SwiftBot: Autonomous Delivery Robot

Kaavish Report
presented to the academic faculty
by

Sarfraz Shahid Hussain	sh02929@st.habib.edu.pk
Sameer A. Jaliawala	sj02732@st.habib.edu.pk
Syed Ahsan Ahmed	sa02908@st.habib.edu.pk
Syed Hamza Azeem	sa02515@st.habib.edu.pk
Sahran Riaz	sr02975@st.habib.edu.pk

In partial fulfillment of the requirements for
Bachelor of Science
Computer Science

Dhanani School of Science and Engineering

Habib University
Spring 2020

SwiftBot:Autonomous Delivery Robot

This Kaavish project was supervised by:

Dr. Taj Khan
Faculty of Computer Science
Habib University

Approved by the Faculty of Computer Science on _____.

Dedication

Dedicated to our degrees, and **a whole lot of people** we consulted for help. And also dedicated to the frontline fighters of the current Pandemic situation.

"I ultimately got into robotics because for me, it was the best way to study intelligence."
- Sebastian Thrun

Acknowledgements

We want to acknowledge Engr. Junaid Ahmed Memon for his guidance, and support towards the hardware and electrical systems of the project.

We want to acknowledge Dr. Taj M. Khan for his constant encouragement and efforts to deal with various technical and software related issues and for his guidance, progress inquiries, and appreciation throughout the course of the project.

We would like to acknowledge Mr. Ahmed Bilal for his help and support in all cap- stone related matters.

We want to acknowledge our families for their love, support, motivation, and appreciation throughout these trying times.
We want to specially thank Mr Muhammad Saad Saleem for his help and constant support throught the length of the project.

Abstract

The primary scope of our Final Year project team was to design and implement an autonomous delivery robot for large space or indoor deliveries. The core problem that our team is willing to solve was not only on time but timely delivery of small to medium packages including but not limited to food items, administrative supplies, documents for inter-departmental communication etc.

The final model is demonstrable for a unit floor plan, such as, a large indoor space or single floor area. The robot comes integrated with a mobile app to manage deliveries. The proof of concept exhibited functionalities which can be found in the final product including localization, obstacle detection, environment mapping, and rudimentary path planning behavior, on-network communication and security features, among a few.

The major challenges involved in implementing a functional project of this caliber ranged from an effective hardware implementation – making use of limited resources to ensure a smoothly functioning final product – to software and hardware integration in an efficient manner. With dedication and commitment, our team has been able to deliver a product that contributes towards the real-life application in the domain of service robotics.

Our team has been successful in building a practical solution which can be used in many traditional use-cases such as last-mile deliveries, on campus deliveries, but can also aid the front line warriors fighting with the COVID-19. The medical staff can efficiently use this project for receiving and sending test samples, reports etc.

Keywords: Service Robotics, Last-Mile Delivery, Autonomous Delivery Robots.

Contents

List of Figures

List of Tables

1. Introduction

1.1 Problem Statement

The problem of secure and swift delivery has always been a major problem within many parts of our society. There are always issues of moving medium to small-sized items within an organization that can hinder or delay time-critical tasks.
The primary motivation of the project stemmed from observing the inefficiencies of payload delivery on the Habib University campus including but not limited to, receiving copies of payment cheques and receipts from the Office of Career Services and the bank, and moving exam papers or lab manuals to and from different offices/departments. This problem also persists when dealing with bulk loads and faulty printing services. In recent times this problem has become mainstream in the context of hospitals where time-critical deliveries such as the delivery of specimens, important medical instruments are always required, etc.

1.2 Proposed Solution

With the group's interest in robotics and the desire to solve the problem on campus for its students, the group thought about potential use cases for the proposed project and how it could be scaled for other needs which led it to propose our idea.
We are building an autonomous delivery robot that will be used to deliver packages in an indoor office environment. The way it operates is that the user calls the robot using a phone app. When the robot arrives, the user places the package and marks the position where the package has to be delivered. The robot autonomously delivers the package.

1.3 Intended User

This section outlines the User persona of our completed product. The different types of users in existing in our user base and their interaction with the system are briefly described.

App User: This is the primary user of our system. S/He uses the app to call the robot, and then send the package through the robot to the intended user. The user interacts with the system through the android app, and physical robot.

Following is a list of some of the App User-System interactions.

1. Login/Signup on the system
2. Call the robot at the pickup point
3. Receive the package at the dropoff point
4. Open the box via pin from the app

System Admin: The system admin is responsible for adding map location pointers on the map. The system admin can also see many usage stats of the system such as total users, total deliveries, and is also able to export all this data in the form of excel sheets for record keeping or further analysis.

Following is a list of some of the Admin-System interactions.

1. Login to the system
2. View/Export stored data
3. Add/Update map pointers

1.4 Key Challenges

This section mentions the key challenges that we foresee in this project and possible ways to address them.

1. **Sub-System Communication**
   Ensuring that all sub-systems (both Hardware and Software) once implemented properly, would integrate with each other, and complete all dependencies.

Solution: The Web API's we have developed for the system communication purposes, need to be in coherence so that the sub-system communication is efficient and swift.

2. **Modular Integration**
   Problems arose while trying to put all divided modules together in the final phase, which took a lot of input from the group and the supervisors.
   Solution: We have been as coherent as possible in terms of the flow of data, data types and system-wide naming conventions, so that there are lesser bugs, problems and more feedback in the final phase.

3. **Facial Recognition**
   The facial recognition expects to have some sort of images in order to train the model. The other problem we may run into is the detection and recognition of the face using Raspberry pi's camera module. The module's resolution is low and is pretty slow when taking a photo, due to multiple processes being run on the hardware already.
   Solution: One way to speed things up on the pi is to use parallelism. That allows the pi camera to take photos at a more optimized frame rate. We can adjust the pi camera's position in such a way where the user's face is visible and clear; meaning that other conditions favour the recognition model, for e.g lighting conditions and the pose. To send data to the recognition model from app for training we outlined a solution where we would require a face video from the user and that video would be parsed into images on the back-end server.

4. **Obstacle Detection**
   Relying on one sensor for the obstacle detection would create limitations and put boundaries on our field of view.
   Solution: One possible solution is to use a camera and implement computer vision algorithms in order to detect objects and avoid them. Unfortunately the conditions don't favour this solution. However, using Lidar and ROS huge library we can integrate high level obstacle detection algorithms, such as navigation stack.

5. **Processing Power**
   Limitations in the computational resources of available embedded system platforms like the Raspberry Pi 3b.
   Solution: We need to manage the execution of some computationally intensive algorithms on a device with limited memory requirements and processing

capabilities. For the future we would recommend using a more powerful single board computer (maybe an STM based computer etc.)

6. **Ethical Considerations**
   It is a fact that facial recognition poses some potential privacy concerns considering that a database of facial features for users is made for verification purposes. This can be a deterrent for some users who are unsure about their privacy being breached.
   Solution: While there is no easy answer to this question as it opens up a whole new debate, the team believes that the speed, reliability, and benefits offered by using computer vision for facial recognition outweighs the negatives, at least for a project of this scope

2. Literature Review

This chapter presents the current state of the art in the domain and talks about other similar work that has been done in this area. It also establishes the novelty of our work by highlighting the differences between the existing work and our work.

For our project, we did an extensive literature review and read over 10 research papers written on work similar to our domain. This sections tries to compare our approach to the top 5 approaches in the research paper that we studied carefully. It will talk about the advantages and disadvantages of each approach highlighted in each respective research paper.

2.1 Service robots for hospitals

The motivation of the project initially stemmed from observing efficiency and convenience factors of transportation on Habib University's campus, such as, for food deliveries from the cafeteria and moving equipment between labs in a safe and rapid manner. After literature surveys, the transportation problem was also identified in other scenarios, such as, in hospital settings where time-critical deliveries, such as, lab sample delivery, medical instrument delivery etc. are required. Hospitals are vital organs of society and to make sure that the organs remain stable, regular updates are required. A huge problem in hospitals is logistics, which it mainly relies on manual delivery or robots that are either too ineffective or expensive. Some of the automation facilities that are being used here include Pneumatic tube, where items are sent through a network of tubes, driven by pneumatic forces and linked to different departments within a hospital [1]. Another solution, that is being considered, is mobile robot, where different sensors are being used to deliver cargo in secured and sanitized box.

- Advantages:

  - Helps us identify the urgent requirement in health care facilities and the need of these robots.
  - Provides us alternatives to our solution as well as an analysis of the solutions used.
  - Gives us an idea of how mobile robot's efficiency can be improved. One of the solution mentioned here is Helpmate that helped us give a boost in planning and figuring out a solution at hand

- Disadvantages:
  - Technically weak. Gives us analysis but doesn't really give us any technical details of how we can come up with an efficient robot.
  - The paper is mostly concerned with the need of a better solution for the hospital's logistic problem. It even mentions some limitations in mobile robots that are being used, which actually brought our motivation down to build a robot.

2.2 Sensors and Sensor Fusion in Autonomous Vehicles

Autonomous vehicles are currently the one of the most talked about topics in automotive industries. The efficiencies and features of the modern sensors, makes it possible for some really innovative vehicles that can drive on its own.
The solution that is defined here includes the study of finding out whether the fusion of three types of sensors for autonomous driving, namely the Camera, Radar, and LIDAR, can help the automotive industry in designing a self-driving vehicle. The approach talks about dividing the space around the vehicle in the form of an n x n grid and to detect whether something lies in a block of the grid or not. The radar, that is being mentioned here, is used for long ranged obstacle detection and the robot relies on the camera for object classification, and LIDAR for implementing SLAM (Simultaneous Localization and Mapping) [2].

- Advantages:
  - Uses 3 sensor fusion for achieving better results.
  - Uses Radar and Camera for obstacle detection, which it claims that it gives accurate and precise results.

  - Uses Radar which increases the robot navigation range. Radar can help in identifying objects in distance and are readily available.
  - Definitely helps us in planning and identifying the sensors we can use for our solution. It even gives us some information about control mechanisms, which can be made more efficient, by the use of sensor fusion.

- Disadvantages:
  - More computationally heavy as the obstacle detection and avoidance is carried by both the camera and LIDAR.
  - Costly solution.
  - There is no mention of Robots over here, which makes this paper kind of insignificant to the solution that we are proposing.

2.3 A Layered Architecture for Office Delivery Robots

The third solution we tried to look at is very similar to what we are trying to achieve with our own robot. The solution aims in identifying that indoor organizational logistics are proving to become important day by day. Some of these organizations include offices, where robots have to perform a variety of tasks in order to deliver items securely and efficiently. The paper identifies a basic flow of what the robots need to do, in order to deliver parcels, and provides appropriate architecture for it [3]. The tasks include: Path Planning, Obstacle Avoidance, Navigation and task scheduling. It even provides a detailed overview of the user interfaces they considered for their own solution. One of them was the Zephyr interface which is an asynchronous rapid communication mechanism with verification features. The other interface they mentioned is the World Wide Web, in order to accept tasks from any remote location.

- Advantages:
  - The solution that is being presented here is very much similar to our own plans. It's based on the idea that the robot will be requested to go to location X, take the parcel and then go to location Y to deliver it.
  - The architecture that is being proposed here generally helped us in identifying what our robot's functionalities would be. It also helped us identify the tasks the robot would be needed to carry out.

- Disadvantages:
  - The solution is very old and consists of some of old technologies that are being used for designing a user interface. We already knew that in order for our solution to succeed, it needs some kind of an interface that is fully remote and mobile.
  - It doesn't mention any technical details about exactly what kind of sensors would we need and how are we going to make sure that it does all the tasks, in the modern world.
  - The security of the document, that the robot delivers to a user, is not discussed in detail. It doesn't mention how it's going to keep the parcel protected and allow accessibility for the user.

2.4 Analysis of ROS-based Visual and Lidar Odometry for a Teleoperated Crawler-type Robot in Indoor Environment

The research paper that we looked at next was very technical in nature and introduced to a very novel type of robot, i.e a teleoperated crawler-type system. Now the SLAM methodology that they used in order to solve their problem was visual, meaning that it was dependent on the LIDAR sensor as well as the camera sensor. The SLAM methodologies that were analysed are ORB-SLAM, V-SLAM and LSD-SLAM. The robot needs to know where it is currently located and the orientation it is placed in. This is possible by the use of Odometry; the paper suggests the use of visual odometry to solve its tasks. It mentions the use of GPS and IMUs as well [4]. The whole system is based on the the Robot Operating System (ROS), the operating system that our robot will use as well.

- Advantages
  - The solution is based on indoor environments and mentions the use of those sensors and methodologies that we need to consider in order to make our robot.
  - It gives a thorough analysis of the visual and LIDAR odometry and also explains us how mapping would be carried out through a robot.
  - The system is being run on ROS and gives us a boost in motivation that a robot as complicated as the one being introduced here, is actually possible.

It even mentions that the robot needs to have some sort of sensor fusion in order to complete its tasks.

- Disadvantages
  - The robot is just too complicated and a lot of is difficult to understand, without any previous knowledge of how everything is works and what each of their significance is.
  - The paper is also extremely analytical in nature and the robot mentioned doesn't depend on a limited embedded system like ours.
  - There is a lot of information about the virtual odometry and virtual slam techniques that is currently being used. However, it is important to know that the system we are designing is mostly dependent on the LIDAR sensor and not the camera.

2.5 An Evaluation of 2D SLAM Techniques Available in Robot Operating System

The last paper we looked at was mainly about the analysis of different 2d SLAM techniques that are available on ROS. It compares and reports results of the mapping algorithms and suggests the use of some important ones. The five SLAM algorithm it considers are: HectorSLAM, Gmapping, KartoSLAM, CoreSLAM and LagoSLAM [5].

- Advantages
  - Although it won't look significant to the reader, the mapping techniques they mentioned here are actually the ones that we are going to experiment with. The information that is available here will help us know exactly which mapping algorithm to use later on.
  - The solutions are based on ROS and helps us in identifying the pros and cons of each algorithm
- Disadvantages
  - The paper doesn't mention of what robot is being used or other technical details about the robot. It just mainly talks about the SLAM algorithms.

  - It doesn't mention what would happen if we incorporated some kind of fusion with camera. It is mostly concerned with other sensors, such as LIDAR and wheel encoders.

2.6 Good to Know

Current implementations of some delivery robot technologies have some conflicting views regarding their deployments and usage, which brings into play an additional inspection related to regulatory environments that goes beyond the aims and scope of this project. However, keeping some of these criteria in mind, restrictions, such as, size, weight, and speed limits are imposed on most delivery robots. The characteristics of some of the notable delivery robots are summarized in Table 2.1

Company	**Weight (lb)**	**Speed (mph)**	**Capacity (lb)**	**Capacity (chambers)**	**Range (mi)**
Starship Technologies	40	4	40	1	4
Domino's DRU	Unknown	12	21	4	12
Dispatch's Carry	Unknown but requires 2 people to lift device	4	100	4	up to 48 mi with 12 hr battery
Thyssenkrupp's TeleRetail	60	35	77	1	10
Marble	80	4	Unknown	1	Unknown

Table 2.1: Characteristics of Some Notable Autonomous Delivery Robots

In addition to the 5 research papers that we identified above, there were others that helped us out while planning and research. Some of these are divided according to type of contribution they make:

- Automation in Health Sector
  - Mobile Robot Simulation of Clinical Laboratory Deliveries [6]

  - HelpMate: A Robotic Materials Transport System [7]

- Economical Impacts of Automation
  - Quarterly E-Commerce Report 1st Quarter 2018 [8]

- Autonomous Vehicles
  - Car Detection for Autonomous Vehicle: LIDAR and Vision Fusion Approach Through Deep Learning Framework [9]
  - Comprehensive and Practical Vision System for Self-Driving Vehicle Lane-Level Localization [10]

- Sensor Fusion
  - Fusion of 3d-lidar and camera data for scene parsing [11]

3. Software Requirement Specification (SRS)

This chapter provides detailed specifications of the system under development.

3.1 Functional Requirements

This section describes each function/feature provided by our system. These functions are logically grouped into modules based on their purpose/users/mode of operations etc (as per our system). A functional hierarchy may look like:

- Robot:
  - The robot is able to move across the mapped environment and securely and safely makes it to the destination point.
  - The robot detects static as well as dynamic obstacles, and take proper maneuvers in order to avoid crashing into the obstacle.
  - The robot has a secure locking system in place to place any unauthorized access to the products inside the robot.
  - The robot plans a path using the `A* algorithm` and makes adjustments to the path according to the situation it is presented.
  - The robot's trajectory is almost smooth and as simple as possible, to maintain a timely delivery.
  - The robot will not harm or damage any structure or living thing, and it would be environmentally friendly.
  - The robot's internal circuitry and components would be hidden in a compartments to make sure that the look of the robot seems as elegant as possible.

  - The robot's size would be such that the user is able to reach the content box easily and is able to show their face to the camera, so that the box gets unlocked.
  - The robot will use the Lidar sensor readings and Google Cartographer's mapping algorithm to create a map of the environment.

- Web Server:
  - The Web server is responsible for storing booking details, user details and any other information that is necessary for the system to work.
  - The web server can be accessed by the control panel of the admin and only the admin has the rights to alter database table definitions.
  - The web server will be able to communicate with the robot and share booking details, delivery statuses and locking controls.
  - The server will receive the facial recognition video from the app and it will parse the video into images.
  - The web server will use the images to train a model and then generate a file that will be used to recognize faces.
  - The server is also creating frequent backup of all necessary data.

- Android App:
  - The user will fill out its details on the app and the app will send the information to the web server.
  - The user uploads his face video to the server that gets sent to the server for training.
  - The user can book the robot for delivery and can find out the status and estimated time it will reach them.
  - Both the sender and receiver will be informed about the status of the delivery.
  - If there are any delivery requests while the robot is being utilized, the app will inform the user that it has recorded its details and will fulfill the request by adding it to a queue.
  - The app will be able to offer an alternative way of opening the lock, i.e a secure key, in case the facial recognition system does not work.

- Web App:

- The Admin Panel allows admin to view all bookings and users details.
- The Admin Panel allows admin to control the robot that will be used for mapping the indoor space when the robot is deployed somewhere.
- The Admin Panel allows admin to add pointer on the map that will then show on the app. These pointers can then be used for adding source and destination to a bookings.
- The Admin Panel allows admin to export all data in excel format 5. The Admin Panel will be protected and hence the admin will have to login using the right credentials.

3.2 Non-functional Requirements

1. **APP**

   (a) The mobile app should be capable enough to handle the load of high number of users without noticeable effects on its performance, navigation or glitches.

   (b) The app should be responsive and quick.

   (c) The app's design should be made while considering UI Design methodologies and the interface should be appealing to the user.

   (d) The mobile app's backend server should provide data security for user data, via the use of some encryption technique

   (e) The response time of API should not dissatisfy a user.

   (f) The app should determine a way to handle user queues if the robot is booked and processing for another user.

   (g) The app should be ideally available to download from the Google Play Store and the Apple App Store

   (h) The app should capture minimum possible data during signup, to boost the onboarding process.

   (i) The app's codebase should be scale able so that more features can be added in the app.

   (j) The app should have a secure login while making sure that the user does not have to provide login details every time.

2. **SERVER**

(a) The server should have capabilities of performing data processing and executing large amounts of code.

(b) The server should send and receive all requests over HTTPS for security.

(c) The server showed employ methods to prevent SQL injections and other known hacks.

(d) The server should delete uploaded user face videos after training for privacy purpose.

(e) The server should have a bandwidth of `1 GB` to handle large amounts of requests.

(f) The server should be able to communicate quickly and securely with other modules.

(g) The server should have a large storage so that it can store huge amounts of data.

3. **ADMIN PANEL**

(a) The admin panel should be responsive so that it can be accessed on mobile phone.

(b) The admins credential should be encrypted.

(c) The admin panel should have interactive features that are responsive and intuitive.

(d) The admin panel can be accessible from many devices.

4. **ROBOT**

(a) The robot should be easily deployable without interfering with any infrastructure of the organization's space.

(b) The robot should make sure that the payload is secure.

(c) The robot should be time efficient in performing its task.

(d) The robot should be human friendly.

(e) The robot should have an adaptive speed so that if any object comes within its boundary, it can take appropriate measures.

(f) The facial recognition on the robot should be robust.

(g) The structure of the robot should be strong.

3.3 External Interfaces

This section includes our UI screens and briefly explains them.

3.3.1 App User Interfaces

Signup and Login Page

Figure 3.1, on the left, shows the signup page. The signup page is the first step towards creating an account. New users have to specify their email, name, and password in order to create their account.

Figure 3.2, on the right, shows the login page. Existing users can use their email, and password in order to login to the app. Login is required only once, so the next time the user opens the app, he or she does not have to login again.

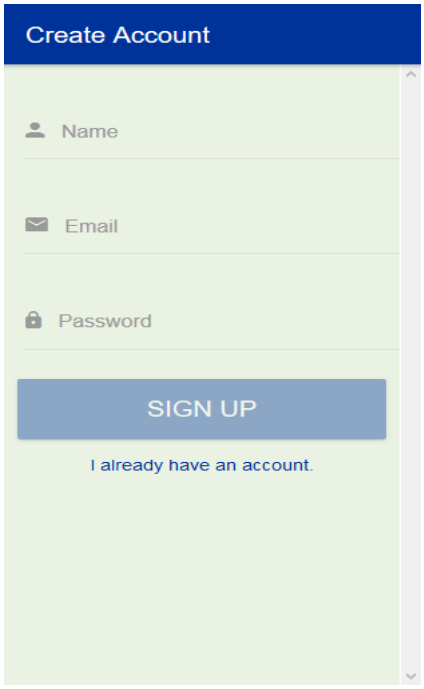


Figure 3.1: Signup Page

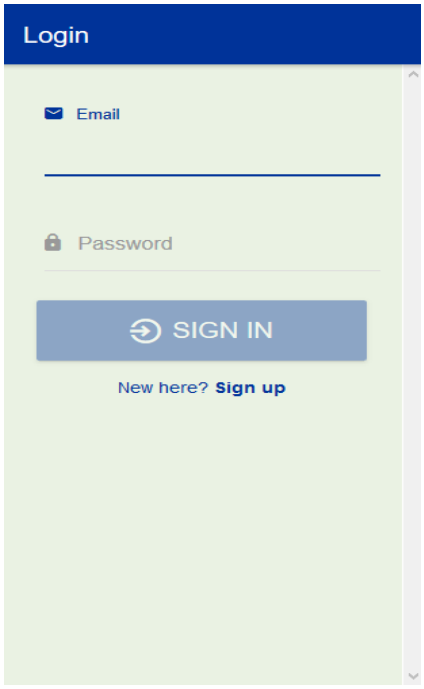


Figure 3.2: Login Page

Facial Registration and Default Location

Figure 3.3, on the left, is the face registering page. Here the user has to upload a 10 seconds video of his or her face, which will later be used by the Facial Recognition module to train itself.

Figure 3.4, on the right, shows the add default location page. The user has to add his or her default physical location, which indicates where the user will be found, and will be used when receiving packages.

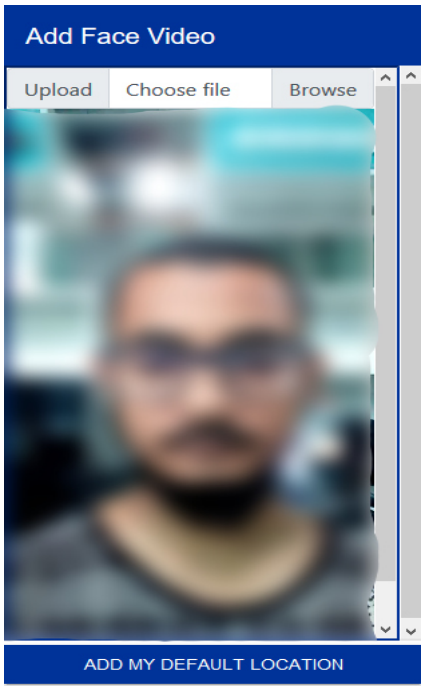


Figure 3.3: Register Face

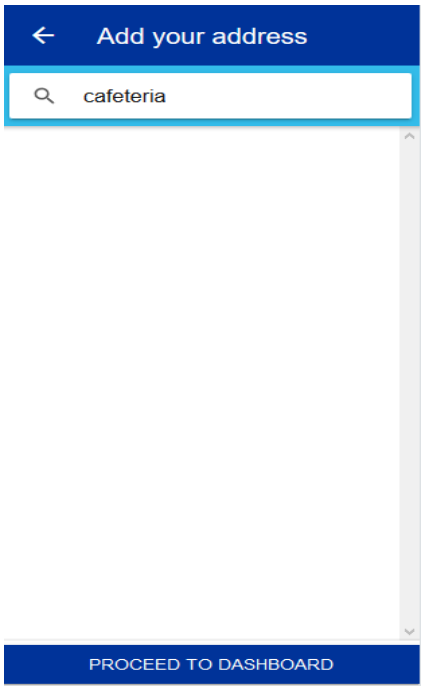


Figure 3.4: Add Default Location

User Dashboard and Delivery History

Figure 3.5, on the left, shows the user dashboard. This is the first screen the signed in users will see once they open their app. It shows the live ongoing delivering, as well as, contains the buttons to initiate a new package delivery, or view previous deliveries.

Figure 3.6, on the right, shows the delivery history. It shows the history of all the past sent, and received packages. User can click a booking to view its detail.

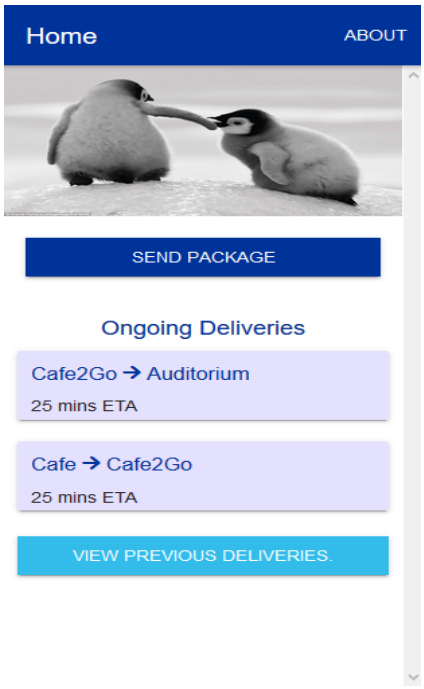


Figure 3.5: Dashboard

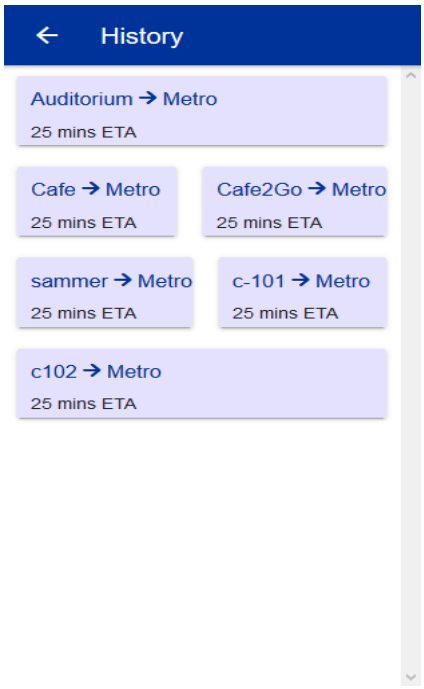


Figure 3.6: History

Booking Details and Pickup Point

Figure 3.7, on the left, shows the booking details page. It shows the details of the selected booking. The booking details include, sender, receiver, data, time sent, and time received.

Figure 3.8, on the right, shows the pickup point page. It is the first step when sending a package. The sender has to add their address so that the robot can come to his or her location to pick up the package.

Figure 3.7: Booking Details

Figure 3.8: Pickup point

Add Receiver and Open Lock Page

Figure 3.9, shows the add receiver page. This is the second step when sending a package. The user has to tag their receiver, so that the robot can go to the receiver's default location to deliver the package, and inform the receiver that his or her package has arrived. Figure 3.10, shows the open lock page. When the box arrives at the receiver's end, the user has the option to open the box with his/her password.

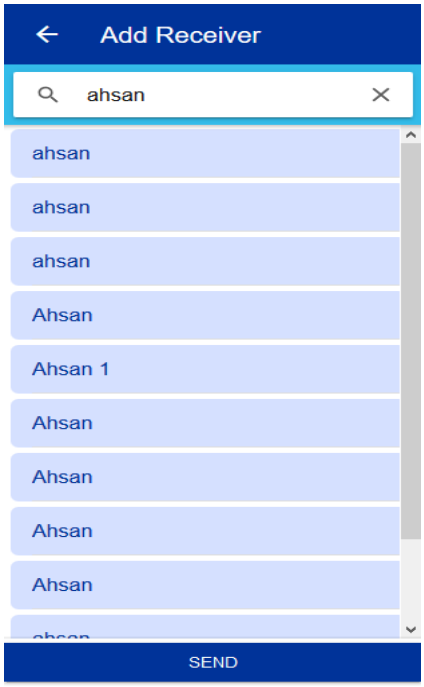


Figure 3.9: Add Receiver

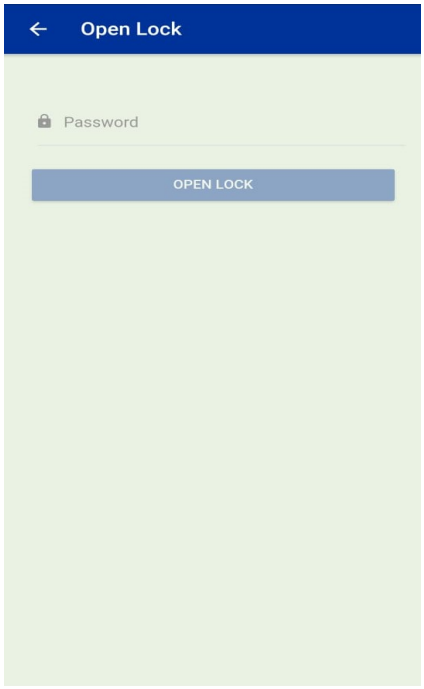


Figure 3.10: Open Lock

3.3.2 Web Application

Login Page

Figure 3.11 shows the admin login page. This is the page that the system admin can use to access the main system.

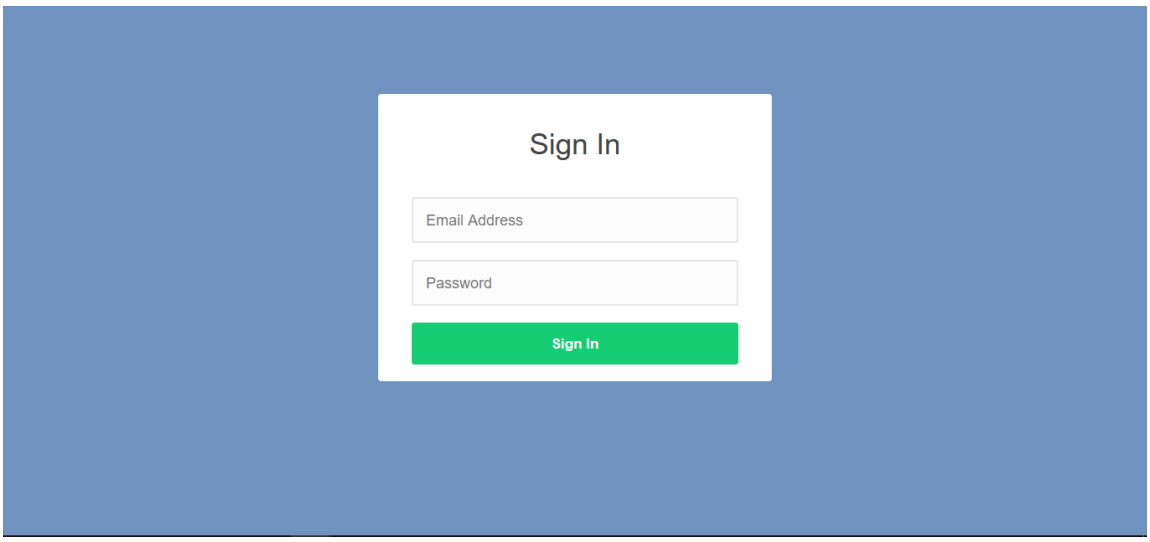


Figure 3.11: Admin Login

Dashboard Page

Figure 3.12 shows the admin dashboard page. This page allows the admin to interact with the data base, and see the users of the apps, along with delivery reports, and add pointers to the generated map.

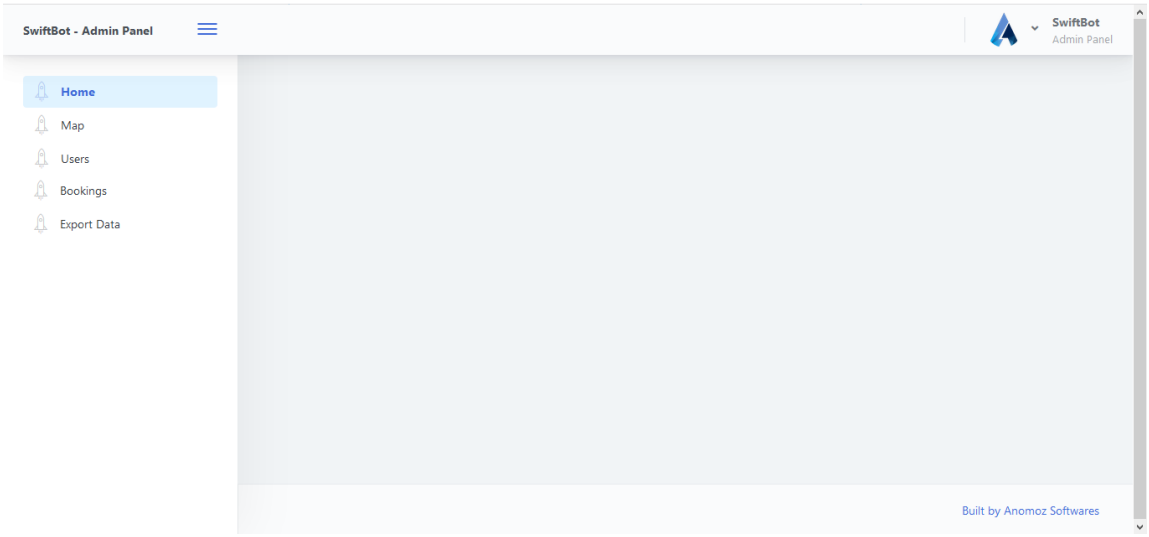


Figure 3.12: Admin Dashboard

Users Table

Figure 3.13 shows the users table. It contains the list of all the registered app users, along with their name, email, and other information.

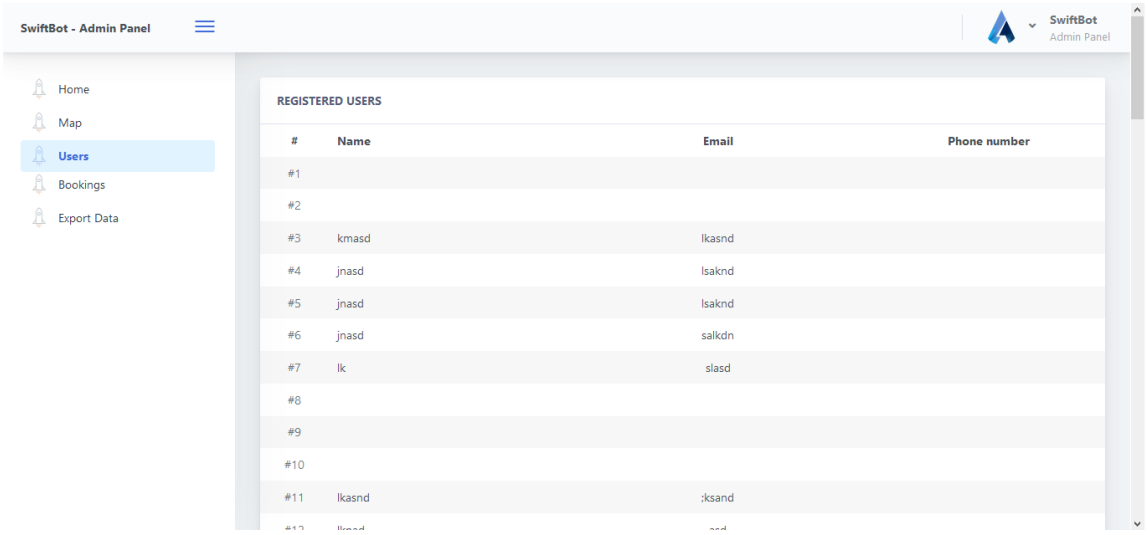


Figure 3.13: Users Table

Floor Map Editor

Figure 3.14 is the floor map editor, one of the most important parts of the whole system. The admin can add map points on the generated LIDAR map, so that the users can use these map pointers to send, and receive packages across the indoor space. It allows the admin to add map pointers, and update the existing pointers. It is important to know that, as the map image is of different type than the one on ROS, hard coded ROS map values were supplied into these pointers. It is not possible to transform coordinates on this map to the ones on ROS, as there seems to be no mathematical relationship between them. Therefore, every time the coordinates were put up on these maps, it was required to first find out what they are corresponding to on ROS.

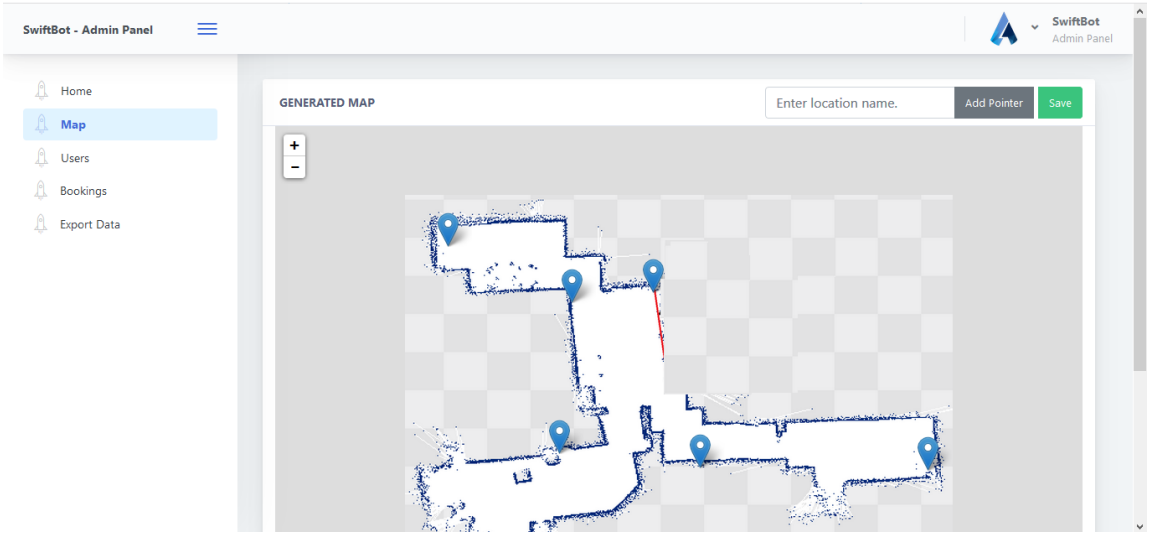


Figure 3.14: Floor Map Editor

Export Data

Figure 3.15 shows the export data page. It allows the admin to export stored data from the database to the excel files, for further data analysis.

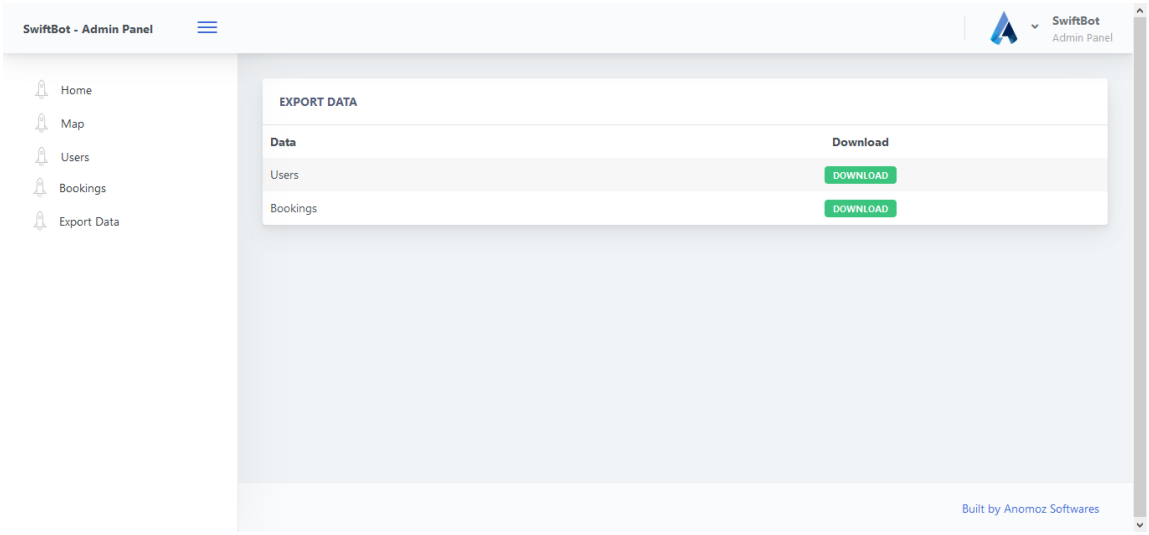


Figure 3.15: Export data

Figure 3.16 shows the export to Excel page. This is the page from where the admin can download the generated excel file.

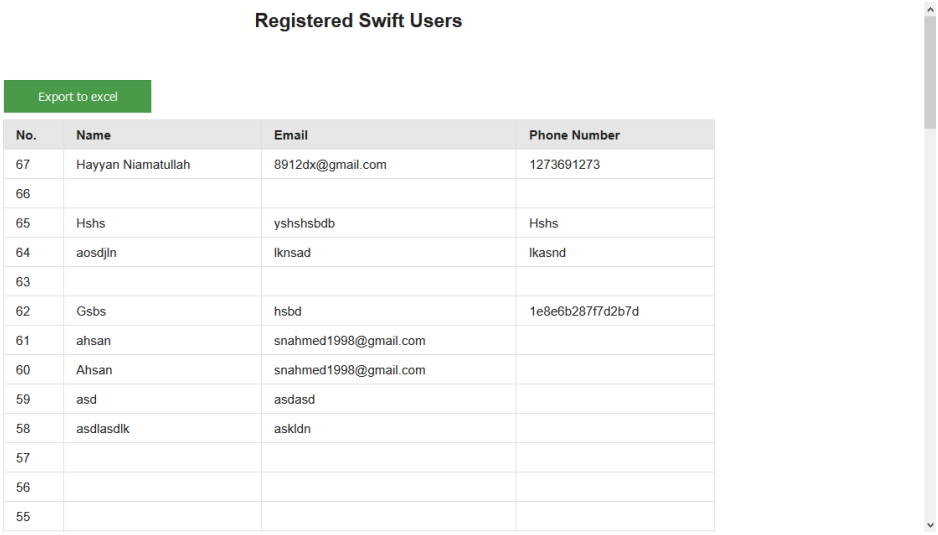


Figure 3.16: Export to Excel

3.3.3 Application Program Interface (API)

For our system, a huge number of APIs will be used for carrying out the operations smoothly. These APIs will be used to connect the bot to the central server, app to the central server, and the central server to the Py server responsible for carrying out the facial recognition.

The APIs that will be used by the system have been listed and described below.

1. `distance: get_distance_between_points(pointA, pointB):`

   This API will be used getting the distance between two points. This info can then be used for solving the scheduling problem, ie which delivery should be done next.

2. `time: get_remaining_time_of_journey():`

   This API will be used getting the time left for completing that ongoind delivery, ie the time taken to reach the destination. This info will then be showed to the user on the App.

3. `void: mark_current_journey_as_finished():`

   This API will be used for marking the current journey on the central server as completed.

4. `deliveryDetails: get_next_delivery_details():`

   This API will be used for fetching the next delivery details. This API will be called one the current delivery is completed and now the robot is looking for the next delivery to fulfill. This information will include, sender information, receiver information, sender location Id, and receiver location Id.

5. `bool: validate_user_faceId(image):`

   This API will be used for validating receiver's faceId against the trained model. One the parcel has reached its place, it will send the receiver's face image to the Py Server to be validated and then the container will be opened.

6. `Locations List: get_map_locations():`

   This API will be used to fetch all map pointer locations to be shown on the app which were initially added by the Admin. These map points will be used by user to select their location from where they will send or receive the package.

7. `bool: signup_user(userInfo):`

   This API will be used to signup user on the server through the app. These details include name, email, password, and default location.

8. `bool: login_user(email, password):`

   This API will be used to login user to the app in case he already has an account and he is logging in a new device. The API will validate user credentials on the server and send a response if the login was successful or not.

9. `users list: get_all_users():`

   This API will be used to fetch all users from the central server to be shown on the app. This users list will then be used at the time of booking to tag the receiver to the order.

10. `bool: insert_booking_to_server(bookingDetails):`

    This API will be used to insert booking to the server. After the user has specified order details, such as current location, and the receiver, the booking will then be pushed to the server though the specified API.

11. `bookingDetails: get_live_bookings(userId):`

    This API will be used to get live user booking. The bookings include the ones he is sending and the ones he is receiving.

12. `bookingDetails list: get_all_user_bookings(userId):`

    This API will be used to show the history of user bookings on the app. The booking details will include, sender name, receiver name, sender location, receiver location, date, and time.

13. `bool: upload_face_video():`

    This API will be used to upload a user's face video to the Py server. This video will then be used by the Py server to train the model and validate receiver's face at the time of package delivery. If upload video successfull then it will return true.

14. `bool: open_lock():`

    This API will be used inside the Main Loop of Ros. This API uses ROS Serial to send command to open and close the lock. The Arduino unit will also report the status back when the lock is actually closed or open.

15. `bool: open_lock_from_app():`

    This API will be called from the app on to the server which will then call the original $open_lock()$ api on the robot. Its called when a user's password is validated successfully on the app.

16. `bool: Call_MoveBase():`

    This API will be called from the Main Loop to the Move Base on our laptop, to turn the navigator on.

We have built a sample API on our remote central server for fetching the map pointers. The API can be accessed at:

`https://api.anomoz.com/api/swift/post/read_all_locations.php`

3.3.4 Hardware/Communication Interfaces

This section describes our project's specific hardware/network interfaces.

- The Hardware component consists of processing components and sensory components.
  - Computing units:
    - Raspberry Pi (Model 3B)
    - Arduino Model UNO
  - Sensors:
    - Camera
    - LiDAR Sensor
    - Ultrasonic Sensors
    - Encoders
- The given hardware components communicate to the Central server (hosted on the web) via Wi-Fi, and then route the data to two different gateways.
  - Web-Panel (Admin Facing)
  - Android Application (User Facing)

The Arduino and Raspberry Pi are used to setup a master-slave communication hierarchy, such that the Arduino is working and collecting the data from the encoders, which give information about the current position of the robot, that is pushed to the Raspberry Pi.

The Raspberry Pi is dealing with the inputs from Arduino as well as the sensor array including (LiDAR, Camera etc.) all of this data is sent to the Web API (discussed in `3.3.3`), and processed and received via the APIs as well, which gives context to the decision making and path planning of the hardware.

3.4 Use Cases

This section presents detailed use cases of our system.
the use cases of the robot are Last Mile Delivery which can be easily scaled and deployed in areas where efficient and secure delivery is of essence.

- University Spaces
  - Cafeterias: Deliveries of food from multiple cafeterias to campus spaces
  - Inter Departmental Communication: the robot is made secure so documents like exam copies and documents can also be sent through this medium
- Hospitals/Medical Laboratories
  - Specimen Collection
  - Delivery of Supplies/Medicine
  - Reports/Cheque Deliveries
- Gated Communities
  - Neighbourhood Deliveries: Gated communities are on a rise locally, our robot can also be deployed in these spaces to promote neighbourhood deliveries, instead of cars or by walk.
- Office Spaces
  - Inter Departmental Communication: the robot is made secure so documents like sensitive documents and parcels can also be sent through this robot.
- Warehouses
  - The robot is made strong which makes it a strong candidate to deliver payload of small to medium load (ideally 5-10 KGs)

3.5 Datasets

This section describes the specific dataset(s) used to build our system.
The Project is based on a lot of data inputs, like real-time data from sensors and encoders.
However, for the `Facial Recognition` module we will be taking an input Video (ideally 7-10 seconds) and then train the model on this video (stripped into frames). That will require around a 100 pictures for each face to be trained.

3.6 System Diagram

This diagram gives a high-level view of the different components of our system and the interactions between them. Each component and the particular tools/technologies/libraries used to build it are described.

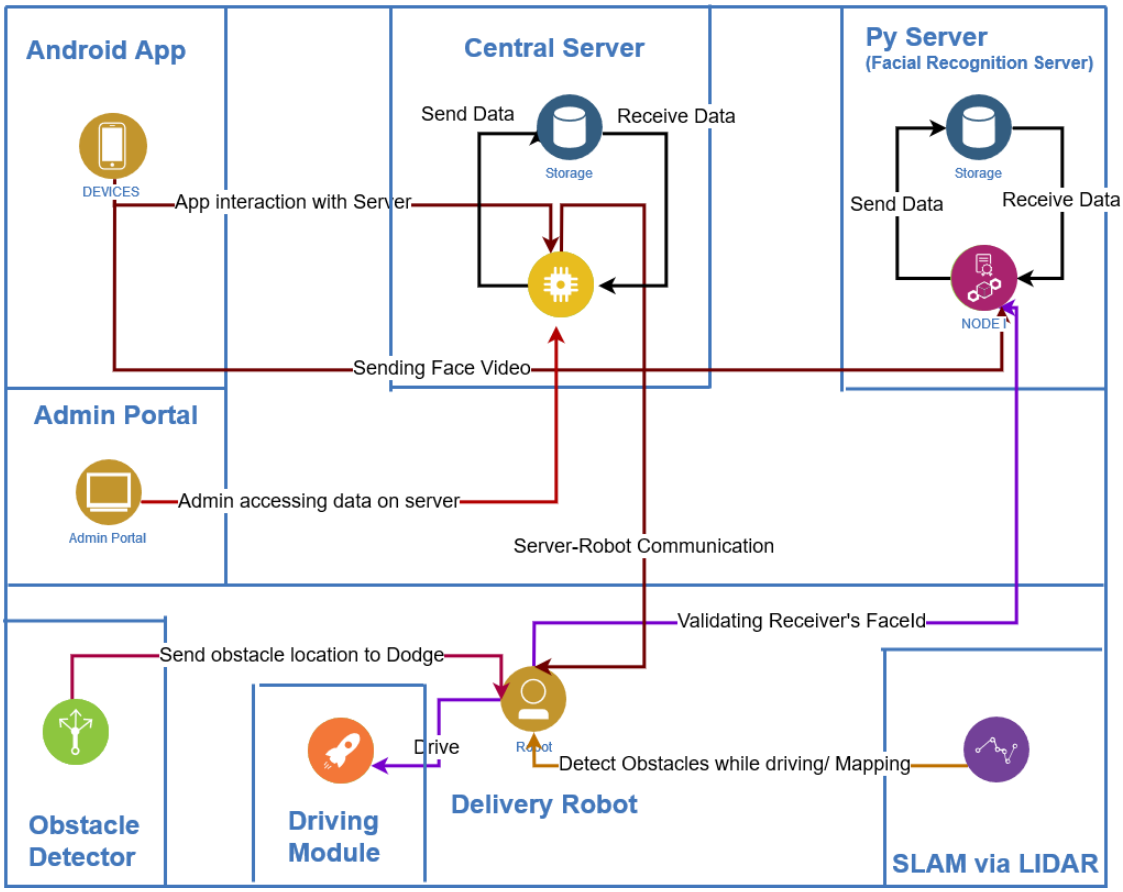


Figure 3.17: System Diagram

The system diagram shows an abstract view of the main modules of the project along with their interaction. There are 7 main modules in this project, which are as follows.

1. Android App:

   This is the module that interacts directly with the user. The user creates account through the app. The user can then use the app to send packages to

other users through the app, by first calling the robot to his location, and they tagging the receiver. During the process, the user can view live location of the robot. The user also has liberty to view previous deliveries. This module directly communicates with two other modules, ie the Central Server and the Py Server. Communication with the central server is to place bookings, and communication with the Py server is to send the user's face video to the Py server for training the model so that the users face can be used to open the container at the time of receiving package.

2. Central Server:

   Central Server is responsible to handling the app/Admin Panel/Robot communication. Data shown on the android app is first sent to the central server from the robot, which is then sent to the android app. The data communication includes, map, map points, user details, current booking details, booking queue, etc.

3. Admin Portal:

   Admin portal is used by the admin to view details stored in the database. The details include users, current bookings, past bookings, robot status. The admin also has the liberty to add new map points on the map which are then shown on the app.

4. Python Server

   Python server is responsible for handling the facial recognition module via an API interface. It gets the user's face video from the app, divide it into pictures, and then use those pictures to train the model. In addition the module also connects with the robot to validate the receiver's face id at the time of receiving the package.

5. Obstacle Detector

   Obstacle detector which is a component in the robot is responsible for detecting obstacles. It runs in the raspberry pie and can detect objects such as people, chairs in order to navigate easily.

6. Driving Module

   Driving module which uses the Master Slave architecture, runs inside the raspberry pie, sends information to the Arduino, which then sends information to the motor driver to drive the robot.

7. SLAM

   The Lidar module is used for building the initial map of the indoor space, and then exporting that map to the central server to be shown on the admin panel. The SLAM module is also used for obstacle avoidance at the time of driving.

The overall system and process block diagram of the project is shown in Figure 3.18.

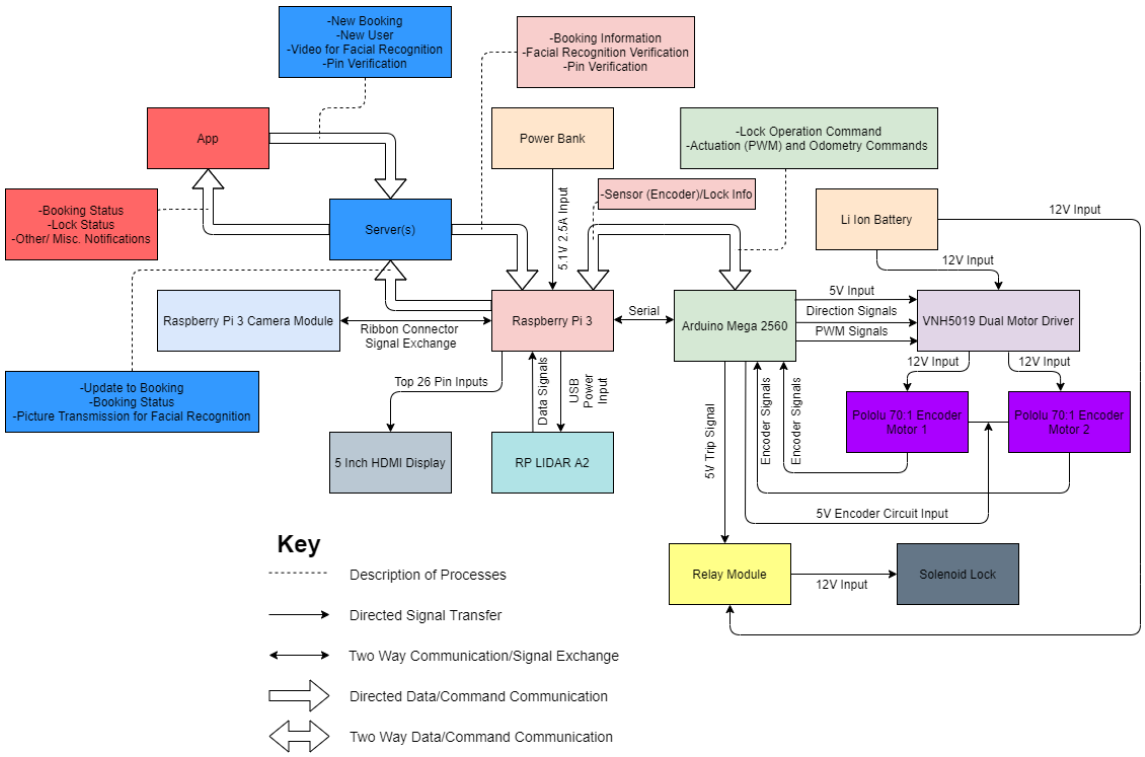


Figure 3.18: Overall System Block Diagram

4. Software Design Specification (SDS)

We have discussed the crux of our project in the previous part of our report (SRS document). In this section the important documents have been attached in relation to the design and implementation of the project.

4.1 Software Design

This section gives a brief overview of how the server is connected with the robot over the internet, communicating with the robot and backing database to the server, so that the robot is provided with the data required in order to function. Attributes and methods of each class and relationship among classes are presented, however are tentative(more to be added as we work on completing the deliverables).

1. Face Recognition: This module will be used to upload photo or video to the server, so that the model is trained and the user's face is registered into the system. When the robot arrives to the user, the facial recognition algorithm at the back end will validate the user's face and send the appropriate decision to the robot.

2. Robot: This module is entirely related to the robot that we are dealing with. It will be able to send and receive information about user's distance from the robot, the time it needs to reach the user, its current location and the details about the next delivery. The robot will also validate if it has reached the destination and communicate it with the server.

3. Central Server: This module is linked with all the other modules in the system. It will send and receive information about user, the bookings and the locations it needs to go to.

4. User: This module will be able to control the login and sign up details of the user as well as other information about the user.

5. Booking: This module will send delivery booking details to the server and will control the current bookings and the details that it will receive from the app.

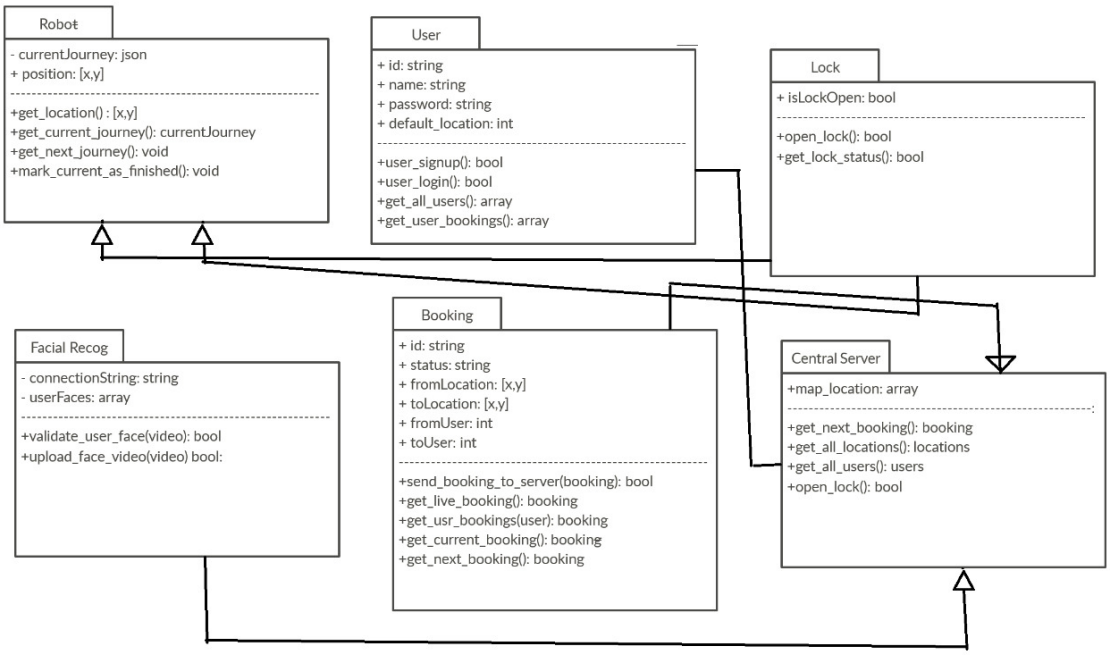


Figure 4.1: Figure: UML Diagram of Swiftbot

We have also designed a Main Loop to link all the modules together(Explained in section 4.3). The program flow diagram is shown below

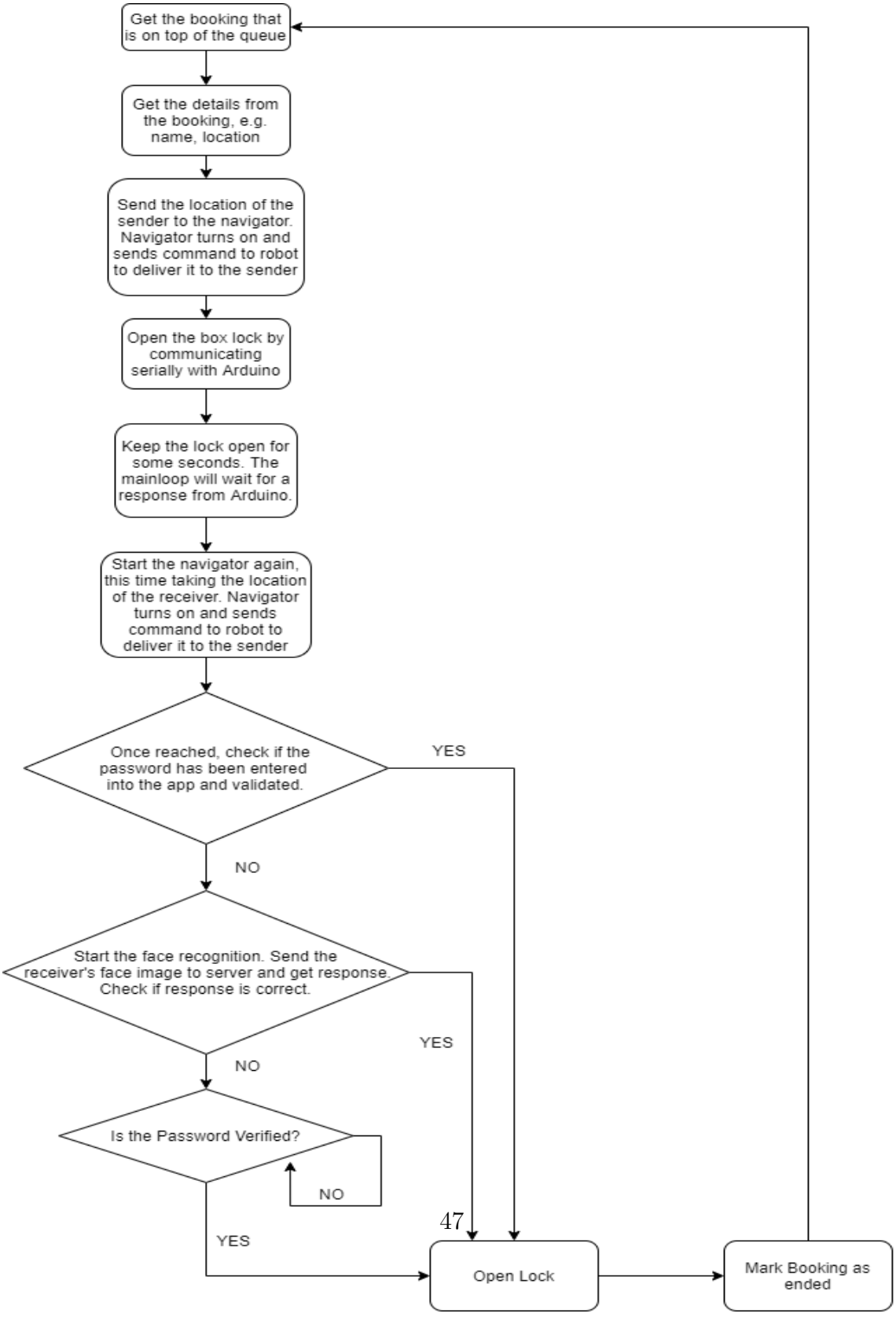


Figure 4.2: Program Flow Diagram

4.2 Data Design

This section presents the structure of our database that caters to persistent data storage in our project. The structure is shown as a normalized data model for relational databases. It clearly shows entities, attributes, relationships with their cardinalities, and primary and foreign keys. We have used DB designer, to give a clear overview of the data which is to be maintained.

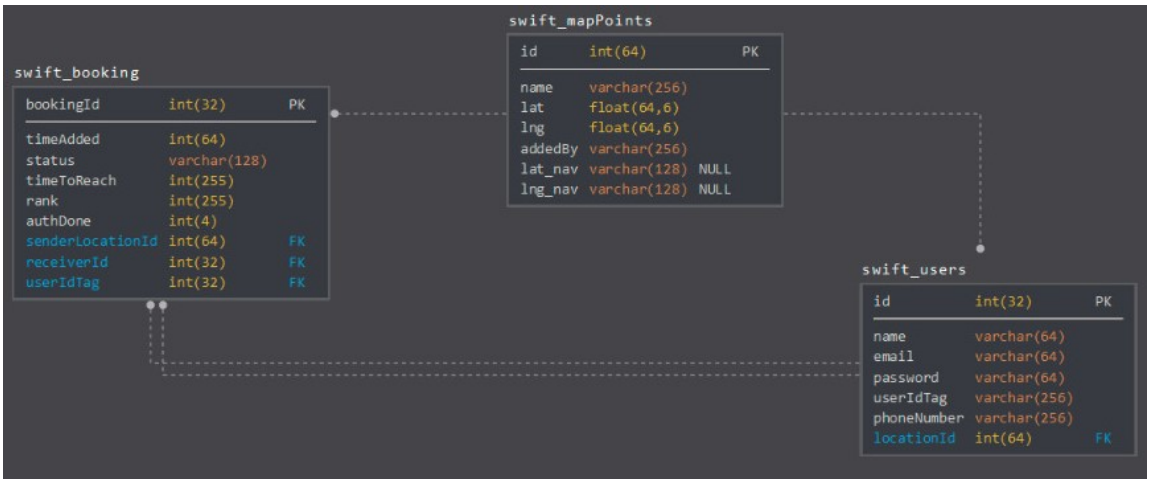


Figure 4.3: Entity relation diagram of the Backing Database

4.3 Technical Details

Our project comprises of different modules integrated with each other and each one of them has several significance. In this section we will discuss some of these details and try to fill in the holes of what our project is trying to achieve.

4.3.1 ROS

Robot Operating System (ROS)

The Robot Operating System (ROS) is an open source framework for use in the robotics industry. The framework works on the Linux architecture, and offers fea-

tures, such as, low-level (hardware) device control, sensor fusion, robot simulation through Gazebo etc.

Why ROS?

ROS has a large community support. The regularly updated, and largely available libraries in ROS offer convenience to new users. Since, the project relies on the fundamentals of localization and mapping, ROS is used. Furthermore, the project also relies on the routing of data between the server, and the low-level hardware (Arduino). Therefore, to deal with such issues, an embedded system, such as, a Raspberry Pi is used.

The team decided to load the Lubuntu OS on the Raspberry Pi since, it allows multi-threading of processes to be done, allowing it to employ less memory. Further, the team also loaded the Lubuntu OS packaged with the ROS Framework, and setup a Catkin work space. After the creation of the work space, the team downloaded the libraries that were essential in developing the software base for the project.

The fundamental reasons to use ROS are as follows:

1. ROS is a famous open source framework that includes packages for a wide range of applications, and contains pre-built packages for a majority of the actuators, sensors, and other hardware, as well as, other packages to build drivers for the components which do not have drivers.

2. Complex projects require suitable simulations before execution. ROS comes pre bundled with Rviz and Gazebo modules through which a completely modifiable virtual environment can be setup for necessary testing of the robot's functions. Moreover, these simulations can also benefit in run time. Rviz can also be used to display a live map for SLAM.

3. ROS offers an extensive language compatibility. This trait can turn out to be quite useful when large teams having members with different programming backgrounds come together to build something. A system based on ROS can include packages and libraries from different programming languages, which as a whole work together. It is also significant when a cross platform system is built, or a vast range of packages is required to be incorporated which might be written in a multiple languages.

ROS Concepts

There are certain characteristics and paradigms of ROS which differentiate it as a software suite. These include concepts, such as, nodes, topics, publishers, ad subscribers, messages, and launch files, which are highlighted henceforth:

1. Message - Messages are data structures with particular fields that specify the type of data for example, String, Boolean, Integer etc. that is being transferred between nodes.

2. Node - A ROS node can be described as a process performing a computation. It is simply an executable program that runs as a part of the application created by the developer. Nodes are typically combined into a graph, and these nodes communicate with each other through protocols including topics, services, and actions. The purpose of handling computations through the encapsulations, called nodes, includes reducing the complexity of code by separating an application into more manageable chunks, which also helps in scaling it up and make it more modular. Nodes also improve the application's fault tolerance since, they communicate via ROS without being directly linked together so that if one node crashes, the others are not affected.

3. Topic - Information exchange between nodes in ROS occurs via buses which are known as topics. They determine what type of data can be transmitted between nodes. Topics establish a decoupling between the production and use of data. Nodes are unaware of which node they are communicating with. Rather, they publish to a particular topic when they want to send data, and they receive data by subscribing to the relevant topic.

4. Publisher and subscriber - The publisher/subscriber (pub/sub) mechanism is the paradigm by which ROS operates. This mechanism facilitates the exchange of data between nodes by subscribing to topics to receive data, and publishing to topics to send data. The pub/sub communication model also facilitates many to many communication between multiple publishers and subscribers which are transmitting and receiving nodes, connected to each other by topics.

5. Launch file - These files provide convenient means to launch multiple nodes, and initialise parameters in an efficient and manageable manner. They usually contain code where the parameters for the application, such as, PID settings for the robot, are stored, as well as, code to initialise ROS packages.

ROS Setup

The project is built on ROS Kinetic. While there are later versions of ROS currently available to the public, ROS Kinetic is the stable one having a Long Term Stable (LTS) version available. It takes a couple of months for packages to migrate from previous distributions to the new ones, making LTS a better option than latest unstable release.

Desktop/Laptop Setup

ROS, originally made for Linux environments is still in the process of transitioning its full compatibility to Windows. Therefore, for full functionalities and stable operation, Debian based environments, such as, Ubuntu is preferred. The team has used Ubuntu 16.04 LTS (Xenial Xerus), compatible with ROS Kinetic. For installation on desktop/laptop machine, the standard ROS installation instructions are available on the ROS Wiki site [12].

Raspberry Pi Setup

The project consists of multiple stationary and mobile nodes. Instead of installing Ubuntu on Raspberry Pi, the use of Lubuntu was preferred. Lubuntu (Light Ubunty) is a lighter and faster variant of Ubuntu specifically designed for computers with limited processing power, such as, the Raspberry Pi. A Lubuntu and ROS compiled image is provided by the Ubiquity Robotics. The image, and the complete tutorial on how to install ROS and Lubuntu together can be found on the Ubiquity Robotics site [13]. The image can be directly burned in an SD card, bypassing any need for the installation that was done earlier for the desktop/laptop.

Catkin Work Space Setup

Catkin work space is the build system for ROS. In ROS, targets can be the libraries, packages, or a dynamically created script which is not said to be hard coded. A build system handles the tasks of generating targets from all program scripts. These targets can be used by the end user, without worrying about the underlying processes, or which libraries are used behind the scenes. More details about the Catkin work space can be found on the ROS Wiki site [14].

Once ROS has been installed, the Catkin work space needs to be built and initialized for the first use. The tutorial to setup, and initialize the Catkin work space can be found on the ROS Wiki site [15].

ROS Packages Used

This section briefly touches upon the packages used in the project. All open source, third-party packages can be installed into ROS by cloning their github repositories to the Catkin work space ./src folder, and rebuilding the workspace using the *catkin make* command in the CLI.

Rosserial

The Rosserial package is used to establish communication between the Raspberry Pi and the Arduino. The data that is being transferred over to the Arduino includes the PWM values for the motors, and the lock mechanism commands. The Arduino uses the package to subscribe to the ROS topics, and is able to publish the encoder values back to the Pi. The package uses the standard Universal Asynchronous Receiver Transceiver (UART) connection with 115200 baud rate. More details on this package can be found on the ROS Wiki site [16].

rplidar_ros

The rplidar_ros package is used to channel the RP LIDAR sensor to ROS. The package publishes the scan data on the /scan topic on ROS. More details related to this package can be found on the ROS Wiki site [17].

Navigation Stack

The Navigation Stack is used to navigate the robot control using odometry, and a pre-generated map that was created using the Cartographer library. The group used Rviz to set the current location and pose of the Robot on the map, and to set the goal position of the robot that was desired for it to travel to. The Navigation Stack uses the amcl node for localization, the differential drive node to send its speed and direction, and lastly the map server node to subscribe to the map. The package translates these commands into a trajectory, and transmits it to the robot using the cmd_vel topic. Rviz will not be used for the final demonstration as the whole process is being coded together into a file by the name MainLoop (found in Appendix C.1). More details related to this package can be found on the ROS Wiki site [18].

Gmapping

Gmapping is a ROS package that provides the OpenSlam Gmapping algorithm which constructs a 2D occupancy grid map of the environment, utilizing laser data and odometry information. More information related to the package can be found on the ROS Wiki site [19]. The algorithm can be found on the Openslam site [20].

Gmapping is a SLAM algorithm which builds a map using odometry and sensor

data. It is highly effective when access to reliable odometry is available, and it performs well even in symmetric environments with very less dynamic obstacle variation.

tf

In order to re-use software components, and better integrate them, a shared convention for coordinate frames is required by the developers of libraries, drivers, and models. Shared conventions render a specification for the developers that create drivers and models for mobile bases. The tf package allows the user to keep track of multiple coordinate frames over time. It keeps the relationship between the coordinate frames in a tree structure, buffered in time, and allows the user to transform vectors and points among various other things, between any two coordinate frames, at any time. More information on this can be found on the ROS website [21]. Information related to the tf package can be found on the ROS Wiki site [22].

A visual representation of various coordinate frames can be seen in Figure 4.4

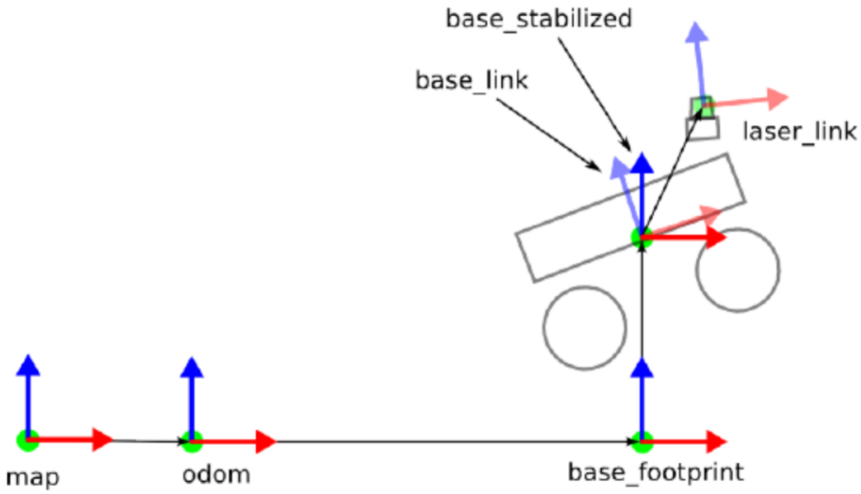


Figure 4.4: The tf Transformed Coordinate Frame for a Differential Drive Robot

tf manages the various transforms that are required to be published for a robot. There are standard terms defined for a robots physical structure, such as, base_frame, laser_frame etc. The robot's frame of reference is the base frame, and it is also referred to as the base link. The laser frame is the coordinate frame of the LIDAR, it is placed on the top of the robot, and is called base laser. Continuous publishing

of transformation from base_link to laser_link is done by the tf, denoting the position of the LIDAR with respect to the center of the robot. The package also maintains dynamic transforms, such as, the odom to base link transform, which represents the position of the robot base frame, base link, with respect to the starting position, which is the odometry. Hence, frame name odom. A visual representation of the system transform tree is shown in Figure 4.5.

Figure 4.5: Visual Representation of System Transform Tree

With ROS, it is possible to publish both the static, and the dynamic transforms. An example of the static transform is the base link→laser link transform. The laser is rigidly fixed to the robot, hence its transform is static with respect to the robot. On the other hand, the odom→base link transform is dynamic since, it keeps changing as the robot moves, hence it is a dynamic transform. This dynamic transform changes are being published by the amcl node.

differential_drive

The differential_drive package provides tools required to interface a differential drive wheeled mobile robot to the ROS Navigation Stack. The purpose of this package is to create an implementation of a differential drive controller that is independent of the specific hardware, for instance, a particular microcontroller unit, used to implement the robot.

This package takes in messages from the Navigation Stack, and converts them into the left wheel and right wheel messages, which go on to serve as the voltage levels transferred to the motor driver in the form of PWM signals. The package also performs the task of receiving the encoder signals from the hardware, and generates messages used in odometry and location referencing for the Navigation Stack, in the form of tf transform messages. To maintain the hardware abstraction, the speed messages are communicated in the form of x-axis metres/second and z-axis radians/second.

The package also provides a PID controller that uses feedback from the rotary encoders attached to the motors to form a closed loop control system to control the wheel velocities. Two PID controllers are instantiated, corresponding to each wheel. The package also provides a node for odometry calculations through the encoders of the motors. The ticks detected by each encoder are counted and published by the low level controller (Arduino) to the differential drive package, using the serial communication package, Rosserial. The odometry node subscribes to these messages, and calculates the odometry values of the robot.

Google Cartographer

The reasons for using Google Cartographer, instead of Hector SLAM and Gmapping are listed as follows:

1. Hector SLAM is an efficient algorithm used to make a map of the environment, and it is what was used earlier by the group to map the environment, however the group later realized that it does not use the odometry information. This can cause some problems while mapping, such as, it does not know where it currently is, and uses the LIDAR to make an assumption of its location. This

makes the map rather unreliable, as it can cause certain map areas to be in those places that do not really align with real world environment.

2. Gmapping overcomes this area, and allows SLAM to be used however, the group did not quite use it because later it was found out that SLAM was not really needed. There was just the need to map the structure, and move the robot about it. Furthermore, some of its unreliabilities are that it makes a map in one go, and it does not make a map on top of it to make sure what it is doing is correct.

Figures 4.6 and 4.7 show Gmapping, Hector SLAM, and Google Cartographer side by side.

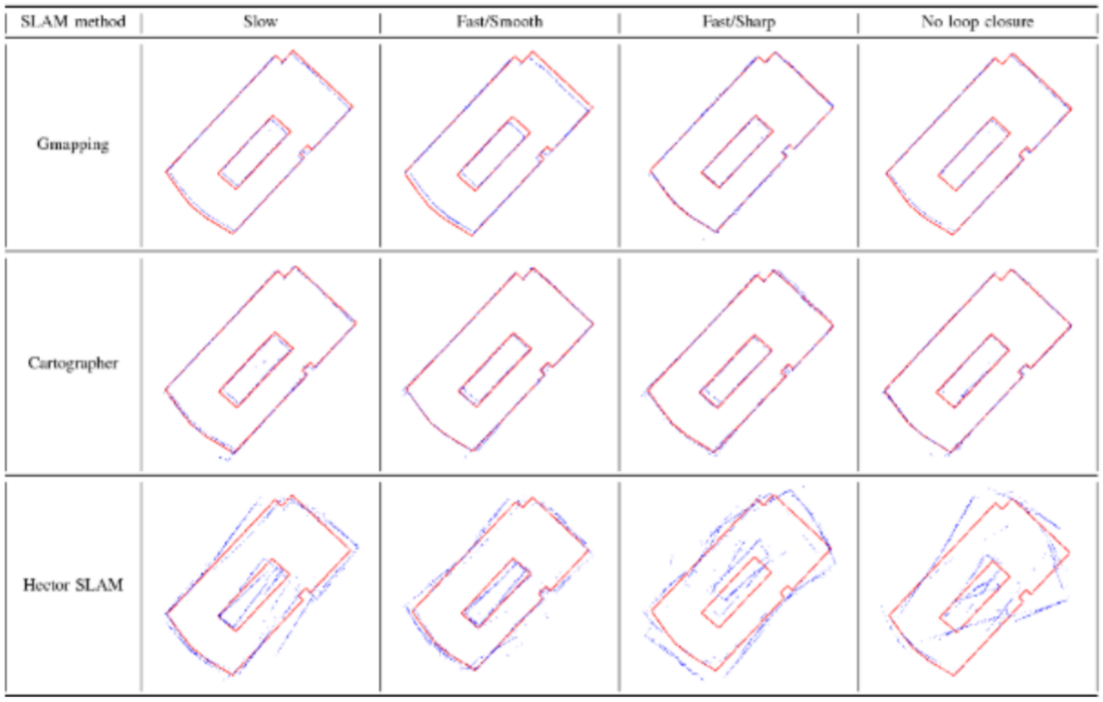
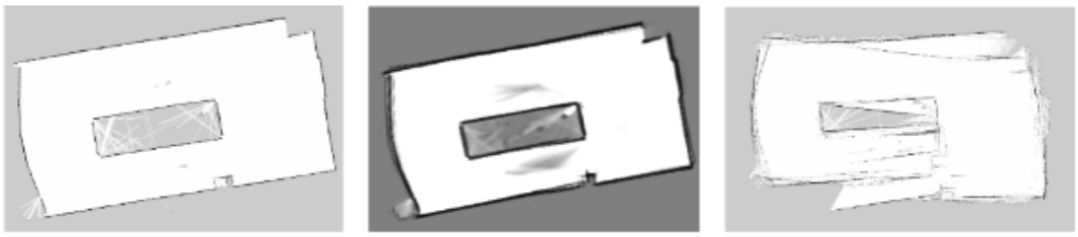


Figure 4.6: An Overview Between Gmapping, Hector SLAM, and Google Cartographer



(a) Gmapping (b) Cartographer (c) Hector SLAM

Figure 4.7: An Overview Between Gmapping, Hector SLAM, and Google Cartographer

This is where the Cartographer comes in, Cartographer is a Google based algorithm that can be used for mapping, as well as, just SLAM. It can improve the map if it is in the area long enough, and can make a map on top of the other map, to make it more efficient. It also uses the odometry information that is provided to it by the Arduino encoders. This makes Cartographer one of the best mapping algorithms, and the optimal solution for the project's scope. More related information can be found on the Google Cartographer site [23].

Figure 4.8 shows an overview of the Google Cartographer package.

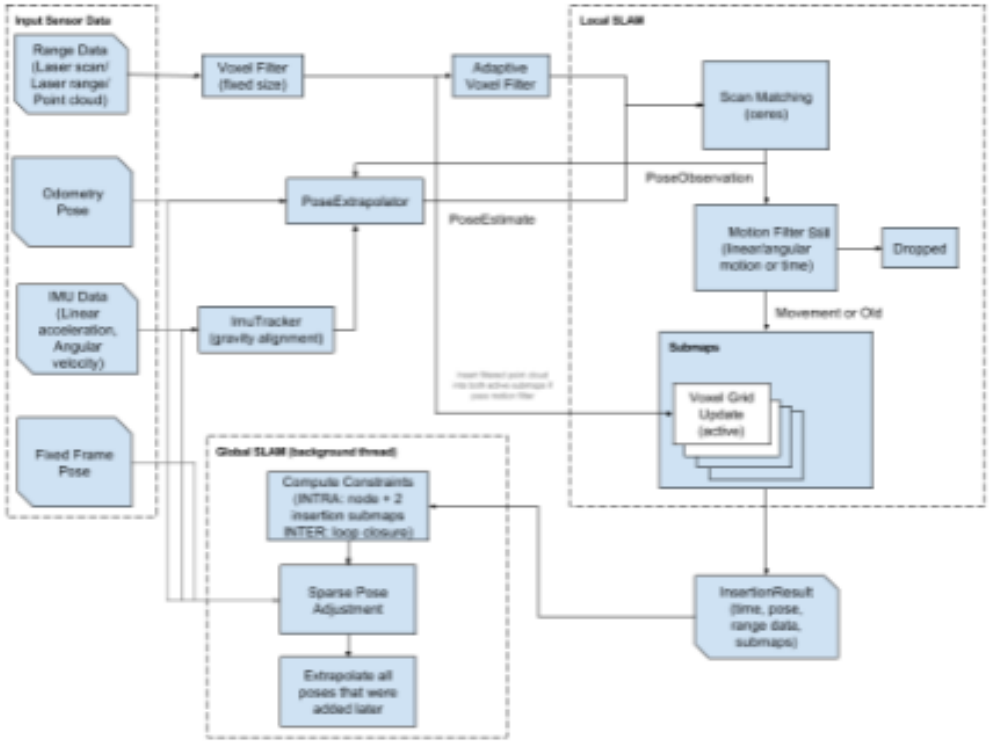


Figure 4.8: High Level Overview of Google Cartographer Package

4.3.2 Facial Recognition

Purpose and Overview

Since, the group wanted to create a solution to be used in indoor spaces, not only to deliver perishables or lunch supplies, but to also aid in the interdepartmental communication, which includes the delivery of essential and confidential packages. It was necessary to make the working system safe and secure for the end user. Hence, the group decided to add a Facial Recognition module to offer an extended biometric security.

A Facial Recognition module can identify or verify a person from a digital image, or a video frame from a video source. It is also known as a Biometric Artificial Intelligence based application that can uniquely identify a person, by analyzing the

patterns based on the person's facial textures and shape.

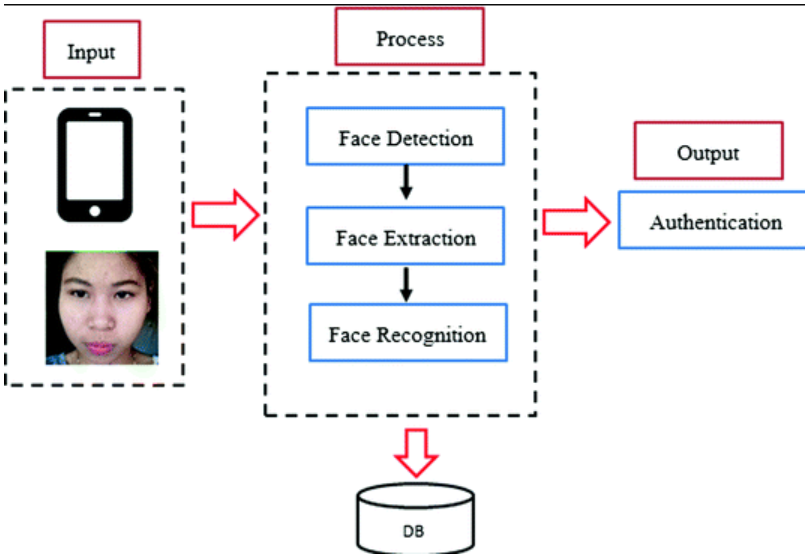


Figure 4.9: Block Diagram of Face recognition flow

Procedure

The Facial Recognition module is built on OpenCV, and the SKlearn modules of Python. The group decided to go for OpenCV since, it is completely open source, and has a tremendous amount of community support. There are two program flows in the Facial Recognition module, Training, and Recognition. The Training module receives a token ID, generated in the app against each user's entry. An input video is given along with the token to train the model. Once the model is successfully trained, it gives the response code ¡201¿. The Recognition module is connected to the robot's on board Pi camera, the robot on completion of the booking prompts the Pi camera to capture an image, and once an image is captured, and sent to the server, the module recognizes, and outputs the token ID. If the token ID matches with the current receiver's token ID, then the Lock API on the Arduino opens the lock successfully.

Face Detection

Face detection falls under a specific case of object-class detection[1]. In object-class detection, the task is to find the locations and sizes of all objects in an image that belong to a given class. Real life examples of this include pedestrians, cars etc.

The Face recognition app can be broken down into two major components, detection and recognition as follows:

- To apply face detection, which detects the presence and location of a face in an image, but does not identify it
- To extract the 128-d feature vectors (called "embeddings") that quantify each face in an image

OpenCV comes with a trainer as well as detector. If you want to train your own classifier for any object like car, planes etc. you can use OpenCV to create one.

Face Recognition

Once we have used OpenCV to detect a face from an input provided. We can process the image further to find whether that image is a match with any of our existing entries or not. To do that we would extract the embeddings (128-d feature vectors) from the input.

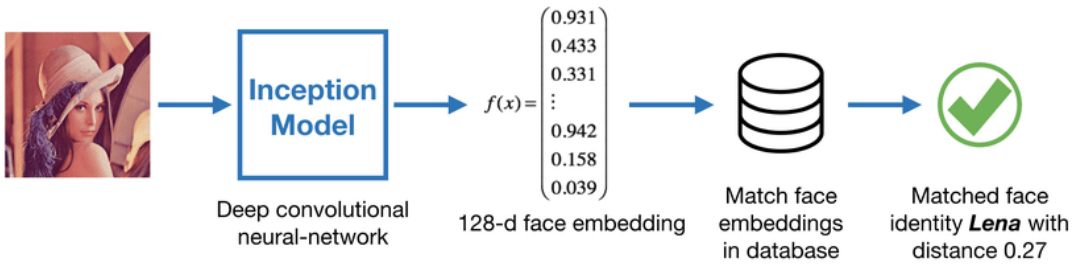


Figure 4.10: Block Diagram of Face recognition module

[1]Further information on Object-class detection at https://dl.acm.org/doi/pdf/10.1145/2522968.2522978?download=true

4.3.3 Android Application

About

This is the module that interacts directly with the user. The user creates an account via the app. The user can then use the app to send packages to other users through the app, by first calling the robot to his or her location, and then tagging the receiver. During the process, the user can view the live location of the robot. The user also has the liberty to view previous deliveries. This module directly communicates with two other modules, i.e, the central server, and the Python server. The app communicates with the central server to place bookings, and with the Python server to send the user's face video for training the model. This allows the user to open the container at the time of receiving the package, by using the Facial Recognition feature.

4.3.4 Servers

Central Server

The central server, as the name suggests, is responsible for all the communication between the different modules of the system. It provides different communication channels, including connection to the app, Facial Recognition server, and the robot.

Load Balancing Server

The load balancing server is present to provide assistance to the Raspberry Pi in handling the computations required by the robot to operate. Due to the Raspberry Pi's limited computational capabilities, it is unable to execute certain control loops at the desired frequency, and hence sharing some of the more computationally expensive programs of the robot to an external server becomes necessary. In this scenario, the load balancing server is nothing but a laptop, running the Ubuntu operating system with ROS installed on it as well. The more resource intensive programs, including the path planning program, is executed on this server, while the rest of the programs are executed on the Raspberry Pi. The communication between the Raspberry Pi, and the load balancing server occurs over WiFi, using an ssh connection.

5. Experiments and Results

5.1 Mapping

A variety of maps were developed to serve as testing environments for the robot. As mentioned in Chapter 4, the map development process uses Google Cartographer, an open source tool developed by Google for 2D and 3D localization and mapping. The SLAM algorithm used by the Cartographer uses the laser range data obtained from the LIDAR to simultaneously situate where the LIDAR is, as well as, develop a map of the surroundings. Without going into too much detail of the mapping algorithm, the scope of which lies beyond this project, laser scans from the LIDAR are successively used to build up a 2D map of the surroundings. Successive laser scans are used to build up submaps, representing small chunks of the surrounding, and these are inserted into a best estimated position. A scoring system among submaps is used to construct the best estimate of the global map from localized chunks [24]. For this project, only the 2D mapping capability of Cartographer is utilized. A map of the Projects Lab at Habib University, developed using the RP LIDAR A2, and the Cartographer, is shown in Figure 5.1.

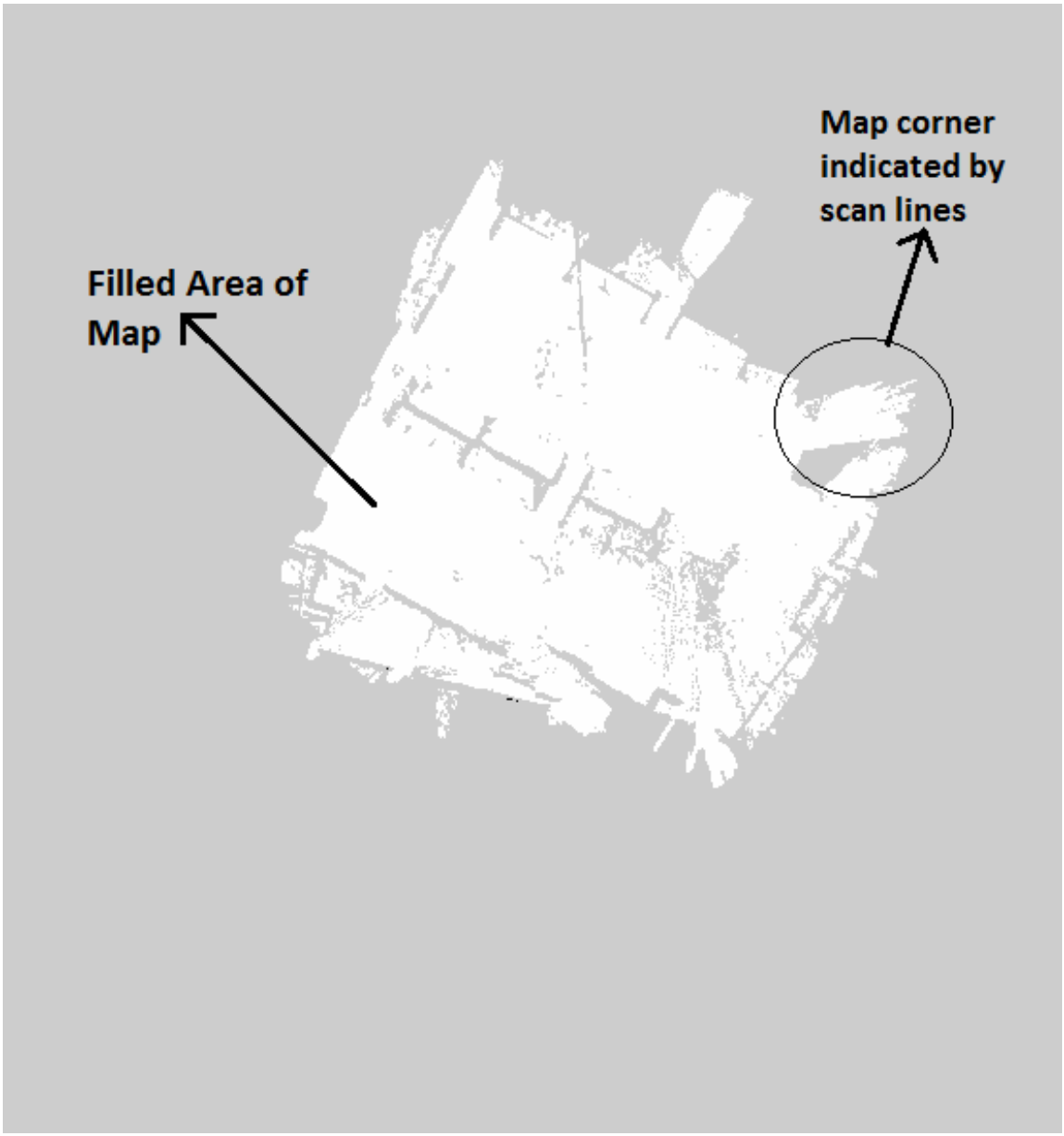


Figure 5.1: Map of Projects Lab

The area filled in, represents the scan lines where no obstructions were detected while mapping. The robot can move around freely within this area. The striations encountered at the edges of the map represent areas where the LIDAR scans could

not detect any obstacles or boundaries, and hence returned scan lines. The mapping process was achieved by using the teleop node of ROS to manually move the robot around the room, and obtain sufficient scans for the LIDAR to build up a best estimate of the surrounding. An ineffective map may be of the robot when it is moved too fast through the environment, which impacts the level of detail which can be recovered from the scans, by affecting the amount of pulse data received by the LIDAR for a given area. Rotating the robot about its position also helps the LIDAR to accumulate more detail for the area, otherwise ineffective maps can be recovered, such as, that shown in Figure 5.2, where sub-areas appear to be merging within the map.

Figure 5.2: Ineffective Map

The other maps that were generated throughout the project are:

Figure 5.3: Projects Lab Map 1

Figure 5.3 shows a map of a portion of the Projects Lab at Habib University, generated using the LIDAR mounted on Prototype 1 during early testing. As the map making process was refined by the group, more accurate maps came about.

Figure 5.4: Projects Lab Map 2

Figure 5.4 shows another map of the same portion, as in Figure 5.3, but with altered movements of the robot to better capture the area.

Figure 5.5: House Map

Figure 5.5 shows a map of a portion of the living room of one of the team members' house. This smaller map was developed in order to test the robot's movement with costmap parameters, set on a smaller map to observe straight line movement for the PID parameter refinement.

For the final testing phase as well as the demonstration, the following map was obtained and utilized, as shown in Figure 5.6:

Figure 5.6: Final Map

The map shown in Figure 5.6 is of a room in a house with the scan lines extending outwards indicating an open door when the mapping process was being implemented. The start and end points for the final demonstration as appearing on the map are shown in Figure 5.7. The start point corresponds to the left side of the room upon entering while the ending point corresponds to the entrance in front of the door.

Figure 5.7: Final Demonstration Start and End Points

5.2 Laser Scanner Simulation

In order to visualize the the capabilities of the laser scanner used by the robot, a simulation was developed using Gazebo, a simulation package that interfaces with ROS. The purpose was also to better visualize the obstacle detection capabilities of a laser scanner. The simulation was developed from the ground up by first defining the parameters of the robot in the Unified Robot Description Format (URDF). URDF is a an XML file format used by ROS to model a robot by defining all its elements e.g. wheel radii, joints, links, general physical parameters etc. A laser scanner link was also defined and attached with the robot. This allowed the configuration of 360° laser scans allowing the observation of the laser's response with different obstacles placed around it. The purpose of this entire exercise was purely for visualization and understanding. The following figures show some of the results of the simulations.

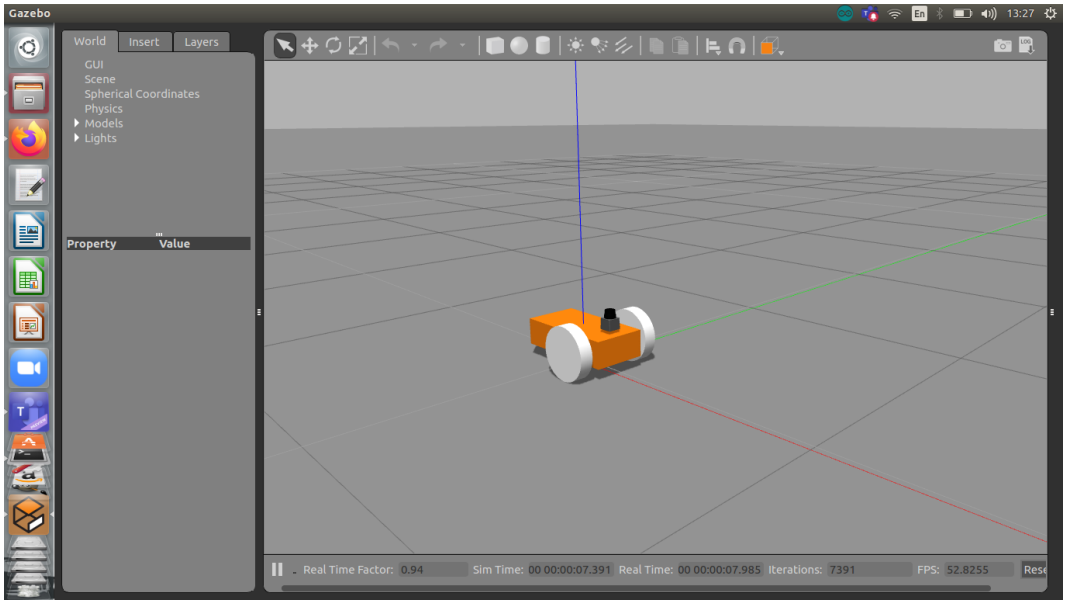


Figure 5.8: Robot Model in Simulator

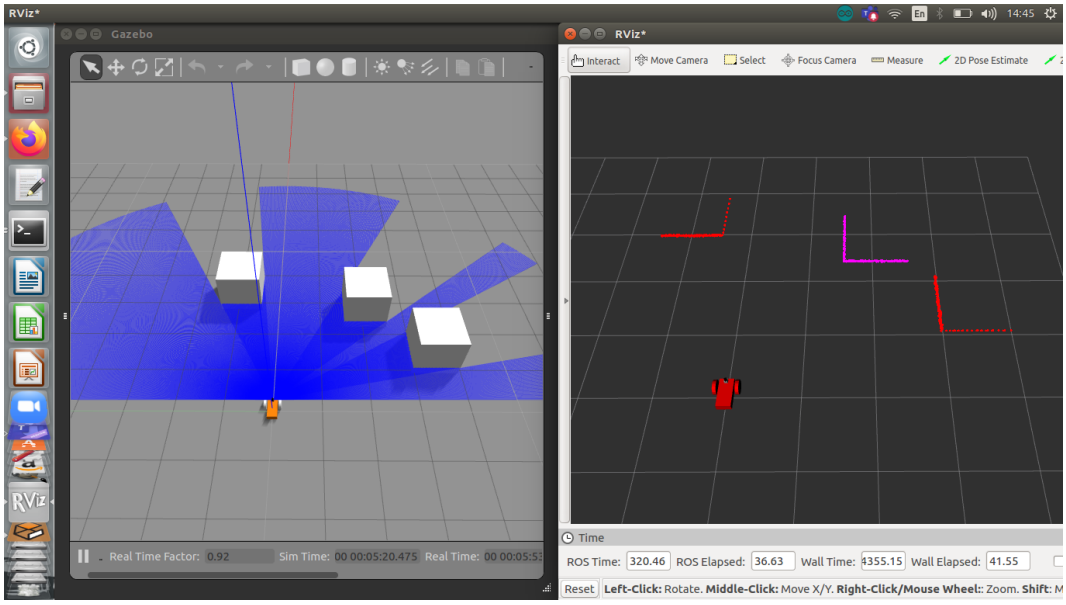


Figure 5.9: 180° Scan with Obstacles - Gazebo and Rviz Visualizations

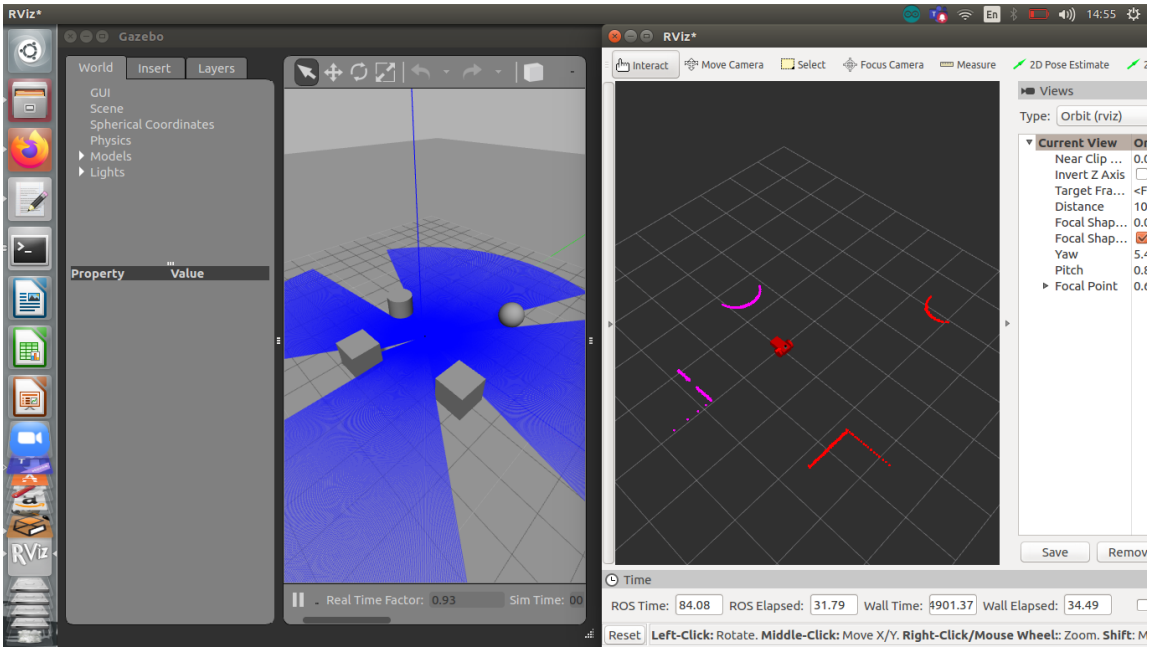


Figure 5.10: 360° Scan with Obstacles - Gazebo and Rviz Visualizations

Figures 5.9 and 5.10 show the obstacle detection simulation environment together with the concurrent visualization of the data received by the scanner in Rviz. Rviz allows the observation of the point cloud data received by the laser scanner in order to make out the edges of the obstacles detected.

6. Conclusion and Future Work

6.1 Conclusion

The fundamental goal of this project, involving the development of an end-to-end automated delivery solution, with an app based front-end, a server based back-end, and an autonomous robot, as the main delivery agent, is realized.

The key features of the robot achieved include:

- A differential drive based robot that possesses the ability to generate a map of its surroundings, based on manual operation, and also the ability to navigate autonomously within an environment, given a known map and reference coordinates.
- The use of a cutting edge sensor, such as, a LIDAR for environmental perception including map building, localization, and obstacle detection and avoidance.
- The use of path planning and navigation algorithms, based on the A-star algorithm to implement autonomous navigation behavior within the robot.
- The use of mobile and web-app development to implement a front-end to enhance the user experience, and implement a point of contact and information management for the delivery solution.
- The use of server side technology for data management, and the implementation of seamless communication between the front-end, and the robot.

6.2 Future Work

A Raspberry Pi V 2.1 Camera module is used in the final prototype. This camera offers a resolution of 8 Megapixels which is not good enough for the Facial Recognition

to function adequately. The group was not able to identify the cause of errors in the module due to the lighting conditions, either it was due to the camera, or due to the algorithm used. Also, the group did not perform rigorous testing on the Facial Recognition module of the project, and, as a result, there is no metric available to justify how accurate, or how reliable the module is. Looking at the future, a better camera can be installed on the prototype, particularly that of a higher resolution, and rigorous testing can be performed on the Facial Recognition module to make the module highly accurate, reliable, and robust according to the environmental conditions.

Going forward, the group has also realized the need to rely on multiple sensors, and sensor fusion, rather than just relying on the LIDAR for mapping, and obstacle detection and avoidance etc. This would enable a more robust environmental perception system. Computer vision, a field of artificial intelligence which trains the computers to understand and interpret the visual world, is an avenue worth exploring for understanding the robot's environment. It makes use of deep learning models, videos, and cameras to help machines identify and classify objects, and consequently, react to what they see. Using cameras as additional input sensors, and employing novel computer vision algorithms can greatly reduce the chances of errors, and bring about a great deal of robustness and accuracy in the sensing functionalities of the robot. Using sensor fusion to combine the inputs of various sensors can also lead to the development of a more robust, and less error prone environmental perception model, by reducing the dependence on any one particular sensor for perception.

The Raspberry Pi model 3B used has a Quad Core 64 bit CPU with 1 GB of RAM to handle the computational tasks, including mapping and path planning algorithms, performing high and low level communication, and handling peripherals. This amount of RAM is not enough to efficiently run some programs. As an alternative, future groups can look toward using a more powerful single board computer, such as, a Raspberry Pi 4, to handle computationally intensive tasks.

While working on the project, the team realized the importance of using simulations to test and predict hypotheses, and system behaviors, and so, going forward, the use of simulation tools can serve as an indispensable measure for testing, and relating theoretical data with real world insights. A focus on data gathering and analysis via simulations should thus be a component to be considered for any future work.

Appendix A. Prototypes

A.1 First Prototype

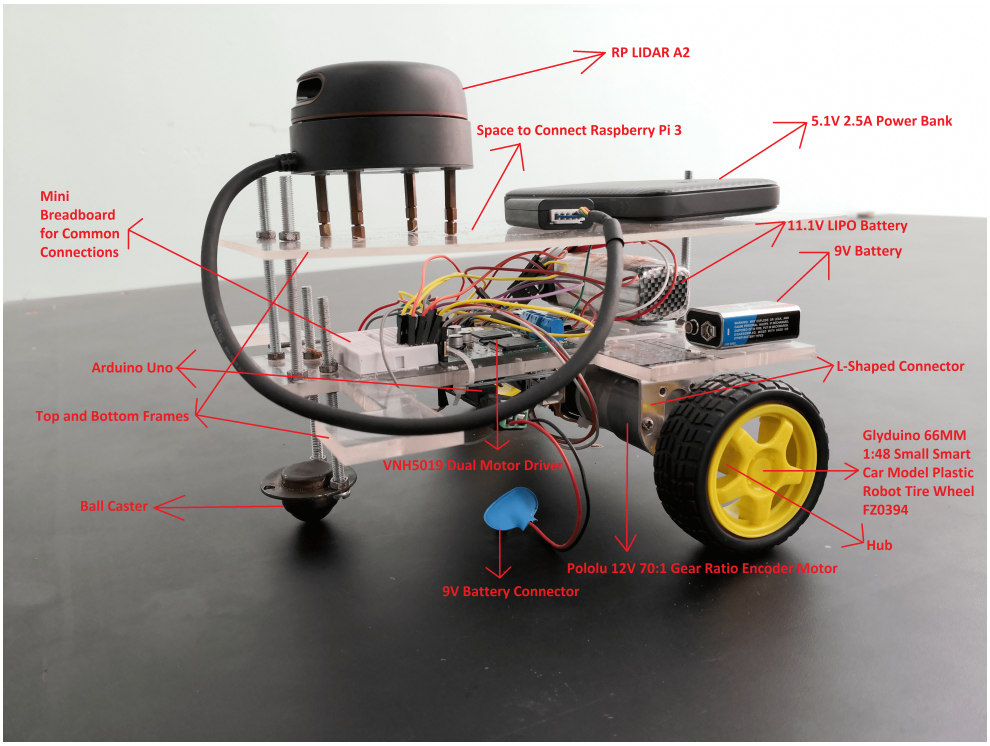


Figure A.1: First Prototype

Our first robot prototype, shown in Figure A.1, was composed of two levels. It had three wheels, with two of them connected to an encoder motor each, and one connected as a free rotating wheel/ball caster to keep the body in balance. Through

our experience with testing, and via literature review, it was identified that the center of gravity in such a robot design has to lay inside the triangle formed by the wheels. If too heavy of a mass is mounted to the side of the free rotating wheel, the robot tips over. All of our major algorithm tests i.e. localization and mapping, PID/locomotion optimization, autonomous navigation, and obstacle detection and avoidance were performed on this prototype. However, due to the fact that this design was not suitable for the integration of the payload delivery system, with the facial recognition and lock mechanisms, the group decided that it was best to come up with a new design, and integrate all of the prior work in it during the final stages of the capstone project.

A.2 Final Prototype

Using the Google Sketchup Pro software, the final robot prototype design proposed in Capstone 1 is shown in Figure A.2.

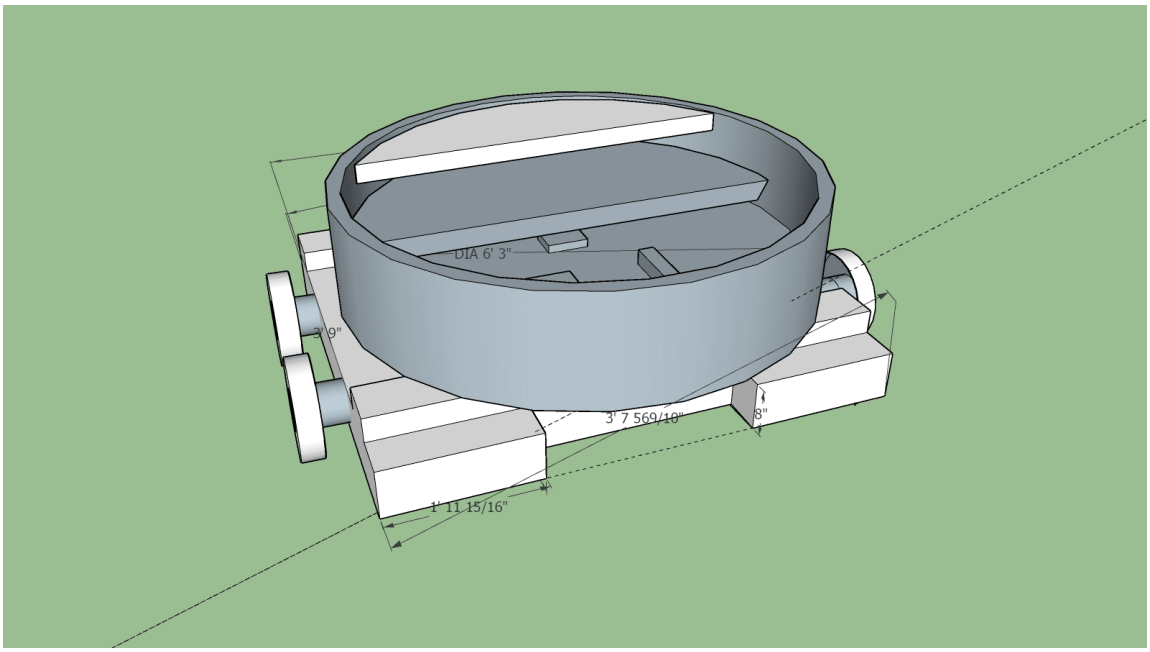


Figure A.2: Final Prototype First Variant

Due to the complexity of the proposed design, and the unavailability of the 3D

printing facility, this design was abandoned.

The re-imagined final robot prototype is composed of two driving wheels, connected to an encoder motor each, along with two ball casters, one at the front and the other at the back, to keep the body in balance. This design is more stable than the three wheel version because the center of gravity is to remain inside the quadrilateral formed by the four wheels, instead of a triangle. The design makes use of two different frames, rectangular in shape, and connected to each other via metallic spacers. The width of the robot chassis is kept 45 cm whereas, the length is kept 35 cm. According to the literature, such a design allows the front wheels to give negligible resistance to the overall circular motion generated by the rear wheels when the robot takes a zero-radius 360 degree turn (pivot turning). Therefore, the torque applied by motors on each wheel gets harnessed properly, this results in a better performance without any drag or wheel slip while taking a turn.

The final robot prototype structure is shown in Figure A.3, A.4, and A.5 respectively.

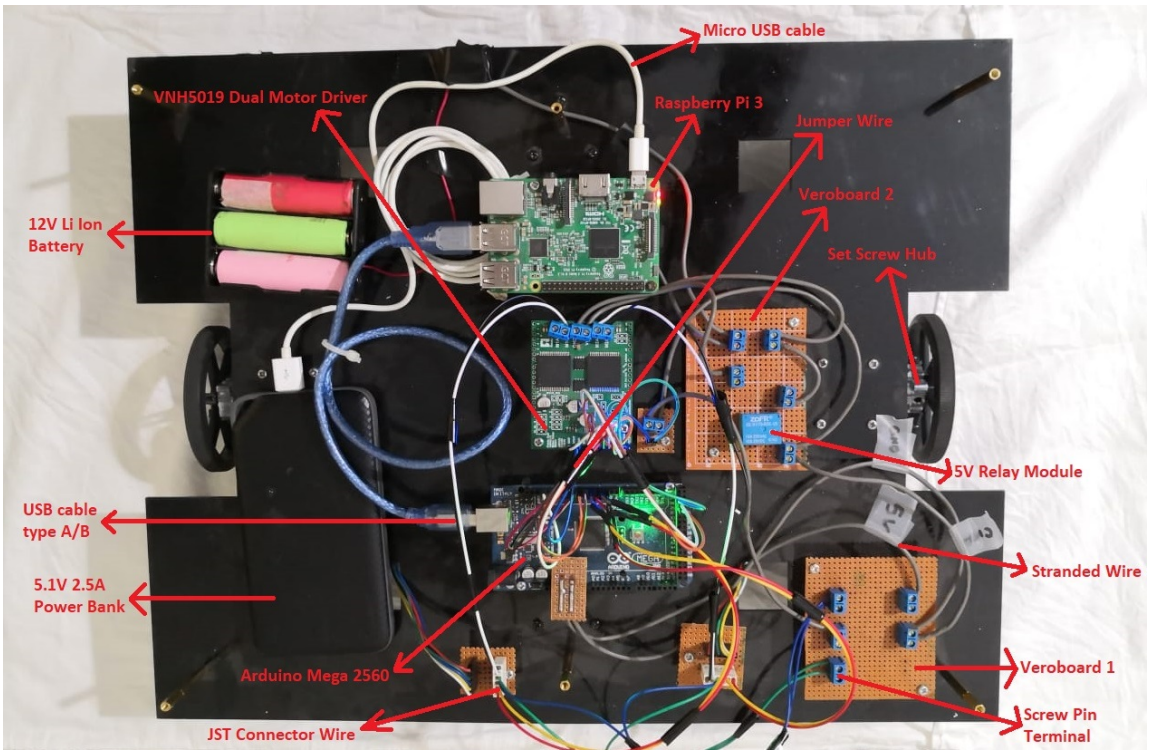


Figure A.3: Internal View

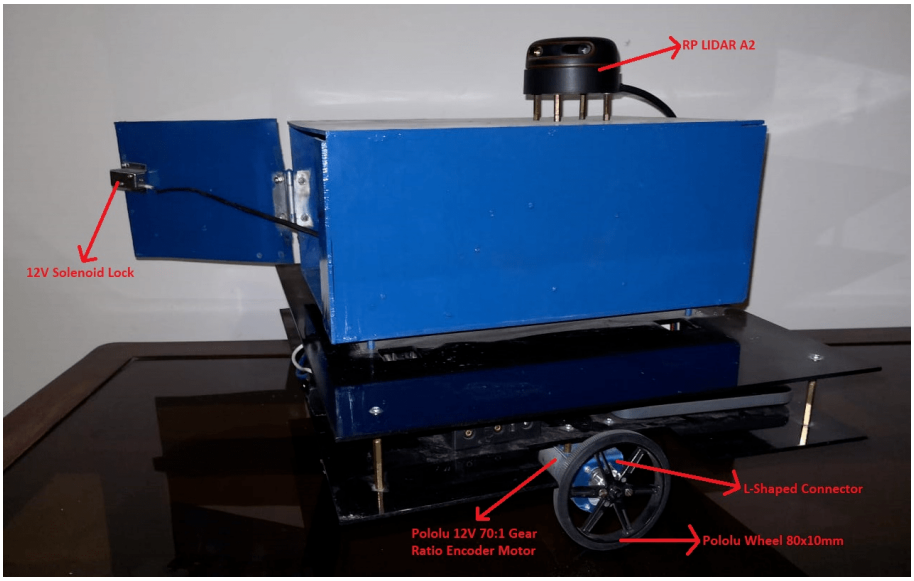


Figure A.4: Side View

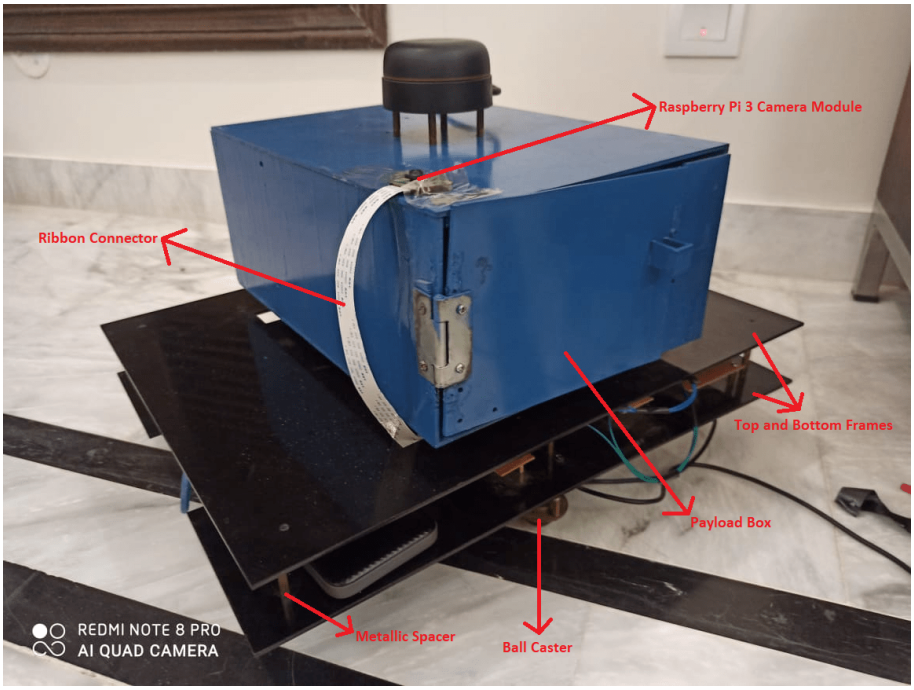


Figure A.5: Elevated View

Several small and large sized metallic spacers are used for the installation of circuit and structural components. This ensured neatness in execution and also made troubleshooting relatively easier to perform. Acrylic sheets are used for the frames and the payload box because they are light in weight and durable. Ball casters are used because they are more stable as compared to wheel casters. Pololu wheels are used as they have a better grip on smooth or tiled surfaces. The other wheels considered were better suited to grassy and rough surfaces. Hubs are used to mount the wheels to the encoder motor shafts. The L-shaped connectors are used to grip the encoder motors.

Appendix B. Data

B.1 Application Data

The user's table can be seen in B.1. The details were added for testing purposes and in no way represents the real data that we collected as it is not ethical to share user information of others.

No.	Name	Email	Phone Number
117	Sarfraz	sarfraz@gmail.com	
116	Hamza	syedhamza19@gmail.com	
115	Syed Hamza Azeem	syedhamza19@gmail.com	
114	Sahran Riaz	sahran.srv@gmail.com	
113	Hal	syedhamza19@gmail.com	
112	Syed Hamza Azeem	syedhamza19@gmail.com	
111	Syed Hamza Azeem	syedhamza19@gmail.com	
110	Hamza	syedhamza19@gmail.com	
109	Sameer Jaliawala	sj02732@st.habib.edu.pk	
108	Hamza	syedhamza19@gmail.com	
107	Azeem	admin@admin.admin	
106	Azeem	admin@admin.admin	
105	Hal	admin@admin.admin	
104	Jagjeet	jugnuboy@gmail.com	
103	Bzbs	haha	
102	Haha	haha	
101	Sameer Anees1	sameeranees1@gmail.com	
100	jnsajnj	nj	
99	Fgg	yuu	
98	ahsa	sjs	
97	Sarfraz Hussain	sarfraz.hussayn900@gmail.com	
96	Sameer Anees	aneessameer@ymail.com	
95	Johndoe	elgoogclass2@gmail.com	
94	Sarfraz Hussain	sarfraz.hussayn900@gmail.com	
93	Ahsan	snahmed1998@gmail.com	
92	Jsj	udjd	
91	Dddx	dd@ghh	
90	Dddx	dd@ghh	
89	asljkd	sadjkn	
88	Uehs	hshs	
87	Vsh	hshs	
86	Yd	gd	
85	Sameer	ahsan	
83	Ahsan	ahsan	
68	Dr. taj	taj@gmail.com	

Figure B.1: Users

The test bookings, that were carried out, can be visible in B.2.

Booking ID	From	To	Status	Sender	Receiver	Date time
205	right side	left side	completed	Hamza	Sameer Jaliawala	6/18/2020 18:28
204	right side	left side	completed	Hamza	Sameer Jaliawala	6/18/2020 17:49

Figure B.2: Bookings

Appendix C. Code

C.1 Main Loop

```
#!/usr/bin/env python
import requests
import json, os
import cv2
import rospy
from std_msgs.msg import Float32
import random
import time
from new import funct #uncomment this when on pi

headers={'content-type' : 'image/jpg'}

currentBookingInfo = ""
currentLocation = [1,1]
customerId = None

pub = rospy.Publisher('locker', Float32, queue_size=10)
rospy.init_node('talker', anonymous=False)

URL='https://186ad9c5edee.ngrok.io/upload'

def send_nodes(img_file=None):
   f=open("Gray.jpg","rb")
   img={'file': f}
   response=requests.post(URL,files=img)
   response=response.json()
```

```
    print(response)
    return response['message']

def getNextBookingDetails(currentBookingId):
    URL = "https://api.anomoz.com/api/swift/post/
        ↪ read_new_booking.php?currentBookingId="+str(
        ↪ currentBookingId)
    r = requests.get(url = URL)
    # extracting data in json format
    data = r.json()
    info = (data[0])
    return info

def markBookingAsEnded(currentBookingId):
    URL = "https://api.anomoz.com/api/swift/post/
        ↪ mark_booking_as_ended.php?currentBookingId="+str(
        ↪ currentBookingId)
    r = requests.get(url = URL)
    return True

def getPickupLocation(currentBookingInfo):
    print("getPickupLocation set: ", [currentBookingInfo["
        ↪ fromLocation_lat"], currentBookingInfo["
        ↪ fromLocation_lng"]])
    return([currentBookingInfo["fromLocation_lat"],
        ↪ currentBookingInfo["fromLocation_lng"]])

def getFinalLocation(currentBookingInfo):
    print("getFinalLocation set: ", [currentBookingInfo["
        ↪ toLocation_lat"], currentBookingInfo["
        ↪ toLocation_lng"]])
    return([currentBookingInfo["toLocation_lat"],
        ↪ currentBookingInfo["toLocation_lng"]])

#def startNavigator(currentLocation, destination):
    #programCommand = "rosrun simple_navigation_goals
        ↪ simple_navigation_goals _param_x:="+str(destination
        ↪ [0])+" _param_y:="+str(destination[1])
```

```
    #if os.system(programCommand) == 0:
        #print("program finished successfully")
    #else:
        #print("launch failed")

def startNavigator(resp):
    # programCommand = "rosrun simple_navigation_goals
        ↪ simple_navigation_goals _param_x:="+str(destination
        ↪ [0])+" _param_y:="+str(destination[1])
    # if os.system(programCommand) == 0:
    # print("program finished successfully")
    # else:
    # print("launch failed")
    URL = "http://192.168.86.28:2015/movebase?resp="+str(
        ↪ resp[0])+"&resp1="+str(resp[1])
    r = requests.get(url = URL)
    return True

def generateExe():
    if os.system("g++ -g foo.cpp -o foo") == 0:
        print ("exe compiled")
    else:
        print ("exe compiling Failed")

def open_lock(data):
    pub.publish(data)
    # try:
        #rospy.init_node('talker', anonymous=True)
     #except:
        #print("some")
    print("hello")
    rospy.Subscriber("open_lock", Float32, callback)
    rospy.sleep(5.0)

def callback(data):
    print(data.data,"this is data")
# time.sleep(5.0)
# if data.data=="open":
```

```
# time.sleep(10.0)
# pub.publish("close")
# time.sleep(5.0)
# if data.data=="close":
# rospy.sleep(10.0)
# # return "some"
# # rospy.signal_shutdown("The lid has been closed")
# # return "some"

def Get_pin():
   #send the number as a callback to customer to enter pin
   #get callback from app that the pin has been entered
      ↪ successfully
   try:
      file1 = open("lockstatus.txt","r+")
      fileinp = file1.read()
      file1.close()
      if(fileinp=="opened"):
         return True
      else:
         return False
   except:
      return False

def runSingleCycle():

   currentBookingInfo = getNextBookingDetails(0)
   #currentBookingInfo = {'bookingId': '187', 'timeAdded':
      ↪ '1573634449', 'fromLocation': 'Auditorium', '
      ↪ toLocation': 'Auditorium', 'fromPerson': 'Dr. taj',
      ↪  'toPerson': 'Ahsan', 'status': 'waiting', '
      ↪ fromLocation_lat': '1.17', 'fromLocation_lng':
      ↪ '0.77', 'toLocation_lat': '0.17', 'toLocation_lng':
      ↪  '-0.775'}
   print("currentBookingInfo␣updated:␣", currentBookingInfo
      ↪ )
   customerId=currentBookingInfo['toPerson']
   #go to pickup location
```

```
    pickupLocation = getPickupLocation(currentBookingInfo)
    print(startNavigator(pickupLocation)) #program stops
        ↪ until
    #print(startNavigator(currentBookingInfo))
    #open lock for pickup
    open_lock(1)

    #to to destination
    dropoffLocation = getFinalLocation(currentBookingInfo)
    print("dropoffLocation", dropoffLocation)
    startNavigator(dropoffLocation)

    #openlock for dropoff
    name=""
    if(Get_pin()==False):
        try:
            name=send_nodes(funct())
        except:
            print("no")
    if name==customerId:
        open_lock(1)
        print("Need␣pin␣to␣unlock")
    else:
        while(True):
            if(Get_pin()==True):
                open("lockstatus.txt", 'w').close()
                break
    open_lock(1)
    #mark booking
    markBookingAsEnded(currentBookingInfo['bookingId'])

    #runSingleCycle()


def main():
    #generateExe()

    runSingleCycle()
```

```
    #verifyUser()
    #pass



    #get next booking Status
# print(send_nodes(funct())) #uncomment this when on pi
main()
rospy.signal_shutdown("The lid has been closed")
#print()
```

C.2 open_lock.py

```
from flask import Flask
app = Flask(__name__)
from flask import request

@app.route('/openlock')

def openlock():
    isAuth = request.args.get('auth')
    print("inp", isAuth)
    if(isAuth=="true"):
        file1 = open("lockstatus.txt","w")
        file1.write("opened")
        file1.close()
        #open_lock("open")
        return "true"
    else:
        return "false"

if __name__ == '__main__':
    app.run(host= '0.0.0.0',port=4010)
```

C.3 Server.py

```
import os
import urllib.request
from app import app
from flask import Flask, request, redirect, jsonify
from werkzeug.utils import secure_filename
# from werkzeug.utils import secure_filename
from Recognizer import indexe
from split import splitter
from train_model import train_function
from extract_embeddings import extract_function

ALLOWED_EXTENSIONS = set(['mp4','txt', 'pdf', 'png', 'jpg',
   ↪  'jpeg', 'gif'])

def allowed_file(filename):
      return '.' in filename and filename.rsplit('.', 1)
         ↪ [1].lower() in ALLOWED_EXTENSIONS

@app.route('/upload', methods=['POST'])
def upload_file():
      # check if the post request has the file part
      if 'file' not in request.files:
            resp = jsonify({'message' : 'No␣file␣part␣in␣
               ↪ the␣request'})
            resp.status_code = 405
            return resp
      file = request.files['file']
      if file.filename == '':
            resp = jsonify({'message' : 'No␣file␣selected␣
               ↪ for␣uploading'})
            resp.status_code = 406
            return resp
      if file and allowed_file(file.filename):
            filename = secure_filename(file.filename) #
               ↪ Creates Files name
```

```
        basedir = os.path.abspath(os.path.dirname(
            ↪ __file__)) #Fetches Base DIR
        file.save(os.path.join(basedir,'./UPLOAD_FOLDER
            ↪ ', filename)) #Saves File on the UPLOAD
            ↪ Destination

        if filename.rsplit('.', 1)[1].lower()=="mp4":
              splitter(filename.rsplit('.', 1)[0],"./
                 ↪ UPLOAD_FOLDER/"+str(filename))
              extract_function()
              train_function()
              resp = jsonify({'message:␣Upload␣Done'})
              resp.status_code = 201
              return resp


        ########OBSOLETE#########
        #destination = file.save(os.path.join(os.path.
            ↪ abspath(os.path.dirname(__file__)),app.
            ↪ config['UPLOAD_FOLDER'], filename))
        #print(os.path.join(os.path.abspath(os.path.
            ↪ dirname(__file__)),app.config['
            ↪ UPLOAD_FOLDER'], filename))
        #print(destination)
        ##########################
        banda=indexe("./UPLOAD_FOLDER/"+str(filename))
        #Runs Recognizer on the Last Uploaded File
        print(banda)
        #Will print the Response of Facial Recognition

        print(filename)

        resp = jsonify({'message' : banda})
        resp.status_code = 201
        return resp
    else:
        resp = jsonify({'message' : 'Allowed␣file␣types
            ↪ ␣are␣txt,␣pdf,␣png,␣jpg,␣jpeg,␣gif'})
```

```
            resp.status_code = 400
            return resp

if __name__ == "__main__":
   app.run(host='0.0.0.0',port=5000, debug=True)
```

C.4 Recognizer.py

```
# USAGE
# python recognize.py --detector face_detection_model \
# --embedding-model openface_nn4.small2.v1.t7 \
# --recognizer output/recognizer.pickle \
# --le output/le.pickle --image images/adrian.jpg

# import the necessary packages
import numpy as np
import argparse
import imutils
import pickle
import cv2
import os
from PIL import Image
import requests
from io import BytesIO
# from skimage import io
import urllib.request
import json
import imghdr

#import sklearn

def indexe(inter):
   try:
      hdr = {'User-agent': 'Mozilla/5.0'}
      #req = 'C:\\Users\\UN\\Desktop\\IMAGE_HERE.jpg' #"
         ↪ https://raw.githubusercontent.com/ahsansn/
         ↪ workspace/master/IMG_20191126_100411670.jpg"
```

```
    #return "yaaa"

    # if(req == None):
    # return "nothing provided"

    img_o= inter
    #"https://ik.imagekit.io/demo/img/newPages/
       ↪ dl_SysFHc2jm.png"
    #"https://api.anomoz.com/testing/pic.jpg"#"https://
       ↪ images.pexels.com/photos/736716/pexels-photo
       ↪ -736716.jpeg?auto=compress&cs=tinysrgb&dpr=1&w
       ↪ =500"
    img_ = '.\images\mukesh.jpg'

######## construct the argument parser and parse the
   ↪ arguments
    ap = argparse.ArgumentParser()
    ap.add_argument("-i", "--image", default=inter,
          help="path␣to␣input␣image")
    ap.add_argument("-d", "--detector", default='
       ↪ face_detection_model',
          help="path␣to␣OpenCV's␣deep␣learning␣face␣
             ↪ detector")
    ap.add_argument("-m", "--embedding-model", default='
       ↪ openface_nn4.small2.v1.t7',
          help="path␣to␣OpenCV's␣deep␣learning␣face␣
             ↪ embedding␣model")
    ap.add_argument("-r", "--recognizer", default='output
       ↪ /recognizer.pickle',
          help="path␣to␣model␣trained␣to␣recognize␣faces"
             ↪ )
    ap.add_argument("-l", "--le", default='output/le.
       ↪ pickle',
          help="path␣to␣label␣encoder")
    ap.add_argument("-c", "--confidence", type=float,
       ↪ default=0.5,
```

```
            help="minimum probability to filter weak
                ↪ detections")
    args = vars(ap.parse_args())
#
    ↪ ##########################################################
    ↪

##### load our serialized face detector from disk
    #print("[INFO] loading face detector...")
    protoPath = "./face_detection_model/deploy.prototxt"
        ↪ #os.path.sep.join([args["detector"], "deploy.
        ↪ prototxt"])
    #return (protoPath)
    #r"face_detection_model/deploy.prototxt"#os.path.sep.
        ↪ join([args["detector"], "deploy.prototxt"])
    modelPath = "./face_detection_model/
        ↪ res10_300x300_ssd_iter_140000.caffemodel"
    #os.path.sep.join([args["detector"],"
        ↪ res10_300x300_ssd_iter_140000.caffemodel"])
    detector = cv2.dnn.readNetFromCaffe(protoPath,
        ↪ modelPath)

    # load our serialized face embedding model from disk
    #print("[INFO] loading face recognizer...")
    embedder = cv2.dnn.readNetFromTorch("./"+ args["
        ↪ embedding_model"])
    #
        ↪ ##########################################################
        ↪

    # load the actual face recognition model along with
        ↪ the label encoder
    recognizer = pickle.loads(open( "./" + args["
        ↪ recognizer"], "rb").read())
    le = pickle.loads(open("./" + args["le"], "rb").read
        ↪ ())
```

```
#
   ↪ ##########################################################
   ↪

# load the image, resize it to have a width of 600
   ↪ pixels (while
# maintaining the aspect ratio), and then grab the
   ↪ image dimensions
'''
print(img_o)
img = Image.open(BytesIO((requests.get(img_o)).
   ↪ content))
print("--", img_)
'''
#
   ↪ ##########################################################
   ↪

#image = cv2.imread(img)

image = cv2.imread(inter)
#print(image)
'''
with urllib.request.urlopen(img_o) as url:
      req = url.read()

#req = urllib.urlopen(img_o)
arr = np.asarray(bytearray(req.read()), dtype=np.
   ↪ uint8)
image = cv2.imdecode(arr, -1)
'''
#urllib.request.urlretrieve(img_o, "hello.jpg")
#print("-here")
#imaasdsdge = cv2.imread("hello.jpg")


#print(image)
```

```
#
   ↪ ##########################################################################
   ↪

image = imutils.resize(image)
(h, w) = image.shape[:2]

# construct a blob from the image
imageBlob = cv2.dnn.blobFromImage(
      cv2.resize(image, (300, 300)), 1.0, (300, 300),
      (104.0, 177.0, 123.0), swapRB=False, crop=False
         ↪ )

# apply OpenCV's deep learning-based face detector to
   ↪  localize
# faces in the input image
detector.setInput(imageBlob)
detections = detector.forward()
naam=[]
# loop over the detections

for i in range(0, detections.shape[2]):
      # extract the confidence (i.e., probability)
         ↪ associated with the
      # prediction
      confidence = detections[0, 0, i, 2]

      # filter out weak detections
      if confidence > args["confidence"]:
            # compute the (x, y)-coordinates of the
               ↪ bounding box for the
            # face
            box = detections[0, 0, i, 3:7] * np.array
               ↪ ([w, h, w, h])
            (startX, startY, endX, endY) = box.astype
               ↪ ("int")

            # extract the face ROI
```

```
face = image[startY:endY, startX:endX]
(fH, fW) = face.shape[:2]

# ensure the face width and height are
   ↪ sufficiently large
if fW < 20 or fH < 20:
      continue

# construct a blob for the face ROI, then
   ↪  pass the blob
# through our face embedding model to
   ↪ obtain the 128-d
# quantification of the face
faceBlob = cv2.dnn.blobFromImage(face,
   ↪ 1.0 / 255, (96, 96),
      (0, 0, 0), swapRB=True, crop=False)
embedder.setInput(faceBlob)
vec = embedder.forward()

# perform classification to recognize the
   ↪  face
preds = recognizer.predict_proba(vec)[0]
j = np.argmax(preds)
proba = preds[j]
name = le.classes_[j]
# draw the bounding box of the face along
   ↪  with the associated
# probability
text = "{}: {:.2f}%".format(name, proba *
   ↪  100)
y = startY - 10 if startY - 10 > 10 else
   ↪ startY + 10
cv2.rectangle(image, (startX, startY), (
   ↪ endX, endY),
      (0, 0, 255), 2)
cv2.putText(image, text, (startX, y),
      cv2.FONT_HERSHEY_SIMPLEX, 0.45, (0,
         ↪  0, 255), 2)
```

```
                    naam.append(name)
    except IndexError:
        print("Type Unknown")
    #print(naam[0])#print(name)
    #cv2.imshow("Image",image)
    #cv2.waitKey(0)
    # if os.path.exists("saved_img.jpg"):
    # os.remove("saved_img.jpg")
    # else:
    # print("The file does not exist")
    #return json.dumps({"personName": name[0] })
    returne = "[INFO] Type Unknown please try again..."
    try:
        return naam[0]
    except (IndexError, ValueError) as e:
        pass
#
    ↪ ##########################################################################
    ↪

# show the output image

# print(indexe("./Gray.jpg")) #run the main loop to return
    ↪ the name of the recognized person
#'.\\saved_img.jpg'
#print(x)
```

C.5 call_this.py

```
from flask import Flask
app = Flask(__name__)
from flask import request
import os

def getPickupLocation(currentBookingInfo):
```

```
    print("getPickupLocation set: ", [currentBookingInfo["
        ↪ fromLocation_lat"], currentBookingInfo["
        ↪ fromLocation_lng"]])
    return([currentBookingInfo["fromLocation_lat"],
        ↪ currentBookingInfo["fromLocation_lng"]])

def getFinalLocation(currentBookingInfo):
    print("getFinalLocation set: ", [currentBookingInfo["
        ↪ toLocation_lat"], currentBookingInfo["
        ↪ toLocation_lng"]])
    return([currentBookingInfo["toLocation_lat"],
        ↪ currentBookingInfo["toLocation_lng"]])

def startNavigator(destination):
    programCommand = "rosrun simple_navigation_goals
        ↪ simple_navigation_goals __param_x:="+str(
        ↪ destination[0])+" _param_y:="+str(destination[1])
    if os.system(programCommand) == 0:
        return "True"
    else:
        return "False"

@app.route('/movebase')
def movebase():
    isAuth = request.args.get('resp')
    isAuth2 = request.args.get('resp1')
    var=[float(isAuth),float(isAuth2)]
    print(var)
    # isAuth= request.args.get('resp2')
    # pickupLocation = getPickupLocation(isAuth)
    # dropoffLocation = getFinalLocation(isAuth)

    resp = startNavigator(var)

    # return resp
    return resp
```

```
if __name__ == '__main__':
    app.run(host="0.0.0.0",port=2015)
```

C.6 base_local_planner_params.yaml

```
TrajectoryPlannerROS:
  max_vel_x: 0.4
  min_vel_x: 0.2
  max_rotational_vel: 0.5
  max_vel_theta: 1.5
  min_vel_theta: -1.5
  min_in_place_rotational_vel: 0.25
  min_in_place_vel_theta: 0.7
  escape_vel: -0.25
  acc_lim_theta: 0.3
  acc_lim_x: 0.5
  acc_lim_Y: 0.07
  holonomic_robot: false
  meter_scoring: true
  xy_goal_tolerance: 0.15
  yaw_goal_tolerance: 1.7
```

C.7 costmap_common_params.yaml

```
obstacle_range: 1.4
raytrace_range: 2.0

max_obstacle_height: 0.2
min_obstacle_height: 0.05

robot_radius: 0.20
inflation_radius: 0.55

transform_tolerance: 5.0
```

```
map_type: costmap
cost_scaling_factor: 100

map_topic: /map
subscribe_to_updates: true

observation_sources: laser_scan_sensor


laser_scan_sensor: {sensor_frame: laser, data_type:
   ↪ LaserScan, topic: scan , marking: true, clearing: true
   ↪ }
```

C.8 global_costmap_params.yaml

```
global_costmap:
  global_frame: /map
  robot_base_frame: base_link
  update_frequency: 5.0
  rolling_window: false
  static_map: true
  publish_frequency: 1.0
  width: 15
  height: 15
  resolution: 0.1
```

C.9 local_costmap_params.yaml

```
local_costmap:
  global_frame: odom
  robot_base_frame: base_link
  update_frequency: 5.0
  publish_frequency: 2.0
  static_map: false
  rolling_window: true
```

```
 width: 2
 height: 2
 resolution: 0.025
```

C.10 trajectory_planner.yaml

```
TrajectoryPlannerROS:
 max_vel_x: 0.4
 min_vel_x: 0.2
 max_rotational_vel: 0.5
 max_vel_theta: 1.5
 min_vel_theta: -1.5
 min_in_place_rotational_vel: 0.25
 min_in_place_vel_theta: 0.7
 escape_vel: -0.25
 acc_lim_theta: 0.3
 acc_lim_x: 0.5
 acc_lim_Y: 0.07
 holonomic_robot: false
 meter_scoring: true
 xy_goal_tolerance: 0.15
 yaw_goal_tolerance: 0.3
 latch_xy_goal_tolerance: true
```

C.11 my_robot_configuration.launch

```
<launch>
<arg name="map_file" default=" /home/ubuntu/sameer_house.
   ↪ yaml"/>
<node name="map_server" pkg="map_server" type="map_server"
   ↪ args="$(arg map_file)">
    <param name="frame_id" type="str" value="map"/>
</node>
<include file="$(find amcl)/examples/amcl_diff.launch"/>
<rosparam param="ticks_meter">17825</rosparam>
```

```
<node pkg="differential_drive" type="diff_tf.py" name="
   ↪ odometry" output="screen">
      <remap from="lwheel" to="left_encoder" />
      <remap from="rwheel" to="right_encoder" />
      <rosparam param="base_width">0.40</rosparam>
      <rosparam param="odom_frame_id" subst_value="True" >
         ↪ "/odom" </rosparam>
      <rosparam param="base_frame_id" subst_value="True">
         ↪ "/base_link" </rosparam>
      <rosparam param="global_frame_id" subst_value="True">
         ↪  "/map" </rosparam>
      <rosparam param="rate">50</rosparam>
</node>
<node pkg="differential_drive" type="pid_velocity.py" name
   ↪ ="lpid">
      <remap from="wheel" to= "left_encoder"/>
      <remap from="motor_cmd" to= "left_pwm"/>
      <remap from="wheel_vtarget" to= "left_target"/>
      <remap from="wheel_vel" to= "left_actual"/>
      <rosparam param="Kp">400</rosparam>
      <rosparam param="Ki">10</rosparam>
      <rosparam param="Kd">1</rosparam>
      <rosparam param="out_min">-255</rosparam >
      <rosparam param="out_max">255</rosparam >
      <rosparam param="rate">40</rosparam >
      <rosparam param="timeout_ticks">4</rosparam>
      <rosparam param="rolling_pts">5</rosparam>
</node>

<node pkg="differential_drive" type="pid_velocity.py" name
   ↪ ="rpid">
      <remap from="wheel" to="right_encoder"/>
      <remap from="motor_cmd" to="right_pwm"/>
      <remap from="wheel_vtarget" to="right_target"/>
      <remap from="wheel_vel" to="right_actual"/>
      <rosparam param="Kp">450</rosparam>
      <rosparam param="Ki">10</rosparam>
      <rosparam param="Kd">1</rosparam>
```

```
      <rosparam param="out_min">-255</rosparam>
      <rosparam param="out_max">255</rosparam>
      <rosparam param="rate">40</rosparam>
      <rosparam param="timeout_ticks">4</rosparam>
      <rosparam param="rolling_pts">5</rosparam>
</node>

<node pkg="differential_drive" type="twist_to_motors.py"
   ↪ name="twist" output="screen">
      <remap from="twist" to="cmd_vel"/>
      <remap from="lwheel_vtarget" to="left_target"/>
      <remap from="rwheel_vtarget" to="right_target"/>
      <rosparam param="base_width">0.40</rosparam>
</node>
<node pkg="rosserial_python" type="serial_node.py" name="
   ↪ serial_node">
      <param name="port" value="/dev/ttyACM0"/>
      <param name="baud" value="57600"/>
</node>

<node pkg="tf" type="static_transform_publisher" name="
   ↪ laser_transform" args="0 0 0.14 4.71 0 0 base_link
   ↪ laser 100"/>
</launch>
```

C.12 move_base.launch

```
<launch>

  <master auto="start"/>
  <node pkg="move_base" type="move_base" respawn="false"
     ↪ name="move_base" output="screen">
   <rosparam file="$(find my_robot_name_2dnav)/
      ↪ costmap_common_params.yaml" command="load" ns="
      ↪ global_costmap" />
   <rosparam file="$(find my_robot_name_2dnav)/
      ↪ costmap_common_params.yaml" command="load" ns="
```

```
      ↪ local_costmap" />
   <rosparam file="$(find my_robot_name_2dnav)/
      ↪ local_costmap_params.yaml" command="load" />
   <rosparam file="$(find my_robot_name_2dnav)/
      ↪ global_costmap_params.yaml" command="load" />
   <rosparam file="$(find my_robot_name_2dnav)/
      ↪ base_local_planner_params.yaml" command="load" />
   <param name="base_global_planner" type="string" value="
      ↪ navfn/NavfnROS" />
   <param name="controller_frequency" type="double" value
      ↪ ="10.0"/>
   <param name="planner_frequency" type="double" value
      ↪ ="10.0"/>
 </node>

</launch>
```

C.13 simple_navigation_goals.cpp

```
#include <ros/ros.h>
#include <move_base_msgs/MoveBaseAction.h>
#include <actionlib/client/simple_action_client.h>
#include<iostream>

typedef actionlib::SimpleActionClient<move_base_msgs::
   ↪ MoveBaseAction> MoveBaseClient;

int main(int argc, char** argv){
  ros::init(argc, argv, "simple_navigation_goals");
  std::float param_x;
  std::float param_y;

  ros::NodeHandle nh("~");

  nh.getParam("param_x",param_x);
  nh.getParam("param_y",param_y);
  std::cout << param_x;
```

```
  //tell the action client that we want to spin a thread by
     ↪ default
  MoveBaseClient ac("move_base", true);

  //wait for the action server to come up
  while(!ac.waitForServer(ros::Duration(5.0))){
    ROS_INFO("Waiting␣for␣the␣move_base␣action␣server␣to␣
       ↪ come␣up");
  }

  move_base_msgs::MoveBaseGoal goal;

  //we'll send a goal to the robot to move 1 meter forward
  goal.target_pose.header.frame_id = "base_link";
  goal.target_pose.header.stamp = ros::Time::now();


  goal.target_pose.pose.position.x = param_x;
  goal.target_pose.pose.position.y = param_y;
  goal.target_pose.pose.orientation.w = 1;

  ROS_INFO("Sending␣goal");
  ac.sendGoal(goal);

  ac.waitForResult();

  if(ac.getState() == actionlib::SimpleClientGoalState::
     ↪ SUCCEEDED)
    ROS_INFO("Hooray,␣the␣base␣moved␣1␣meter␣forward");
  else
    ROS_INFO("The␣base␣failed␣to␣move␣forward␣1␣meter␣for␣
       ↪ some␣reason");

  return 0;
}
```

C.14 Ardunio Code Interfaced with ROS

```
# include <Encoder.h>
# include <ros.h>
# include <std_msgs/Int16.h>
# include <geometry_msgs/Twist.h>
# include <std_msgs/Float32.h>
# include <std_msgs/String.h>
# include <TimerOne.h>

Encoder knobLeft(2,4) ;
Encoder knobRight(3,5);

#define LOOP_TIME 100000
#define ENA 6
#define ENB 9
#define IN1 24
#define IN2 26
#define IN3 7
#define IN4 8
int relay_pin= 22;
float left_pwm, right_pwm;

ros :: NodeHandle nh;
std_msgs::Int16 left_wheel_enc;
std_msgs::Int16 right_wheel_enc;
std_msgs::Int16 right_wheel_values;
std_msgs::Int16 left_wheel_values;
std_msgs::Float32 open_lock_value;
std_msgs::Float32 check_open_lock;

ros :: Publisher check_open_lock_pub("open_lock",&
   ↪ check_open_lock);
ros :: Publisher left_wheel_enc_pub("left_encoder",&
   ↪ left_wheel_enc);
ros :: Publisher right_wheel_enc_pub("right_encoder",&
   ↪ right_wheel_enc);
void timerIsr()
```

```
{
  Timer1.detachInterrupt();//stopthetimer
  right_wheel_enc.data=knobRight.read();
  left_wheel_enc.data=knobLeft.read();
  right_wheel_enc_pub.publish(&right_wheel_enc);
  left_wheel_enc_pub.publish(&left_wheel_enc);
  Timer1.attachInterrupt(timerIsr);//enablethetimer
}

void cmdLeftWheelCB(const std_msgs::Float32& msg)
{
   left_wheel_values.data = msg.data;
}

void cmdRightWheelCB(const std_msgs::Float32& msg)
{
  right_wheel_values.data = msg.data;
}

void open_lock(const std_msgs::Float32& msg)
{
  if(msg.data > 0){
    check_open_lock.data=msg.data;
    check_open_lock_pub.publish(&check_open_lock);
    digitalWrite(relay_pin, HIGH);
    delay(5000);
    check_open_lock.data=0;
    check_open_lock_pub.publish(&check_open_lock);
    digitalWrite(relay_pin, LOW);
  }
}

ros::Subscriber<std_msgs::Float32> subCmdLeft("left_pwm",&
   ↪ cmdLeftWheelCB);
ros::Subscriber<std_msgs::Float32> subCmdRight("right_pwm"
   ↪ ,&cmdRightWheelCB);
ros::Subscriber<std_msgs::Float32> openLock("locker",&
   ↪ open_lock);
```

```
void setup() {
  // put your setup code here, to run once:
  Serial.begin(57600);
  pinMode (ENA, OUTPUT) ;
  pinMode (ENB, OUTPUT) ;
  pinMode ( IN1 , OUTPUT) ;
  pinMode ( IN2 , OUTPUT) ;
  pinMode ( IN3 , OUTPUT) ;
  pinMode ( IN4 , OUTPUT) ;
  pinMode ( relay_pin, OUTPUT) ;
  analogWrite(ENA,0);
  analogWrite(ENB,0);
  digitalWrite(IN1,LOW);
  digitalWrite(IN2,LOW);
  digitalWrite(IN3,LOW);
  digitalWrite(IN4,LOW);
  Timer1.initialize(LOOP_TIME);
  nh.initNode();
  nh.subscribe(subCmdLeft);
  nh.subscribe(subCmdRight);
  nh.subscribe(openLock);
  nh.advertise(left_wheel_enc_pub);
  nh.advertise(right_wheel_enc_pub);
  nh.advertise(check_open_lock_pub);

  Timer1.attachInterrupt(timerIsr);
  left_wheel_values.data=0;
  right_wheel_values.data=0;
}

void loop() {

  nh.spinOnce();

  if (left_wheel_values.data > 0 && right_wheel_values.data
     ↪ > 0) //FWD Condition
  {
```

```
  digitalWrite(IN1,HIGH); //right forward
  digitalWrite(IN2,LOW);
  digitalWrite(IN3,HIGH); //left forward
  digitalWrite(IN4,LOW);
}
else if (left_wheel_values.data < 0 && right_wheel_values.
   ↪ data < 0)
{
  digitalWrite(IN1,LOW); //right backwards
  digitalWrite(IN2,HIGH);
  digitalWrite(IN3,LOW); //left backwards
  digitalWrite(IN4,HIGH);
}

else if (left_wheel_values.data > 0 && right_wheel_values.
   ↪ data < 0)
{
  digitalWrite(IN1,LOW); //right backwards
  digitalWrite(IN2,HIGH);
  digitalWrite(IN3,HIGH); //left forward
  digitalWrite(IN4,LOW);
}

else if (left_wheel_values.data == 0 && right_wheel_values
   ↪ .data == 0)
{
  digitalWrite(IN3,LOW); //left backward
  digitalWrite(IN4,LOW);
  digitalWrite(IN1,LOW); //right forward
  digitalWrite(IN2,LOW);
}

else
{
  digitalWrite(IN3,LOW); //left backward
  digitalWrite(IN4,HIGH);
  digitalWrite(IN1,HIGH); //right forward
  digitalWrite(IN2,LOW);
```

```
  }

    analogWrite(ENB, abs(left_wheel_values.data));
    analogWrite(ENA, abs(right_wheel_values.data));
}
```

References

[1] A. G. Özkil, Z. Fan, S. Dawids, H. Aanæs, J. K. Kristensen, and K. H. Christensen, "Service robots for hospitals: A case study of transportation tasks in a hospital," IEEE International Conference on Automation and Logistics, 2009, pp. 289–294.

[2] J. Kocić, N. Jovičić, and V. Drndarević, "Sensors and sensor fusion in autonomous vehicles," in *2018 26th Telecommunications Forum (TELFOR)*, 2018, pp. 420–425.

[3] R. Simmons, R. Goodwin, K. Z. Haigh, S. Koenig, and J. O'Sullivan, "A layered architecture for office delivery robots," in *Proceedings of the First International Conference on Autonomous Agents*, ser. AGENTS '97, Marina del Rey, California, USA: Association for Computing Machinery, 1997, pp. 245–252, ISBN: 0897918770. DOI: 10.1145/267658.267723. [Online]. Available: https://doi.org/10.1145/267658.267723.

[4] M. Sokolov., O. Bulichev., and I. Afanasyev., "Analysis of ros-based visual and lidar odometry for a teleoperated crawler-type robot in indoor environment," in *Proceedings of the 14th International Conference on Informatics in Control, Automation and Robotics - Volume 2: ICINCO,*, INSTICC, SciTePress, 2017, pp. 316–321, ISBN: 978-989-758-264-6. DOI: 10.5220/0006420603160321.

[5] J. M. Santos, D. Portugal, and R. P. Rocha, "An evaluation of 2d slam techniques available in robot operating system," in *2013 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*, 2013, pp. 1–6.

[6] M. D. Rossetti, A. Kumar, and R. A. Felder, "Mobile robot simulation of clinical laboratory deliveries," Winter Simulation Conference, 1998, pp. 1415–1422.

[7] J. Evans, B. Krishnamurthy, W. Pong, R. Croston, C. Weiman, and G. Engelberger, "Helpmate: A Robotic Materials Transport System," *Robotics and Autonomous Systems*, pp. 251–256, 1989.

[8] "Quarterly E-Commerce Report 1st Quarter 2018," U.S. Department of Commerce, Washington, D.C., Publication CB18-74, 2018.

[9] X. Du, M. H. Ang, and D. Rus, "Car detection for autonomous vehicle: Lidar and vision fusion approach through deep learning framework," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017, pp. 749–754.

[10] X. Du and K. K. Tan, "Comprehensive and practical vision system for self-driving vehicle lane-level localization," *IEEE Transactions on Image Processing*, vol. 25, no. 5, pp. 2075–2088, 2016.

[11] G. Zhao, X. Xiao, J. Yuan, and G. W. Ng, "Fusion of 3d-lidar and camera data for scene parsing," *Journal of Visual Communication and Image Representation*, vol. 25, no. 1, pp. 165–183, 2014, Visual Understanding and Applications with RGB-D Cameras, ISSN: 1047-3203. DOI: https://doi.org/10.1016/j.jvcir.2013.06.008. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1047320313001211.

[12] *Installation/ubuntu*, http://wiki.ros.org/Installation/Ubuntu, ROS.org, 2020.

[13] *Installation/ubuntu*, https://downloads.ubiquityrobotics.com/pi.html, ROS.org, 2020.

[14] *Catkin*, http://wiki.ros.org/catkin, ROS.org, 2020.

[15] *Creating a workspace for catkin*, http://wiki.ros.org/catkin/Tutorials/create_a_workspace, ROS.org, 2020.

[16] *Rosserial*, http://wiki.ros.org/rosserial_python?distro=kinetic, ROS.org, 2020.

[17] *Rplidar*, http://wiki.ros.org/rplidar, ROS.org, 2020.

[18] *Navigation*, http://wiki.ros.org/navigation, ROS.org, 2020.

[19] *Gmapping*, http://wiki.ros.org/gmapping, ROS.org, 2020.

[20] *Openslam*, https://openslam-org.github.io/gmapping.html, ROS.org, 2020.

[21] *Coordinate frames for mobile platforms*, https://www.ros.org/reps/rep-0105.html, ROS.org, 2020.

[22] *Tf*, http://wiki.ros.org/tf, ROS.org, 2020.

[23] https://google-cartographer.readthedocs.io/en/latest/, 2020.

[24] W. Hess, D. Kohler, H. Rapp, and D. Andor, "Real-time loop closure in 2d lidar slam," 1998.

[25] C. Gartenberg, *Amazon is now delivering packages in Southern California with its Scout robots*, http://www.theverge.com/2019/8/9/20798604/amazon-scout-robot-delivery-irving-southern-california-expansion-prime, 2019.

[26] A. J. Hawkins, *Thousands of autonomous delivery robots are about to descend on us college campuses*, https://www.theverge.com/2019/8/20/20812184/starship-delivery-robot-expansion-college-campus, 2019.

[27] E. Ackerman, *Sony Partners with CMU to Develop Food Prep and Delivery Robots*, https://spectrum.ieee.org/automaton/robotics/home-robots/sony-partners-with-cmu-to-develop-food-prep-and-delivery-robots, 23 April 2018.

[28] T. S. Perry, *CES 2018: Delivery Robots Are Full-Time Employees at a Las Vegas Hotel*, https://spectrum.ieee.org/view-from-the-valley/robotics/industrial-robots/ces-2018-delivery-robots-are-fulltime-employees-at-a-las-vegas-hotel, 12 Jan 2019.

[29] K. Rogers, *Higher Demand for Quick Delivery Is Creating a Boom in Jobs*, https://www.cnbc.com/2018/05/04/higher-demand-for-quick-delivery-is-creating-a-boom-in-jobs.html, CNBC News, 2018.

[30] A. Weiner, *Minor Roadblocks Stand in the Way of Personal Delivery Devices*, https://thespoon.tech/minor-roadblocks-stand-in-the-way-of-personal-deliverydevices/, The Spoon, 2018.

[31] M. Harris, *"our Streets Are Made for People : San Francisco Mulls Ban on Delivery Robots*, https://www.theguardian.com/sustainable-business/2017/may/31/delivery-robots-drones-san-francisco-public-safety-job-lossfears-marble, The Guardian, 2018.

[32] A. Zaleski, *San Francisco to Delivery Robots: Get Off the Damn Sidewalk*, https://www.citylab.com/life/2017/05/san-francisco-to-delivery-robots-get-off-the-damnsidewalk/527460/, Citylab, 2018.

[33] B. Zhang, *Personal Delivery Devices to Expand across Bay Area*, https://thecampanile.org/2017/12/07/personal-delivery-devices-to-expand-across-bay-area/, The Campanile, 2018.

[34] H. Prasaath, *Choosing motors for robots*, https://www.engineersgarage.com/egblog/choosing-motor-for-robots/, Engineers Garage, 2020.

[35] S. T. Wasim, F. Ahmed, and F. Farooq, "Squadbot: A multi-agent robotics teaching and research platform," Undergraduate Capstone Report for Habib University, May 2019.