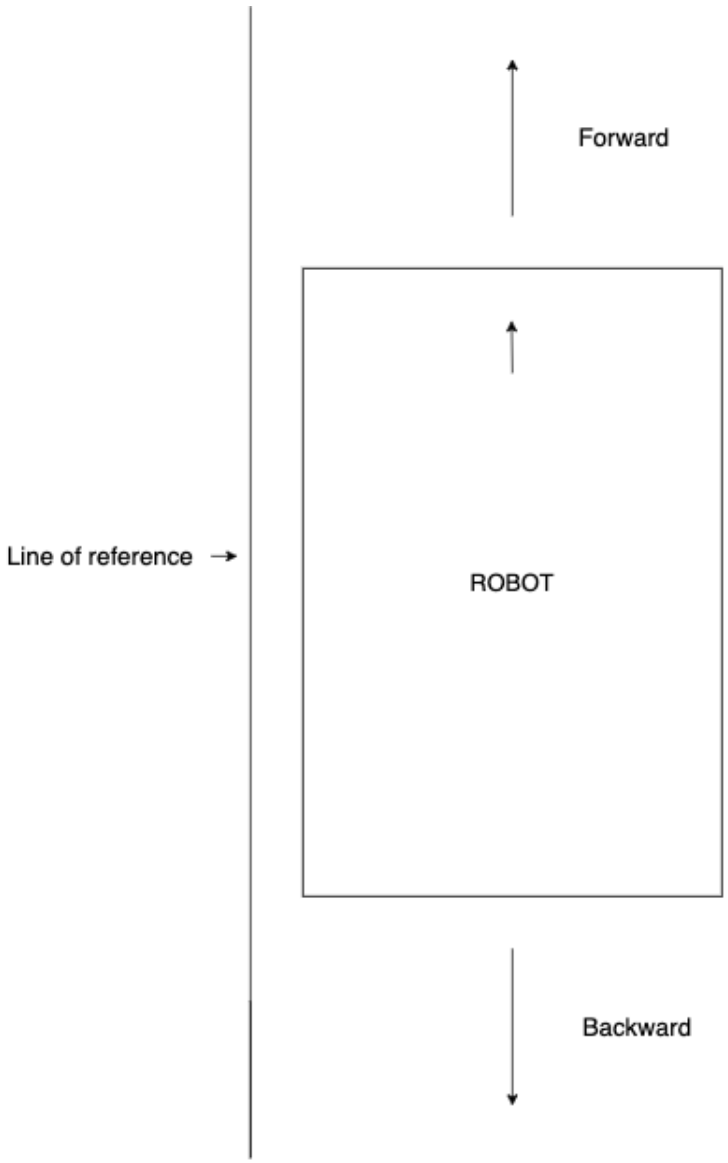

Figure 1. Drivetrain test example

2.3 Precise turning testing

2.3.1 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.

2.3.2 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.

2.3.3 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.

2.3.4 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.
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.

2.3.5 Expected results

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

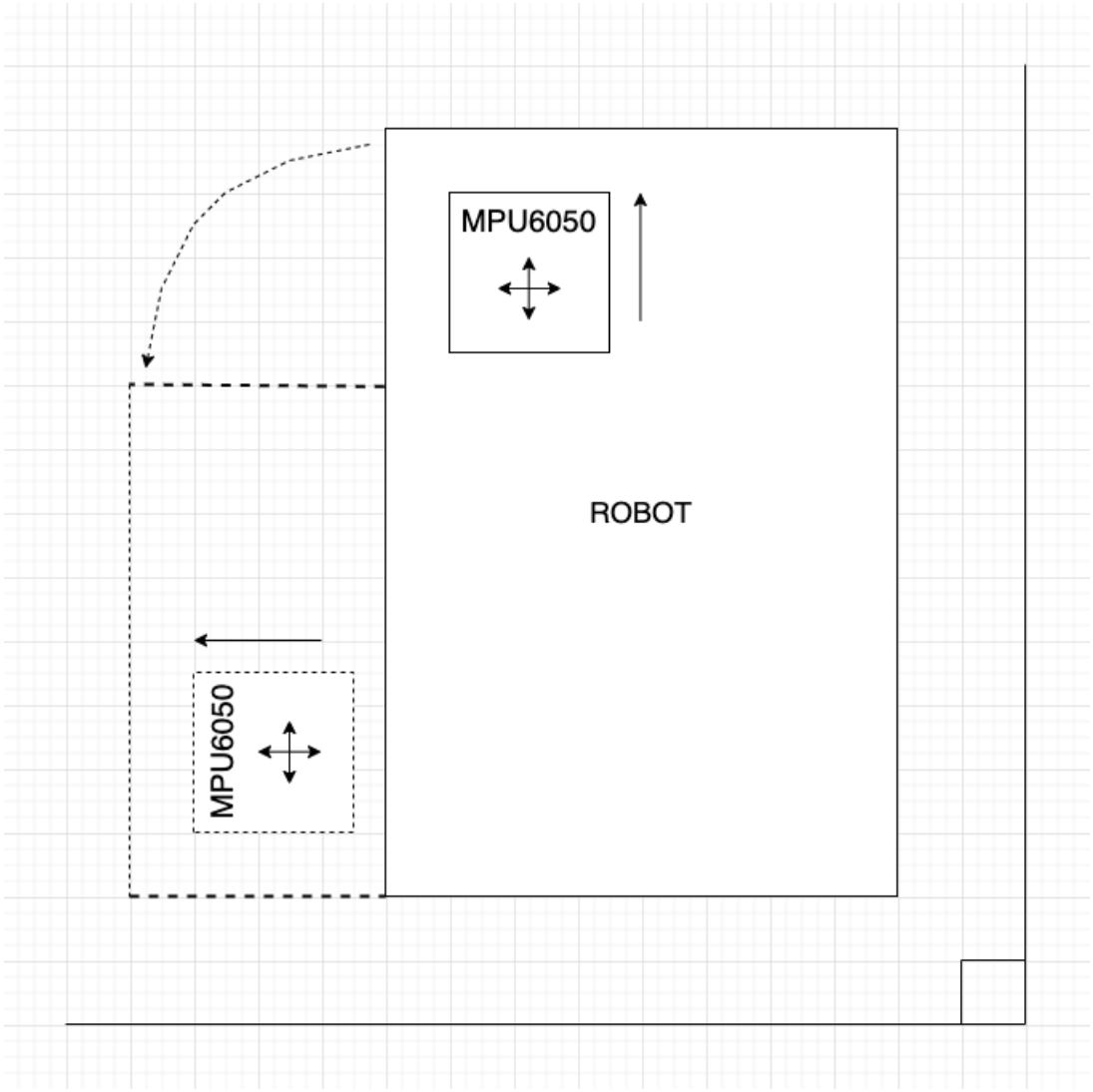

Figure 2. 90° left turn test example

2.4 Distance sensor testing

2.4.1 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.

2.4.2 Precondition

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

2.4.3 Assumption

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

2.4.4 Test steps

1. Turn on the computer (Raspberry Pi 4B).
2. Connect to the computer (Raspberry Pi 4B) using the SSH protocol.
3. 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.

2.4.5 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.

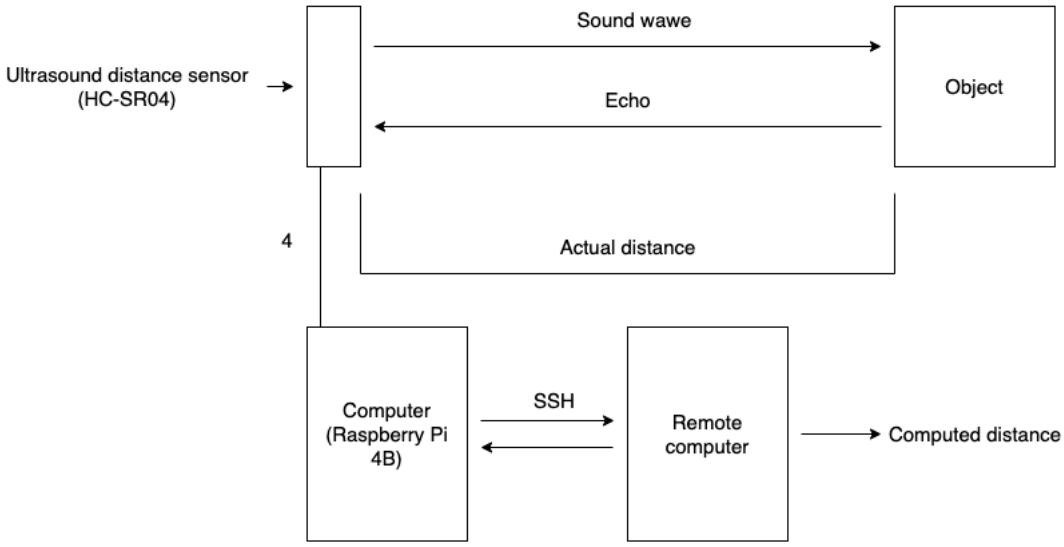

Figure 3. Ultrasound distance sensor testing example

2.5 Camera testing

2.5.1 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.

2.5.2 Precondition

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

2.5.3 Assumption

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

2.5.4 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.

2.5.5 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.

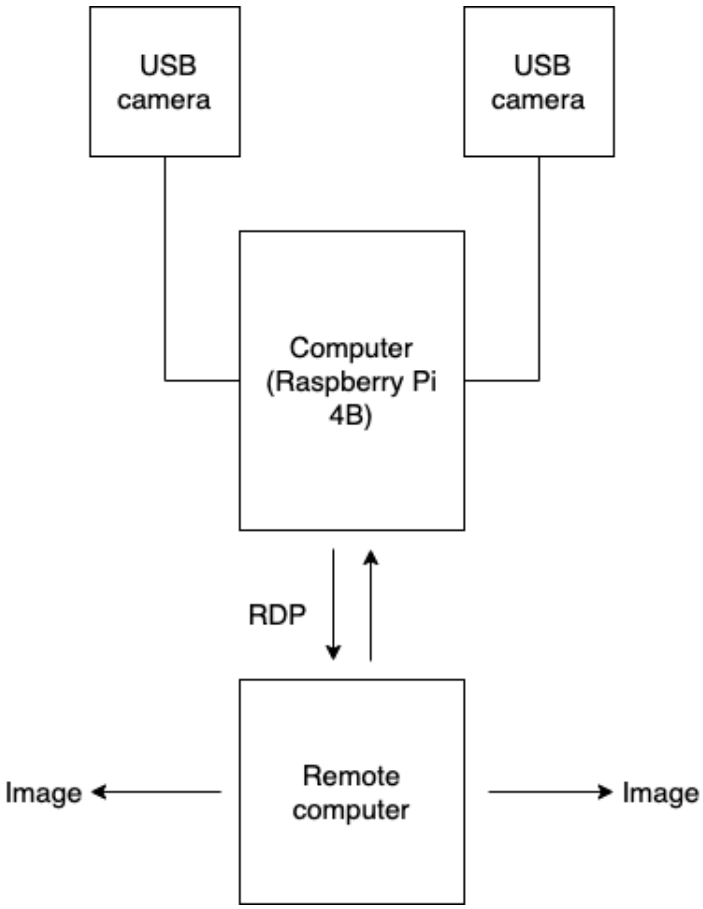

Figure 4. Camera testing example

2.6 Object detection testing

2.6.1 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.

2.6.2 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.

2.6.3 Assumption

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

2.6.4 Test steps

1. Turn on the computer (Raspberry Pi 4B).
2. Connect to the computer (Raspberry Pi 4B) using Remote Desktop Protocol.
3. Locate and execute the testing script using the python3 command.
4. Ensure that cameras can detect and recognize objects.

2.6.5 Expected results

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

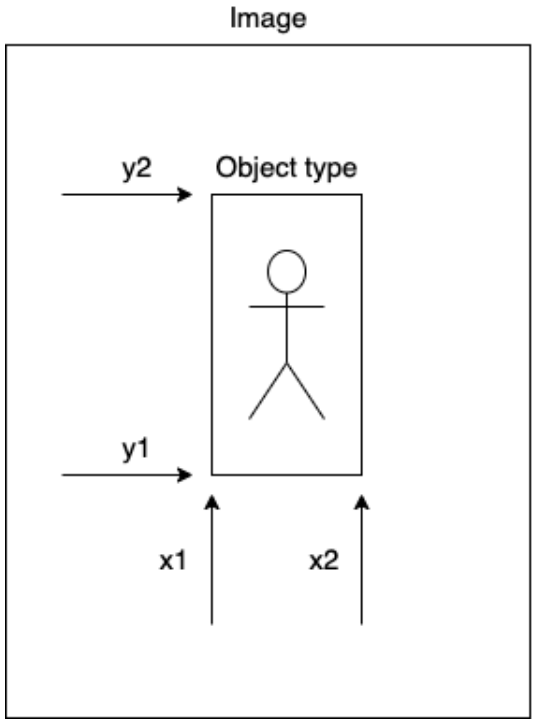

Figure 5. Object detection testing example

2.7 Depth extraction testing

2.7.1 Description

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

2.7.2 Precondition

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

2.7.3 Assumption

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

2.7.4 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. Compare the printed distance from an obstacle to the actual distance.

2.7.5 Expected results

The depth extraction system outputs the expected distance from an object with little to no error.

2.8 Accelerometer / Gyroscope testing

2.8.1 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.

2.8.2 Precondition

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

2.8.3 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.

2.8.4 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.

2.8.5 Expected results

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

2.9 Beeper testing

2.9.1 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.

2.9.2 Precondition

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

2.9.3 Assumption

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

2.9.4 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.

2.9.5 Expected results

The beeper is expected to make clear and audible beeps.

2.10 Mapping system testing

2.10.1 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.

2.10.2 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.

2.10.3 Assumption

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

2.10.4 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.

2.10.5 Expected results

The robot correctly maps out and updates the room layout.