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 17, 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, capacitators, 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.

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.

Introduction

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 Functional requirements

To realize the project's vision the following functional requirements have been set:

- The robot will be able to move around the room, which includes moving forwards, backwards, and turning left and right.
- The robot will decide where to move by itself, without user input.
- 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.
- If the robot cannot move due to an obstacle or error, it will inform the user using a sound signal.

3 Non-functional requirements

The robot will meet the following non-functional requirements:

- The robot will be operational until it runs out of power.
- The expected battery life is 2 hours.
- The robot will be easy to operate - it will be turned on and off with a button.

4 Implementation

4.1 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
- Power switch
- Power bank
- (4) 18650 batteries
- 18650 battery holder
- (4) 1kΩ resistors
- (4) 2kΩ resistors
- (2) 100 μF capacitors
- Beeper
- 32GB MicroSD card
- Remote controller
- Accelerator/gyroscope

4.2 Hardware diagram

The robot's named components will follow the hardware diagram provided in Figure 1.

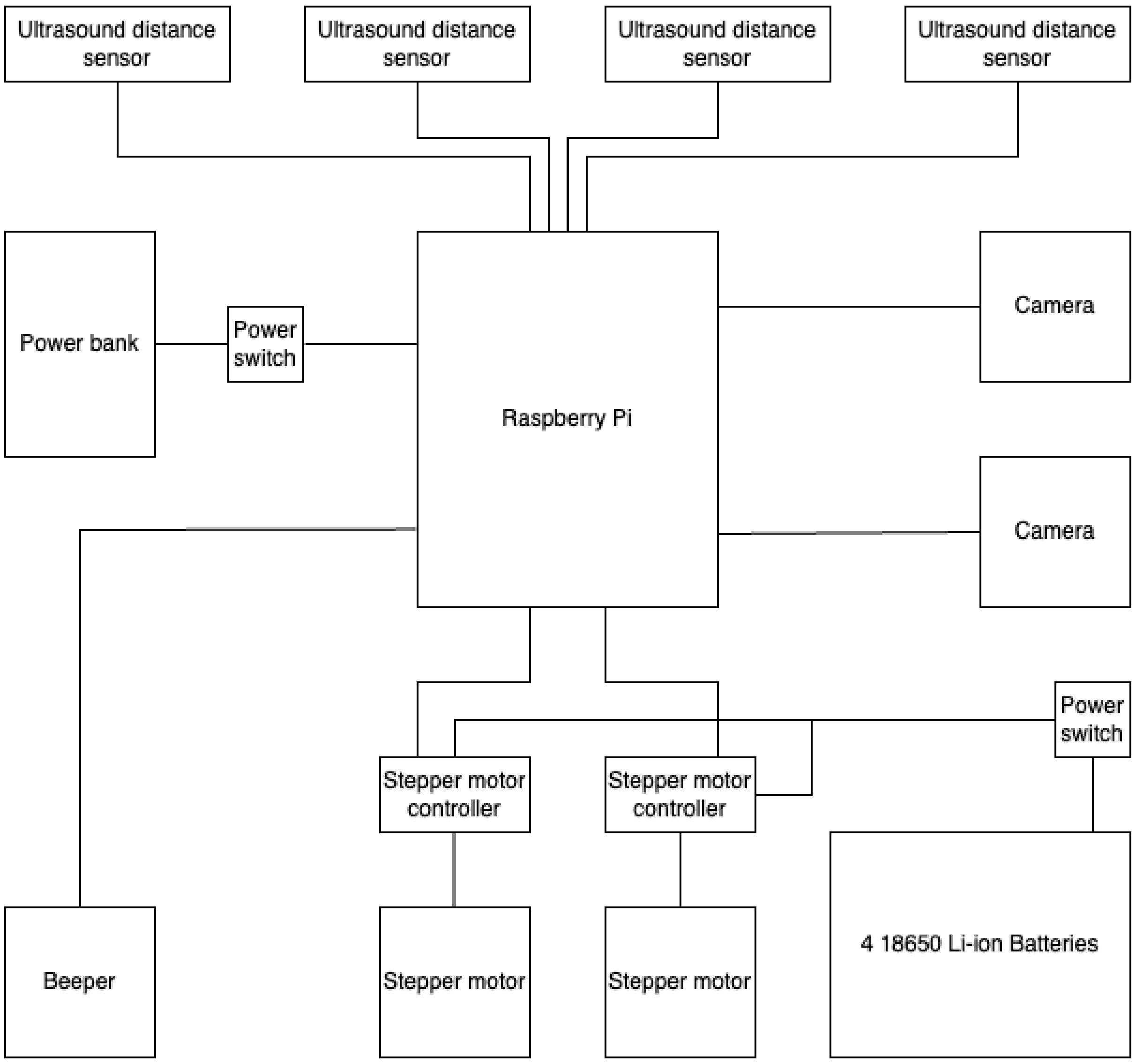

Figure 1. Hardware diagram

4.3 Software

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

- Linux OS
- Python
- OpenCV
- BASH
- Git

5 Algorithm

5.1 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.

5.2 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
```

5.3 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)

5.4 Real data

5.4.1 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

5.4.2 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 , ... }

Conclusions and Recommendations