EECE 4376

Final Project Report

Zakariyya Al-Quran
Edwin Campbell
Michael Delaney

December 9, 2022

User's Manual

Introduction:

This project uses a Raspberry Pi robot car to traverse a room and measure its dimensions. The main program follows the walls of a room until it has reached the starting point, then terminates. The program stores its readings of points to the local file system to allow processing later. The data is also streamed over the local network to any listeners running the second optional program to create a 2D mapping of the room in real time.

Program Overview:

The main python program runs on the robot and uses multiple concurrent threads to enable the robot to navigate the room while reading its surroundings. The second python program runs on another device on the network and listens to the data being published by the robot, plotting the points on a simple GUI for the user to visualize. The programs are available on GitHub at: https://github.com/zakarlyya/mapping-room-robot/

Configuring the Raspberry Pi and Robot:

To set up a Raspberry Pi to run the program in this project, you can use the following steps:

1. Install Python 3 and the necessary libraries by running the following command:

```
sudo apt-get install python3 && pip3 install pyzmq pyqt5 pyqtgraph numpy
```

2. Construct the robot including the motors and ultrasonic sensor according to the manufacturer's instructions. Documentation can be found here: https://github.com/Freenove/Freenove_4WD_Smart_Car_Kit_for_Raspberry_Pi
3. Configure the Raspberry Pi by setting up WiFi connectivity. Ensure the Pi and the computer are on the same WiFi network.
4. Obtain the Pi's wireless IP address and connect to the Pi over SSH.

Starting the Program:

1. Download or clone the main program (main.py) to the Raspberry Pi. Also, download the server program (server.py) to another computer.
2. Start the server program on the computer and enter the Pi's address when prompted:

```
python3 server.py
```

3. Run the main program, using the command below:

```
python3 main.py
```

   The main program should now begin running on the robot. The server program on the other device will be listening for data to be sent over and graphed

Running the Program:

Upon start, the robot must calibrate itself using the ultrasonic sensor. After starting the main program use the following steps to begin the mapping process:

1. Place the robot facing towards a wall of the room to map at about 1-3 feet away.
2. Enter "START" to calibrate the sensor and motors. The robot will then calibrate itself and move closer to the wall and prompt the user with “GO” when it is ready to proceed.
3. Enter "GO" to start the mapping process. The robot will map the room by turning left to be parallel to the wall. The head will begin to pan and collect data on the surroundings. While the robot is panning, the data collected (nearby walls) are converted to absolute points that are stored and broadcast to any devices running the second program.

   The points will be displayed continuously on the server program throughout the mapping process showing the 2D image of the room as it is being explored.
4. Once the robot has panned once entirely, it will decide to either move forward or turn depending on how far away the robot is from the nearest wall. This process will repeat until all of the room’s walls have been traversed.
5. The program will either terminate upon completion of the mapping or if the user sends the “STOP” signal manually from the main program terminal.

Troubleshooting:

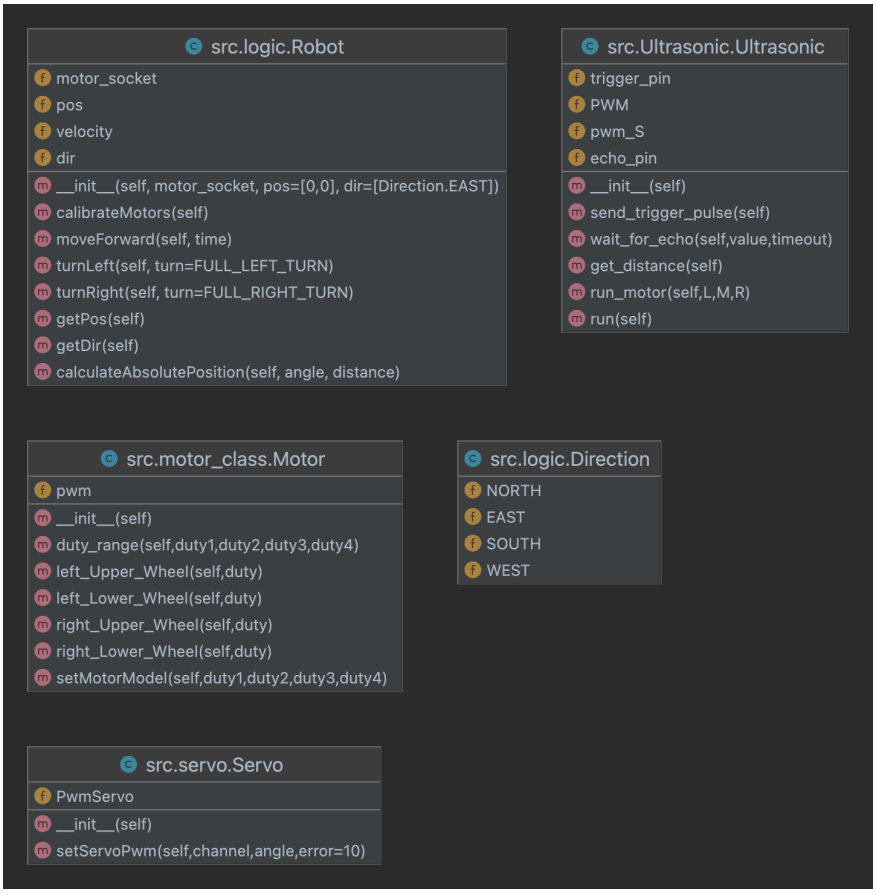
To troubleshoot potential issues with the robot or program, consider the following points:

- The robot's turning values may need to be adjusted if the robot does not turn properly (90 degrees around walls). How much the robot turns will depend on the motors, batteries, wheels, surface, etc. The default value has been tested on both hardwood and carpet flooring and can be found in the appendix.
- Increasing the number of sensor measurements can improve the accuracy of the sensor readings but will slow down the robot’s total traversal time.
- Erroneous data may be measured due to ultrasonic sensor interference or errors in the robot's position updates. This can be caused by the accumulation of errors over time. To avoid this, regularly check and calibrate the sensor and the robot's position updates.
- If the robot batteries are low, then the processor will undervolt and the power supplied will be insufficient to map the room. To prevent this, use charged batteries.
- The robot can operate independently of the secondary program, but ensure that the programs are both exited before starting again to prevent scrambled data.

Technical Manual

- **Describes technical solution of your project**
  - **principle of operations**
  - **design diagrams**
  - **explanation of the design for engineer**
- **INCLUDE UML(CLASS), CONTEXT, AND DFD DIAGRAMS**

The main thread is responsible for setting up and starting the program, while the logic thread handles the communication with other threads and uses sensor data to track the robot's position and calculate the positions of objects in the room. The sensor thread reads the ultrasonic sensor and sends out information, the motors thread moves the robot, and the server thread receives data from the robot and displays a working image of the room. All of these threads use ZMQ sockets for communication.



Once set up, the sensor continuously pans to read the surroundings and take multiple readings to remove any outliers. The robot then uses a voting system to determine whether to turn left, right, or continue moving forward when it reaches the end of a wall. The robot uses math to map the

positions of objects in the room to absolute points on a coordinate plane, and continues this process until all walls in the room have been mapped.

The logic code for a robot implements the features which allow the device to follow a wall, record its movements, and terminate when it returns to its starting position. It does this by receiving sensor data from the ultrasonic sensor and transmitting motor movement requests to the motors. It uses the ZMQ messaging library for communication between threads.

Logic:
Here are the steps it takes to do this:

1. Import necessary libraries and class components
2. Define the Direction enum and the logic_main function
3. Create a zmq context and three zmq sockets: one for receiving start/stop signals from main.py, one for sending motor movement requests and receiving replies, and one for subscribing to sensor data
4. Create an instance of the Robot class, calibrate its motors, and wait for the start signal from main.py
5. Turn the robot to face the wall and wait for the motors to finish moving
6. Set the robot's direction and starting position to (0,0)
7. Start the sensor thread, which will publish sensor data to the sensor_socket
8. Create a zmq publisher for transmitting points (coordinates where the robot detects an object) to main.py
9. Create a zmq subscriber for subscribing to main.py's stop signals
10. Enter a loop where the robot:
    1. Request sensor data from the sensor thread
    2. Wait for the sensor data to be received
    3. If any new points are detected, add them to the points list and transmit them to main.py
    4. If an object is detected in front of and to the right of the robot, turn left. Otherwise, move forward and turn right until an object is detected on the right
    5. Repeat until the robot reaches (0,0)
11. Stop the robot and close the sockets when main.py sends a stop signal.

Motors:
This code is a Python script that receives motor movement requests from logic.py and executes them. It does this by:

1. Importing necessary libraries and the Motor class from motor_class.py
2. Defining the motors_main function
3. Creating a zmq context and socket for REQ/REP communication with logic.py
4. Creating an instance of the Motor class
5. Entering a loop where it waits for a request from logic.py, parses the request message, and executes the corresponding motor movement (e.g. move forward, turn left, turn right)

6. Sending a reply message indicating the movement was successful
7. Repeating the loop until a stop signal is received.

Sensor:
This code is a Python script that continuously pans the robot's ultrasonic sensor back and forth, measures the distance to objects using the sensor, and sends the measured distances to logic.py through a zmq PUB socket. It does this by:

1. Importing necessary libraries and the Servo and Ultrasonic classes
2. Defining the sensor_main function
3. Creating a zmq context and PUB socket for publishing sensor data to logic.py
4. Creating an instance of the Servo and Ultrasonic classes
5. Panning the sensor head straight ahead and setting the default values for the sensor angle and number of readings per angle
6. Entering a loop where it:
    1. Takes multiple readings from the ultrasonic sensor to reduce noise
    2. Removes outliers from the readings and calculates the average distance
    3. Sends the average distance and the current angle to logic.py through the PUB socket
    4. Updates the angle and pans the sensor in the opposite direction
    5. Repeats the loop until a stop signal is received.

Server:
This code is a Python script that receives data published by the robot's logic.py and sensor.py scripts and displays it on a live scatter plot in real-time.

It does this by:

1. Importing necessary libraries including PyQt5, pyqtgraph, numpy, and zmq
2. Defining a *MyWidget* class that extends pyqtgraph's *GraphicsLayoutWidget* to create a custom widget for displaying the scatter plot
3. Using the *__init__* method to initialize the widget, set the layout and title, initialize the plot, and create a timer for refreshing the data
4. Defining the *getNewData* method that is called on each timer tick. This method receives new data published by the robot, updates the plot with the new data, and updates the plot
5. Defining the *main* method that creates a ZMQ context and connects to the robot's PUB/SUB socket, subscribes to all messages, and creates an instance of *MyWidget*
6. Running the *main* method when the script is run

The *MyWidget* class creates a window containing a scatter plot of the points detected by the robot's ultrasonic sensor. The position of the robot is also displayed on the scatter plot in real-time as it moves. The data published by the robot is received over a ZMQ PUB/SUB socket

and is plotted on the scatter plot as it is received. When the robot is finished mapping, it sends a done message and the script closes the window and exits.

Main:

This code launches multiple threads for logic, motors, and server. It then waits for user input, and sends a "START" or "STOP" signal to the logic thread based on the input. The code also logs messages using the logging library, and uses the ZMQ library to communicate with the logic thread using a REQ socket. Once the threads are started and the input is received, the code waits for the threads to terminate before closing the sockets and terminating.

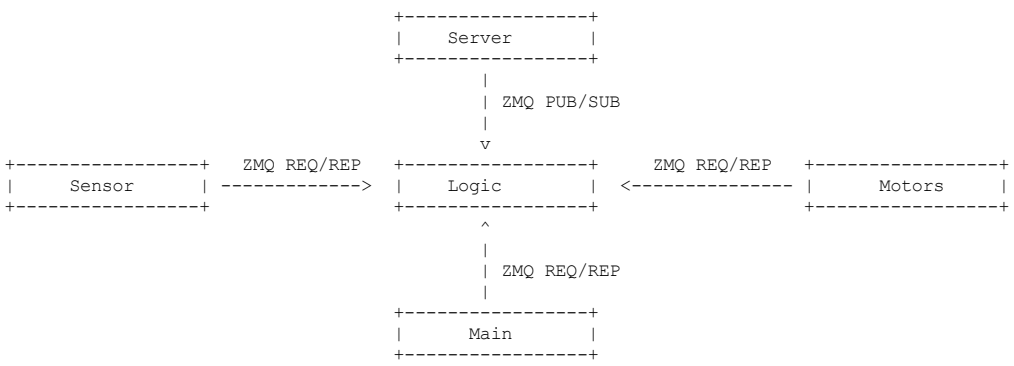


Figure 2: Dataflow Diagram

In the diagram, the *Server* receives data published by the *Logic* and *Sensor* scripts through a ZMQ PUB/SUB socket and displays it on a live scatter plot. The *Logic* script receives sensor data from the *Sensor* script through a ZMQ PUB socket and sends motor movement requests to the *Motors* script through a ZMQ REQ/REP socket. The *Motors* script receives motor movement requests from the *Logic* script through a ZMQ REQ/REP socket and executes them.

Project Personnel

Overview:

The roles and responsibilities of the group members varied depending on their individual skills and expertise. In general, all members of the group were responsible for contributing to the project design, implementation, and testing by providing input and ideas during group meetings, working on project-specific tasks, and testing the robot regularly. Given the scope and duration of this project, it was important for all members to communicate effectively and work together in order to ensure the success of the project.

Additionally, all members were expected to attend weekly meetings and actively participate in discussions about the project. These meetings provided an opportunity for members to share their progress, discuss any challenges they were facing, and come up with solutions to any problems that may have arisen. It was important for every member to attend these meetings and contribute to the discussions to ensure that the project would be completed successfully.

Responsibilities:

The group members who worked on this project were Zakariyya Al-Quran, Michael Delaney, and Edwin Campbell. Each group member was expected to construct, setup, and test their Raspberry Pi 4B and Freenove 4WD robot when the project began. Then, tasks were delegated to each member. Edwin worked with the team to lay out the project goals. Michael worked on the motor thread, concurrency, and logic design. Zakariyya worked on the sensor thread, graphical user interface, networking, and logic as well. All group members were expected to test all the code. The program was mostly tested and then demoed on Zakariyya's robot. Members also had to contribute to the project documentation including multiple presentations and a final report.

Appendix

Parts List:

- Freenove Smart Car Robot with 4WD and ultrasonic sensor
  - https://www.amazon.com/dp/B07YD2LT9D
- Raspberry Pi 4B running Raspberry Pi OS
  - https://www.okdo.com/p/okdo-raspberry-pi-4-4gb-model-b-starter-kit/
- Any laptop capable of SSH and running Python files

Libraries Used:

- PyZMQ for communication between threads and devices
  - https://github.com/zeromq/pyzmq
- Freenove 4WD Smart Car Code for the robot's motor and sensor drivers
  - https://github.com/Freenove/Freenove_4WD_Smart_Car_Kit_for_Raspberry_Pi
- PyQt5 & PyQtGraph for GUI framework and real-time plotting
  - https://github.com/pyqt
- Threading for concurrency between logic, motors, and sensor
  - https://docs.python.org/3/library/threading.html

Program Files:

File Name	Description
main.py	The main program that is executed on the robot. This file imports all the necessary files, controls the execution, and launches all the relevant threads.
logic.py	This file contains the actual logic for the room mapping and decision making. Within logic, it connects to the sockets to communicate with the main thread (waiting for a "START" or "STOP"), sensor thread (waiting until the robot has panned entirely), and motor thread (sending commands to move the motor and receiving acknowledgement). It also saves the data processed into points and publishes the data over a socket for any listeners. The decision on whether to go straight, turn left, or turn right uses a voting system. The traversal also contains a simple PID controller to correct any drift in the robot's movement.
sensor.py	File containing the thread which pans the ultrasonic sensor, collects readings, filters outliers, and sends average distance measured at each angle.
motors.py	File containing the thread which accepts requests and moves motors.
server.py	The second program that runs on another device. It connects (subscribes) to the robot, creates a GUI, and plots points as they are received.
sensor_test.py	Tests the sensors.py and server.py functionality.
motors_test.py	Tests the motors.py functionality by accepting user-inputted requests.