Silver Retriever

Clive Toh Soon, Wong cw3915@ic.ac.uk
Louis Kueh Han Sheng lhk115@ic.ac.uk
Samuel Yang Kai Ern sky15@ic.ac.uk
Un Kei Leong ul15@ic.ac.uk
Kavin Winson kw2614@ic.ac.uk
Guo Liang Liew gll15@ic.ac.uk
Jin Yee Ng jyn15@ic.ac.uk

Abstract

This paper introduces Silver Retriever, a robot capable of assisting humans in delivery goods. Equipped with sensors and connected to the cloud, the robot can be dispatched to deliver item according to the user's requirement via a web app. The user can receive real-time status, location and video update from the robot, avoiding obstacles along the way at the same time. Currently only three locations are supported, but the system is built to be versatile and scalable, such that new features and locations can be added non-disruptively. The self-explanatory user interface allows familiarity in less than 30s. Software for the alarm system has been partly built, shall more time be given, hardware and user authentication could have been built to support it.

1. Introduction

The online retail industry evolves rapidly in the past few years as consumers become increasingly accustomed to online shopping for maximum convenience and time-saving. Statistics show that more and more people are shopping online, including senior citizens. According to the Office for National Statistics, the proportion of older citizens shopping online has grown from 16% to 48% over the past decade.[5] This online shopping trend promotes the growth of the parcel delivery market where customers demand fast or even the same-day delivery. This presents the opportunities for delivery automation as delivery automation can make the delivery process more efficient and secure for both the consumers and retailers. The project is motivated by these trends aims to develop and test an operating model of a delivery robot system for public usage. With suitable adjustments and modifications, the system could be scaled and adopted in different industries, e.g. the industrial manufacturing workplace for the transport of goods.

The testing ground is on the university campus with a pre-mapped environment to test and verify the following hypotheses.

- Is the operation model of the Silver Retriever user-friendly for the ease of use and adoption by the public (even for senior citizens)?
- Does the use of Silver Retriever system improve the home delivery efficiency and user experience?

The following evaluation parameters are examined to verify the hypotheses stated above:

- Percentage of successful deliveries (public testing)
- How fast can the public user learn how to use the system.
- Average time needed by the robot to make one complete cycle of the delivery process over a certain distance.

The results will then be used to investigate the limiting factors in the successful robot delivery, and possible solutions to the problems are suggested.

2. Background

Start-ups Marble, Nuro and Starship Technologies are offering delivery robot services, which is an increasingly popular trend of retail automation involving artificial intelligence (AI) and robotics[6]. The delivery robots currently available on the market are equipped with cameras for computer vision, radar and LiDAR for 3D city mapping to aid their navigation in the busy cities. Their sole purpose is to deliver items such as food, groceries and packages from the shops or restaurants to the customers' doorsteps, thus they operate on city streets and neighbourhood sidewalks.

Marble has been developing fully autonomous delivery robots that can navigate the chaos of a city sidewalk, where pedestrians, pets and peddlers of bicycles engage in an unchoreographed dance that can defy logic which the robot is implemented with cameras, LiDAR and high-resolution 3D city maps to navigate[6].

Starship Technologies is equipped with cameras with 360-degree coverage and a radar system enabling the robot to navigate in the city, avoid obstacles and exercise good pedestrian behaviour throughout its journey [2]. The robots are equipped with speakers and microphones to interact with humans.

Relay by Savioke is an indoor delivery robot designed to provide logistics services in hospitality and services industry or in work areas in manufacturing facilities[7]. Relay is equipped with sensors that enable it to navigate building floors and elevators, travelling at a human walking pace.

Out of the three Silver Retriever is most similar to Relay. A key aspect in which Silver Retriever differs is the additional connections to the internet of things. Having a real-time map, video & interface to visualise the robot's path on any platform brings a new level of user experience to the end user.

3. System Design

This section describes the overall design of the robotic system to achieve the goal of the project. Various design alternatives for the overall system, software and hardware are considered and justified to achieve the optimal human-robot interaction experience.

3.1. Metrics for evaluation

To evaluate the system, the metrics shown in Table 1 have been proposed to justify the decisions made. The following sections will evaluate available choices based on the proposed metrics.

Aspects	Description
Cost	As the goal is to build a working prototype with a limited budget, lower component or service cost is preferred
Performance	Performance here refers to the latency between the request and callback. In the case of software implementation, a short latency (or time required) to execute a command is favoured.
Durability	This applies more to hardware such that the component will last longer or able to withstand higher external force before it breaks. In terms of software, it refers to the duration of the use of software before it is expired, deprecated or no longer supported by the provider.
Scalability	This refers to the versatility of the system or software to scale up before the performance drops or the system becomes overcomplicated to develop further.
Flexibility	Flexibility is defined as how well the component can be replaced by alternatives or used with other components to improve the system.
Accuracy	Accuracy here refers to the closeness between the desired output and the actual output from the component, where closer is better.
Precision	The consistency of the component to deliver the output given a fixed input.
Safety and Security	The ability of the system to avoid collision, theft (physical or informational) and intentional damage.
Ease of use	How easy the component can be installed onto the system and used by the users.
Power Consumption	The amount of power required to operate the component.

Table 1: Metrics for Evaluation

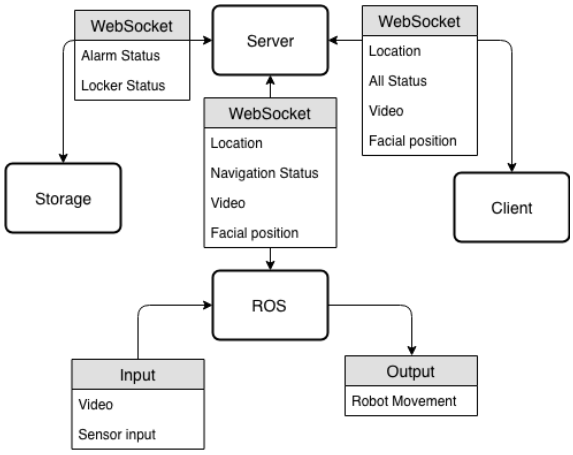

Figure 1: System Architecture 1

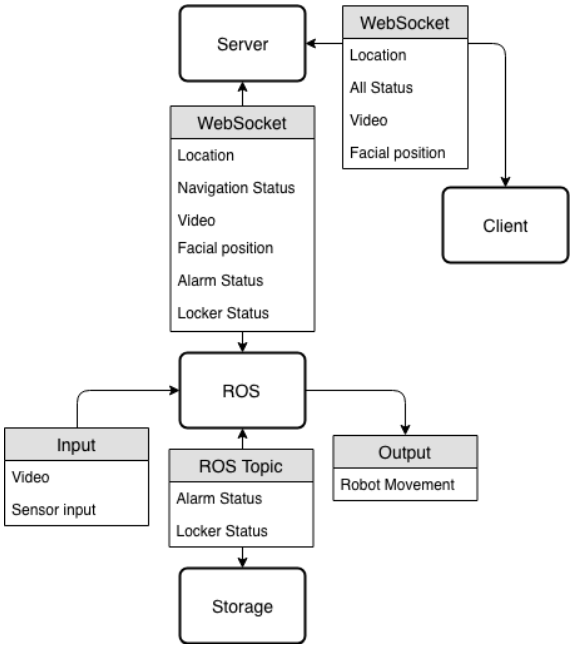

Figure 2: System Architecture 2

3.2. System Architecture

Two variations of system architectures are proposed initially, which are shown in Figure 1 and Figure 2 respectively. The two architectures are mostly similar except the communication between the **Storage** and the **Client**. Architecture 1 suggests that the communication is done via **WebSocket** with the **Server**, whereas in Architecture 2 the communication is done indirectly via **ROS Topic** and **Client**.

This has implications for flexibility, scalability, security and performance on the system. Architecture 1 is more flexible as it eliminates the dependency on ROS. This allows the storage to be replaced by entirely new hardware/software without disrupting the existing packages in ROS. This also offloads the computing power needed to run the Storage system from the machine running ROS onto another machine (such as Raspberry Pi or Arduino), which means better performance for both systems. Nevertheless, Architecture 1 could bring weaker security as communication with the server is done via a wireless connection, which by nature is insecure, exposing the Storage system to the risk of hacking or Cyber attack. On the other hand, Architecture 2 allows the communication done via ROS Topic which can be configured through a wired connection. However, since ROS exchange status of the storage with the server via a wireless connection, this also exposes the storage system to a certain extent of security threat. Based on the metrics evaluated, Architecture 1 is favoured.

3.3. Hardware

3.3.1 Robot

Two robots were considered in this project - the Peoplebot and the Pioneers P3AT. In the context of the project plan where the robot is used for delivery, safety and security have a greater weighting in the decision. Although the Peoplebot is taller and easier for the person to store or retrieve an item, it is less stable because of its height and can easily topple, especially when it comes to unexpected collisions or vandalism. Using a more stable but shorter robot such as P3AT will be more scalable when features such as moving through stairs are to be implemented. Therefore, P3AT is chosen for its better safety and scalability.

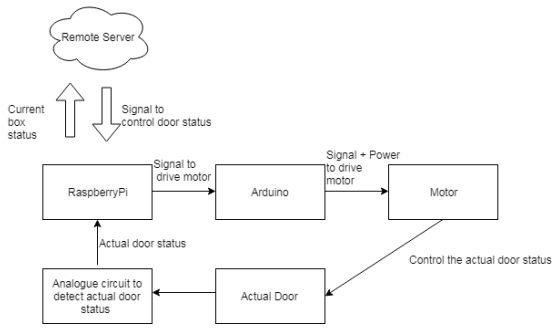

Figure 3: whole storage mechanism

Figure 4: Design of the jigsaw shape connection

3.3.2 Storage

Figure 3 shows the entire storage mechanism. First, the server will send an open signal to the client in the Raspberry Pi when the sender sets the status of the box to be open. Raspberry Pi will then send a signal to the Arduino which will trigger the motor to unlock the door. After 5s of waiting, door self-locking circuit will start to detect the actual door status. When the door is closed by the receiver, the Raspberry Pi will trigger the Arduino and the motor to lock the door. At the same time, the Raspberry Pi will also send the current box status back to the server for the server to decide the next move.

Material for storage - For prototyping purpose, materials such as cardboard and wood are considered to build the container. Cardboard, while being very cheap, is not strong nor safe enough for the purpose of the delivery robot. For wood comparison, Laserply Birch Plywood (Laserply in short) will be harder and has a natural colour of wood but it is harder to process (take more time and effort to cut). On the other hand, MDF is softer and it gives an even colour when painted, however, if it is not finished it could absorb liquid and swell easily. Due to the consideration of strength for better security, Laserply is chosen. Laser cutting technique was being used to create a clean and precise cutting for the box with the aid of AutoCAD software. A jigsaw shape connection at the side of each wall of the storage shown in Figure 4 is used to create a stronger connection between the walls.

Locking Mechanism - For the self-locking mechanism, a linear actuator was considered to push the storage lid outwards, however this mechanism only opens the storage and does not lock. Therefore, a simple version of the self-locking mechanism is designed where an electrical contact is built at the contact between the lid and the lock of the door so that the system can detect when the lid is closed and trigger the motor to activate the lock. Figure 5 shows circuit design of the self-locking mechanism .

Processing Unit - The processing unit such as Arduino with ESP8266 wifi module and Raspberry Pi are considered. Raspberry Pi has an Operating System (OS) integrated into it, which

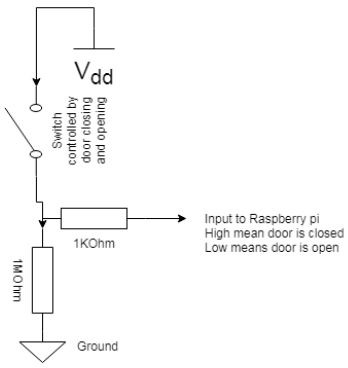

Figure 5: Simple circuit design for self-locking mechanism

makes it more flexible, performant and easy to use. However, as the processing power required for the locking mechanism is not very high, with more configuration the Arduino could be more power efficient as it does not need to run an OS and other unnecessary packages. Therefore, the use of Arduino was initially attempted, however it was experimentally discovered that it is difficult to implement the desired storage system solely using Arduino despite the benefit of power efficiency given the time constraints (which is discussed in detail in Section 4.8).

Motor - Two types of motor are considered, DC motor and stepper motor.The main factor in determining which motor to use will be the power rating as the motor will be powered by a portable power. Despite the stepper motor being more precise and accurate, it draws a higher power and is a lot more expensive than the DC motor. Therefore, for the prototyping phase, the DC motor is chosen as it fits the purpose.

3.4. Software

3.4.1 Client

To implement the user interface, web app and mobile app are considered. It is decided that the web app is more suitable compared to a mobile app as a mobile app is more expensive to develop. Also, it is inflexible and unscalable compared to the web app in terms of availability of third-party libraries. For the web app, frameworks such as React, Angular and Vue.js are evaluated. They are equally performant and configurable. The React framework is chosen for its flexibility and rich choices of middleware, familiarity with the framework and the rich online resources available. Also, the React framework is more easily migrated to the native mobile application with the use of ReactNative framework. Redux, redux-thunk middleware is used to ensure the state of the web app is managed properly. Also, TypeScript is configured as the language to write the web app to ensure type sanity and readability.

3.4.2 Server and Cloud Service Provider

For the server framework, Node.js and Python Flask. Both frameworks use dynamically typed languages, JavaScript and Python respectively, which allow for faster prototyping. Python Flask is arguably more scalable as it could be made multi-threaded whereas Node.js is by design single threaded. Nevertheless, Node.js is event-driven and it supports multiple concur-

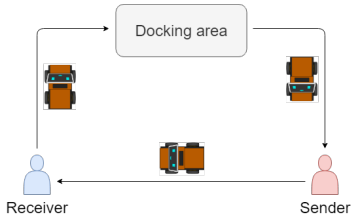

Figure 6: Design of the work-flow of delivery robot

rent requests and operations via asynchronous calls and a non-blocking I/O, which maximises the usage of a single CPU and computer memory, making servers faster and more productive. Since the server does not need to handle CPU-intensive operations but event-driven operation, and it is convenient to write the server and client-side web app in similar language (TypeScript and JavaScript), Node.js is chosen over Python Flask.

Several cloud service providers are considered. One of the famous backends is Google Firebase for its integration with database (up to 1GB), cloud storage (up to 5GB), free web hosting and various cloud computing functionalities for free. However, due to its closed backend environment, it is very difficult if not impossible to set up a WebSocket. Therefore, it is needed to use other open cloud computing services, such as Amazon Web Services (AWS) and Google Cloud. AWS is preferred for its AWS Educate plan, which allows the setup of cloud services, hosting and domain name for free.

3.5. Planned Work-flow

The two main users of the delivery robot are the receiver and the sender. The sender will interact with the robot through a web app by inputting the receiver's location as the destination. Once the destination is received, the robot will move to the sender from the docking area, which allows the sender to place his item into the storage. Then, the robot will move to the receiver's location. The sender is able to receive the delivery status, location, and facial tracking from the robot in real time. When the robot arrives at the receiver's location, the sender can verify the receiver through the real time video stream and decide whether to unlock or to keep the door closed. After the item is retrieved, the robot will return to the docking area, ready for the next delivery. The designed work-flow of the delivery robot is visualised in Figure 6.

4. Experimental Setup & Methodology

4.1. Software development environment

Git is used as the version control for software development. The git repository is hosted on Github. The main branch is the master branch, and different features are developed in different branches. Once the feature is working, it is merged with the master branch. In summary, the master branch acts as the production branch, it makes sure the packages in the master branch are working and ready to be deployed.

There are five main directories in the repository, namely the client, server, mock_ROS, ROS and raspberry_client. The client, server and ROS directories are arguably self-explanatory, in which they contain the code for the client, server and ROS respectively. The directory mock_ROS contains the ROS packages that has the desired functions without executing the ROS framework, such as connection to the WebSocket, facial recognition, and locking mechanism of the lock running on the Raspberry Pi. The directory raspberry_client contains the code responsible for the auto-locking mechanism.

The software is developed on the local machine, therefore it is essential to ensure that the dependencies on the local machine are present in the production machine, i.e running on the same environment. Therefore, packages manager such as Node Package Manager (npm) and Node Version Manager (nvm) are used in local and production machine to ensure equal node versions.

The production machines consist of the Elastic Cloud Compute (EC2) VMs hosted on AWS and the laptop provided. The client in production will run on port 8080 whereas the server on port 9000. Therefore, the EC2 firewall has to be configured to whitelist the aforementioned ports. Ubuntu server 18.04 is installed on the EC2 instances whereas Ubuntu 16.04 is installed on the laptop provided, which is the version supported by Kinect-ROS.

4.2. Face Tracking

Face tracking is done through a laptop's built-in webcam. The laptop screen will display a face that moves its eyes based on the human's position in front of the robot. Different face tracking algorithms were used to find the most suitable one for the robot system. The goal of face tracking is to publish topics containing information about the face's coordinates. The web server will display a face on a browser that will move its eyes based on the face's position captured by the webcam. Important factors for a suitable face tracking algorithm are: **High frame rate** so that robot will be able to respond without much delay. The webcam must track only **one face at a time** when there are more than one human detected. Below shows the different tracking methods that were tested.

HOG descriptors[9]: It is most suitable when it is used to track humans that appear small in the camera frame. During testing, HOG descriptors tracked a face that was far away from the camera instead of one which was closer. If the person comes closer to the webcam it will fail to track.

Medianflow tracker[10]: The tracker tracks the face backwards and forwards in time, and the disparity between these two directions are measured. However, when testing large motion of the face, the tracker fails. The medianflow tracker is also heavily affected by occlusion and will not recover from it.

Haar-casscade Classifiers[8]: This is a feature-based cascade classifier that detects objects using a cascade of simple features. OpenCV contains different types of pre-trained classifiers for different features such as face, eyes, upper body and even a cat's face. By loading the given XML classifiers, the classifier is able to detect new images that have the same features as the pre-trained ones. This method shows promising results and was able to detect multiple faces accurately.

After trying out different methods of detection and tracking, Haar-Casscade Classifiers is chosen. In order to detect and track only one face in a frame, the face tracking algorithm is written in such a way that it only tracks the face with the largest area in the frame as shown in Figure 7.

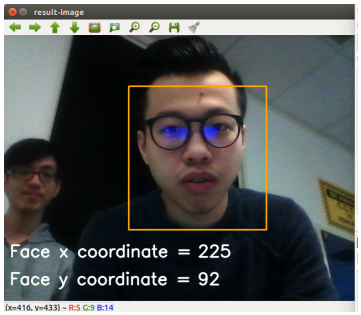

Figure 7: Face detection and tracking using Haar-Casscade Classifiers

4.3. Obstacle Avoidance

4.3.1 Sonar

Sonar is implemented as an alternative for obstacle avoidance. The robot is self-equipped with 16 sonar sensors, 8 at the front and 8 at the back. Initially, sonar sensors were planned to be used in the short range obstacle avoidance to cover up the blind spots (z axis) of laser scanner.

An algorithm comparing the range of obstacles is used as input to the auto-correction mechanism for the navigation command from the nav2d package. The experiment was carried out by putting obstacles at different angles in front of the robot. Results demonstrated that the sonar sensor has limitations such as inaccuracy, slow response time and difficulty in reading the reflection signal from the soft, curved, thin and tiny objects.

4.3.2 Microsoft Kinect

Freenect and OpenCV libraries will be used in the obstacle avoidance algorithm via the Kinect. Some experiments are carried out by using humans as the subject to determine accuracy and sensitivity of the Kinect. They will approach the Kinect from different directions, where different warnings like "collision alert", "move left" or "move right" shall be generated.

The Kinect has an advantage as its view is almost the same as the human eye view. However, its limitation is that it is not able to detect obstacles in a close range approximately shorter than 0.8m. Kinect is also very sensitive to the surrounding noise. Therefore some filters are needed to filter out all those noises.

4.3.3 Laser

The laser scanner used is the RPLidar A1M8. It is a 360° laser scanner with scan frequency of 1-10 Hz and angular resolution of 1° (when scanning at 5.5 Hz). It has a wide sensing range up to 12 metres. The laser scanner is used as the primary source of input data for the robot to visualise its surroundings.

4.3.4 Discussion

The laser scanner is chosen to be the standalone sensor unlike the Kinect which cannot detect objects that are too close. Moreover, the Sonar is unable to detect obstacles with smooth surfaces; non-existing objects are detected due to reflections produced by the sonar; the environment can cause fluctuations in the travel time of the echo[1].

Aspect	Gmapping	hector SLAM
Type of laser scanner needed	Optimises for a long-range laser scanner, like SICK LMS or PLS scanner	Suitable for short range laser scanner (RPLIDAR A1 has a range of 12m)
Odometry data	Mobile robot has to provide odometry data	No odometry data is required from the robot
Computational resources required	Require higher computational resources	Low computational resources required
Condition of environment	Performs better in ambiguous environments, e.g. a large space or a long hallway without significant features	Does not perform well in an area with great laser ambiguity

Table 2: Comparison between Gmapping and hector SLAM

4.4. Mapping

The mapping of the robot's surrounding is done via RPLidar and implementing hector SLAM (simultaneous localisation and mapping). The map server package is used to save the map for robot localisation and path planning. Several mapping and localisation methods were considered, including GPSc& mobile networks, Gmapping and hector SLAM. Localisation via GPS & mobile networks were not accurate enough, especially since the robot is operated inside a building. Thus, Gmapping and hector SLAM were the main methods considered when implementing SLAM. Table 2 shows the comparisons between Gmapping and hector SLAM.

Hector SLAM works well with the RPLIDAR. Requiring no odometry data and low computational resources are the advantages that make the hector SLAM an attractive choice.

However, as the laser scanner used is short range, the robot cannot scan further than a room. For instance, a long corridor would be difficult during mapping as it has to navigate along the corridor to map it. When mapping, hector SLAM suffers when the robot is moving over an uneven surface or trying to overcome an obstacle. This could be easily solved by mounting the laser scanner on top of the robot firmly and securely. The robot cannot make a sharp turn in a short time when mapping as doing so would adversely affect the quality of the map which cannot be recovered. Besides, using the hector SLAM in an area with no significant features could cause confusion problem to mapping. Thus, the map is built by driving the robot around and scanning the surroundings.

4.5. Localisation

Monte Carlo localisation approach is used for the robot to localise itself on a known map. The algorithm uses a particle filter to estimate the robot's position on a given map based on its motion and sensor data. It estimates the robot's pose by tracking the propagation of particles following the robot's motion that was initially distributed on the map. Upon receiving new sensor

data from the laser scanner, each particle will evaluate its accuracy and converge to a single cluster near the true pose of the robot if localisation is successful.

In order to ensure fast and accurate localisation of the robot, multiple tests were performed with varying number of particles (Table 3). To obtain accurate localisation in a short time span, the minimum and maximum particles used were 5000 and 10000.

Min particles	Max particles	Observation
5000	8000	Very fast localisation but not accurate
5000	10000	Fairly fast localisation and more accurate
5000	20000	Very accurate but takes a long time to localise

Table 3: Effects of number of particles on localisation

The number of features in the map is also an important factor for accurate localisation. Two scenarios are tested for localisation: a corridor with minimum or no features, and the same corridor but with some boxes arranged on each side. Figure 8 shows how localisation will look like in RViz. When the corridor is featureless, sensor readings will look the same and the particles will not be able to re-sample itself to more accurate particles. The robot will think that it does not really move at all when moving along a featureless tunnel/corridor. Thus, the robot is not able to localise itself properly. When adding features such as boxes along the corridor the robot is able to redistribute the particles by its sensor readings and the known map, resulting in a better localisation.

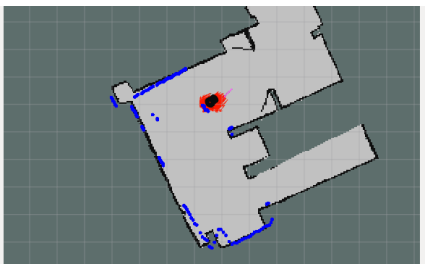

Figure 8: Localisation shown on RViz, red arrows represent particles and blue colour represents laser scans

4.6. Navigation

With the main robot functionality to deliver, the process of ascertaining the position, planning and following a route are vital to the objectives. Also, as the robot system services are designed to operate within a predefined area, any interfacing with the mapping, localisation and obstacle avoidance must be suitably established. Navigation within the same floor of a building is adopted to control the mobile robot within a planar environment. There are presently two packages that provide the essential modules for 2D-navigation, namely the navigation stack[4] and nav2d[3].

4.6.1 Navigation stack

The navigation stack requires the robot set-up to be in a particular configuration, in order to run and fully utilise the package. A number of new peripheral components are created to interface the specific P3-AT robot platform to the package. This includes the transform configuration, sensor information, odometry status and map server package. Each individual node is responsible for regularly reading/measuring relevant physical status and in turns publishing them to the target topics. The inclusive launch files are written to make sure that all the nodes are instantiated in a consequential order.

During experimentation, there was a problem with how the robot localises itself within the global map previously created for the surrounding. Though with manual assistance to provide an initial localising pose, it would not localise and any planning instructions were failed as a consequence forcing the robot into a recovery mode. No matter how many trails and resolutions were carried out, with the complexity and lack of framework supports, the issue remained persistent and as a result the nav2d package was explored as an alternative.

4.6.2 Nav2d

The nav2d libraries provide a straightforward mobile robot control in a planar environment, with an emphasis on a purely reactive obstacle avoidance and simple path planner. Setting up the package is similar to that of the navigation stack configuration, but as the horizontal laser range finder is used as the sole primary input, only the transform configuration, laser data, map server and p2os driver are adequate in this case. In addition, RViz is being used as a 3D visualiser for displaying sensor data and state information from ROS with regards to the real time navigation. One useful value of this is the ability to run a simulation of the P3-AT robot with a laser scan data in simulated environment. This allows preliminary testing on the package functionality which it has proved to be competent with the objective.

At implementation, as the robot is localised, the navigation goal command can be given through RViz or by publishing the goal topic from the server. The latter requires a suitable map translation, as the real world location command from the server has to interface with the corresponding map coordinates and correct orientations, all of which can be derived from the mathematical scaling method. As the goal command is received by the package, the optimised plan is initiated and the robot will start to navigate. After fine tuning the configurations, the robot is successful to perform realtime obstacle avoidance with great responsiveness and precision, where the new suitable path will override and lead the navigation as obstacles are present.

To process the laser data effectively, two preprocessing steps were undertaken. Firstly, a static transform was applied to represent the translation between the robot base frame and the frame of the laser. This is set to a $0.3m$ static translation in the Z axis, which is the measured distance between the centre point of the robot to the centre point of the laser.

Next, the laser data was filtered such that the data only included data from 0 to π - i.e only the front of the laser. This is because the locker would be behind the laser, and the robot

would detect it as a wall directly behind it. This would interfere with the navigation aspect of the problem - in particular, localisation, obstacle avoidance and path planning as the robot will incorrectly assume that there is a wall behind it and act accordingly.

4.6.3 Costmap Parameters

The costmap is a data structure that represents places that are safe for the robot to be in a grid of cells, or simply where the robot will be able to move to. The configuration of key parameters can determine if a robot is able to find a valid path to the destination. The main parameters that were tested are the obstacle range and the inflation radius.

The obstacle range is the maximum range in which the robot will insert obstacles into the costmap, essentially ignoring obstacles further than the cutoff range. After experimentation, setting this parameter to $4m$ gave the best result for navigation. The inflation radius is the radius to which the cost scaling factor or inflation is applied. This factor is used to make objects closer to the robot have more cost and therefore the robot will try to avoid it. From testing,the best value for navigation was $0.55m$.

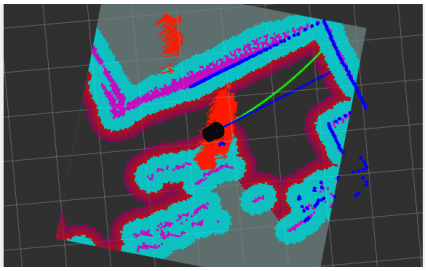

Figure 9: Costmap shown on rviz

4.6.4 Discussion

One interesting issue was solving the problem of navigating through doorways. Initially, the robot would always detect a narrow doorway (e.g. a door) as an obstacle. Upon viewing the cost maps it was clear that the costs associated with both ends of the door were overlapping the empty space in the middle, resulting in the robot detecting it as an obstacle. This was fixed by fine-tuning the parameters for the navigation stack, in particular, the inflation radius.

The plan was to have a map visualised on the web - this was achieved through the Twitch streaming platform. However, once the laptop started streaming, the robot could no longer localise itself. This is presumably due to the amount of processing required for streaming. A temporary fix was to localise first and then stream. A better way would be to either improve the hardware on the robot(laptop) and optimise settings for the stream. Furthermore, with the current laptop, the battery lasts around 20 minutes with everything running. Ideally, an additional power bank could be connected or the laptop wired up to the p3at robot

4.7. Server and Client on the Cloud

Client - The frontend web app is developed with the React framework. It is configured with web pack to be written in TypeScript with the library awesome-typescript-loader. Node Package Manager (npm) is used to record and maintain the libraries needed to run the client. Using npm middleware such as Redux, Redux-thunk, socket.io-client and material-ui are installed. Redux-middleware is used to manage the state of the web app, which allows for better app stability and debugging speed. To establish a WebSocket, the Socket.io-client library is used. The Material-ui library is used such that the components of the web app are better designed and animated, which allow for a better user experience. The design of the client interface is shown in Figure 10. The client interface is used to set the pickup location for the robot to navigate to.

Specification	Stepper Motor	DC motor
Torque (Nm)	2.2	1.0
Power Rating (W)	3.724	1.23
Price (£)	28.75	0.44

Table 4: Decision metrics for motor used

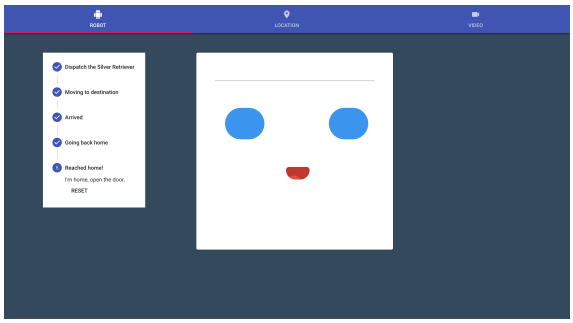

Figure 10: Design of the client user interface

Server - Nodejs is used as the framework for backend development. Plugins such as express.js, cors are used to facilitate the development. To make sure the local development machine and production machine run the server as expected, nvm is used to manage the node version on both local and remote machine. Socket.io is used to quickly establish a WebSocket on the server, which is open to both storage, client and ROS.

Cloud - Both client and server are first developed and tested locally. After that, Amazon Web Service (AWS) Educate account is set up, and two separate Elastic Cloud Compute (EC2) VMs are instantiated with Ubuntu server 18.04. When the client and server are ready for production, they are served on the two EC2 instances separately. They are run permanently on the cloud using command nohup and node, which is configured and managed by nvm to run the same version as the local machine.

4.8. Storage

Material for storage: The strength of cardboard, MDF and Laserply is investigated by measuring how long it takes for the electric drill to drill through each material. Cardboard can be drilled through less than 1s, 6mm MDF in 20 seconds (s) and 6mm in Laserply in 39s. The stronger material, Laserply, is chosen as the building material for the storage.

Locking Mechanism: Some researches had been done in determining which motor is the best for the use in the storage box. 3V DC motor is chosen in the end. Table 4 shows the data obtained from the datasheet of the motors. Stepper motor requires

an input current of 1.68A while DC motor need about 0.4A.

Processing Unit: For server communication, Arduino using GET HTTP takes about 10s (to established new connection) whereas the Raspberry Pi takes less than 1s. On the other hand, the interval to recharge the 10000 mAh power bank used to power the Raspberry Pi is less than one day whereas for Arduino it is about one week. The Arduino and Raspberry Pi give a maximum output current of about 40mA and 16mA for each pin respectively. Even though both of them have not enough current to drive the DC motor(400mA), during experimentation only the Arduino is able to drive the motor. In the future, different approaches like using L293 motor driver will be considered to get rid of these output current issues. Based on the experiment, the Arduino will be slow in communicating with the server while the Raspberry Pi does not supply enough power to drive the motor. Therefore, they are combined together to create the final storage mechanism. The Raspberry Pi will have the role in communicating with the server and the Arduino will be used in supplying power to drive the motor. The communication between them will be established using jumper wires.

5. Integration

All the components mentioned above are integrated together and the robot is tested in the EEE department on the 5th floor. Some boxes were added along the corridor to add features for mapping. Following the designed work flow, the communication between humans and the robot is made through the specially created web interface. The robot is first tested with no moving obstacles to test the navigation aspect. Afterwards, group members are positioned randomly along the map and the robot avoided them while staying on its path to the given goal. The robot will ignore strangers walking along the map by steering itself away to avoid collision. Testings show that the robot has a success rate of 90% where the only scenario that the robot fails is when there are too many obstacles. This happens when there are too many people crowding around the robot or when the robot can't pass through a small doorway due to overlapping in the costmap. As a result, the robot stays where it is because it can't find a path to its goal. The web interface is tested on a public user who utilised all the features in under 30 seconds, partly due to the UI/UX design. The robot attention demand (RAD) will be highly influenced by the time interval before the power source (power bank) of Raspberry pi needs to be recharged. Service time (t_s) is the time used to charge the power bank and neglect time (t_N) is the time for the power bank to discharge fully. t_s is about 10 hours and t_N is about 24 hours. Therefore, RAD is about 29.4%. The whole system will take around 1 minutes for a delivery trip of 5m. 20s for sender to place the item, 20s for receiver to take out the item and 20s for back and forth journey of the robot assuming starting point is the docking station.

6. Future Works

Thus far, the robot is successful aiding a person to delivery goods to another person while navigating around corners and obstacles. There are still areas that can be improved in terms of security and authentication. When the robot is moving autonomously without human supervision, it is crucial to ensure that the robot arrives safely at its destination and any cases of theft along the way must be eliminated. In order to do that, an alarm system can be implemented on the robot to alert the user by sending a notification to the user's mobile device. An ultrasonic sensor or gyroscope can be integrated so that the user can be alerted when the robot is knocked over or lifted up (shows occurrence of theft). Secondly, some sort of authentication will be implemented where the receiver will be given a bar-code and the robot will scan it and unlocks the storage locker when it is the right bar-code. Finally, information containing location of the robot on the map will be sent and visualised on the user's interface to allow the user to track the robot better.

7. Conclusion

The project aims to explore the delivery automation opportunity, capable of exploiting the intelligent human interaction within the present-day robotic systems. The web server is considered and chosen to serve as a common two-way human-robot interface, allowing each party to recognise each other's actions. While a human can observe and physically interact with the machine, the robot itself utilises components such as a camera and laser scan to perceive a human presence. As the user's need to deliver a package is perceived by the robot systems, the user model will be constructed for the robot to determine the most appropriate way of achieving the task with the optimal planner. The locking mechanism and validation technique provide a method to control the system externality. Various issues that might arise in the robot actions are taken into account to ensure the task completion. With such models, the project demonstrates the basis that a robot is capable of assisting human duties and react to accommodate the changing needs.

References

[1] B. Bright, M. Curl, and M. Elswick. Sonar and the uses in mobile robots. 2011. http://www.eng.auburn.edu/ troppel/courses/5530

[2] M. Butcher. Skype co-founders launch the starship 'ground drone' for deliveries. 2015. https://techcrunch.com/2015/11/05/skype-co-founders-launch-the-starship-ground-drone-for-deliveries/.

[3] S. Kasperski. nav2d. 2017. http://wiki.ros.org/nav2d.

[4] E. Marder-Eppstein. navigation. 2017. http://wiki.ros.org/navigation.

[5] K. Morley. One in every five pounds spent with uk retailers is now online, figures show. 2018. https://www.telegraph.co.uk/news/2018/08/16/one-every-five-pounds-spent-uk-retailers-now-online-figures/.

[6] Nanalyse. 8 delivery robot startups for last mile delivery. 2018. https://www.nanalyze.com/2018/04/8-delivery-robot-startups-last-mile-delivery/.

[7] R. B. Review. Savioke relay autonomous delivery robot. 2015. https://www.roboticsbusinessreview.com/consumer/relay/.

[8] R.Raja. Face detection using opencv and python: A beginner's guide. 2017. https://www.superdatascience.com/opencv-face-detection/.

[9] S.Mallick. Histogram of oriented gradients. 2016. https://www.learnopencv.com/histogram-of-oriented-gradients/.

[10] S.Mallick. Object tracking using opencv(c++/python). 2017. https://www.learnopencv.com/object-tracking-using-opencv-cpp-python/.