# Mail Delivery Robot in Carleton Tunnels

**Project Proposal**

Denis Cengu 101077902
Douglas Lytle 101147977
Cholen Premjeyanth 101184247

Supervisor: Dr. Babak Esfandiari

Department of Systems and Computer Engineering
Faculty of Engineering
Carleton University

October 28th, 2024

# Table of Contents

## List of Tables



## List of Figures

# 1 Objectives

## 1.1 Project Objectives

By Douglas Lytle

The primary objectives identified up to now for this project remain largely the same as they were in previous iterations. The robot should be developed such that it is able to autonomously navigate from an arbitrary location in the Carleton tunnels to any of its destination docking stations, while avoiding obstacles. The robot should be able to carry mail and ensure it is not lost in transit, and must have a battery capacity large enough to complete deliveries. This year, the robot should be able to navigate between at least three different locations in the Carleton Tunnels. The web app should also continue to be developed and will remain the primary method for users to interact with the robot.

Early in the development of the project, an additional focus should be placed on rectifying the issues outlined in last year's report's sections titled *Documented Issues*, and *Future Work*. These include problems preventing the Create 2 from properly docking, undocking, and stopping. They also include improvements to intersection detection, and the state machine used for navigation should be greatly simplified or potentially removed. Also, more powerful beacons may be necessary as their signal strength seems inconsistent.

In order to facilitate these goals, another objective which should be handled early in development is the implementation of simulations using Gazebo, which is a much more sophisticated way to test the system compared to what was used last year.

## 1.2 Project Description

By Denis Cengu

This will be the fourth year of this project. The main goal is to create an autonomous robot that can deliver mail between different buildings at Carleton University through its tunnels. The project will include a web application that will allow users to easily place their orders and manage deliveries.

Last year's team made improvements to the robot's movement abilities, refining the wall following, turning and intersection navigation. The primary focus for this year's team will be to ensure more consistent and reliable behavior. This may include the introduction of new sensor systems and reworking or altogether scrapping the state machine approach currently used for navigation as well as making the system more scalable and adaptable.

The key areas for this year's team are to ensure the primary focus are improvements to docking and undocking, which the robot currently struggles due to reliance on the short range IR sensors, intersection detection, a potential redesign or altogether removal of the state machine to simplify the navigation, further web application development, implementation of Gazebo simulations as well as further improvements to the security of the mailbox.

### 1.3 Requirements

By Douglas Lytle and Cholen Premjeyanth

#### 1.3.1 Functional Requirements

- **R1**: The robot shall be able to autonomously navigate from any starting location within the Carleton tunnels, to one of the designated destination stations.
- **R2**: The robot shall be able to detect and handle collisions with obstacles.
- **R3**: The robot shall be able to detect and identify intersections within the tunnels.
- **R4**: The robot shall ensure mail is not lost during delivery.
- **R5**: The robot shall be able to notify the system of its current status, including mail delivery progress, battery levels, and docking status.
- **R6**: The robot shall be able to receive signals from Bluetooth beacons placed in the tunnels.
- **R7**: The robot shall be able to dock automatically upon reaching its destination or when the battery is low.
- **R8**: The system shall allow users to register an account using the web application.
- **R9**: The system shall allow users to make delivery requests using the web application.
- **R10**: The system shall be able to notify the robot of new delivery requests.
- **R11**: The system shall be able to provide detailed logs of all robot activities.
- **R12**: The web application shall be able to display the approximate location of the robot.

#### 1.3.2 Non-Functional Requirements

- **R13**: The system shall ensure only administrators have access to administration features.
- **R14**: The robot shall maintain a speed of at least 0.15m/s while navigating through the tunnels.
- **R15**: The system shall be tested thoroughly to ensure reliability of the system.
- **R16**: The robot shall maintain enough of its battery level to reach the nearest docking station at all times.
- **R17**: The web application shall be available to any registered user connected to Carleton University's Wi-Fi.
- **R18**: The project shall remain within the specified budget.
- **R19**: The project timeline and deadlines shall be followed and met.

### 1.4 Use Cases

By Cholen Premjeyanth

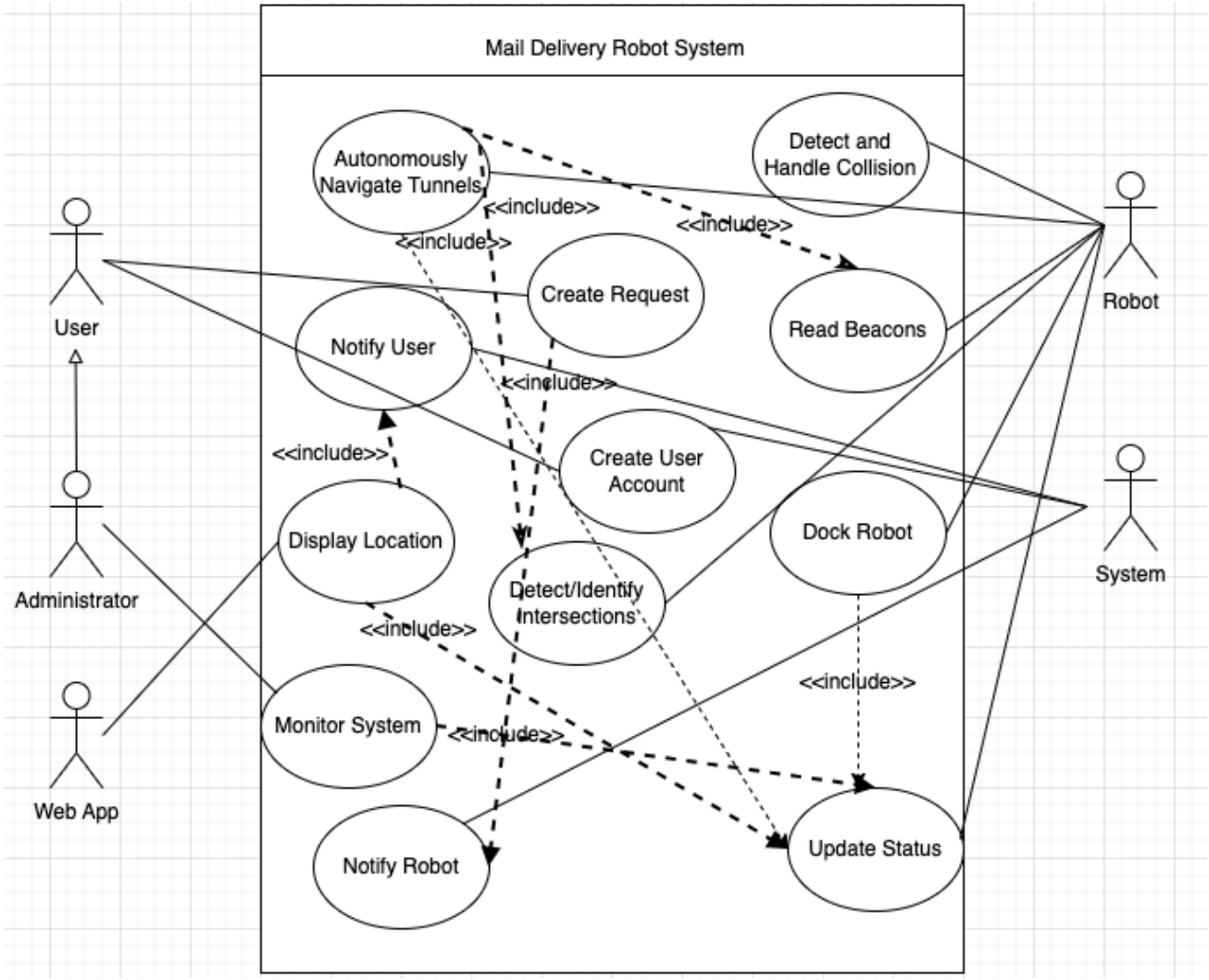


Figure 1: The use case diagram for the Mail Delivery Robot System.

#### 1.4.1 Individual Use Cases

By Cholen Premjeyanth

Table 1: Autonomously Navigate Tunnels use case

| | |
|---|---|
| **Use Case Name** | Autonomously Navigate Tunnels |
| **Brief Description** | The robot is able to navigate autonomously from one location to another within the Carleton tunnels |
| **Primary Actor** | Robot |

| | |
|---|---|
| **Secondary Actor** | System |
| **Precondition** | System is active and running, and the robot is turned on, with enough battery, and the beacons activated, and the robot has received a request. |
| **Postcondition** | Robot successfully navigates to the specified destination and updates its location status |
| **Basic Flow** | 1. Robot initializes a path to the destination using map data and available beacons.<br>2. As the robot moves, it uses sensor data to detect and navigate around obstacles.<br>3. The robot continually seeks and moves towards the nearest beacon in its path, updating its location periodically.<br>4. If a beacon is missed at a certain distance (e.g., 10m threshold (taken from testing)), the robot performs a sensor-based recalibration to confirm or correct its position.<br>5. The robot detects intersections, using sensor and beacon signals to make routing decisions.<br>6. Robot reaches the designated station and updates the system with its current location and status. |
| **Alternate Flows** | a) If the path is blocked, the robot recalculates the route and notifies the system.<br>b) If robot fails to detect the next beacon, it will recalibrate positions through other sensors. |

Table 2: Detect and Handle Collision use case

| | |
|---|---|
| **Use Case Name** | Detect and Handle Collision |
| **Brief Description** | The robot is able to detect any collisions (such as walls, humans, carts, etc.), and be able to reorient itself. |
| **Primary Actor** | Robot |
| **Secondary Actor** | N/A |
| **Precondition** | Robot is in motion within the tunnels |
| **Postcondition** | Robot successfully detects and/or avoids |

| | |
|---|---|
| | obstacles while being able to reorient itself onto its path. |
| **Basic Flow** | 1. Robot encounters an obstacle.<br>2. Robot detects the obstacle using sensors.<br>3. Robot reroutes to avoid the obstacle. |
| **Alternate Flows** | N/A |

Table 3: Read Beacons use case

| | |
|---|---|
| **Use Case Name** | Read Beacons |
| **Brief Description** | The robot is able to detect beacon signals traveling through the tunnels. |
| **Primary Actor** | Robot |
| **Secondary Actor** | N/A |
| **Precondition** | System is active and running, and the robot is turned on within the tunnels. |
| **Postcondition** | Robot successfully provides the system with its beacon data. |
| **Basic Flow** | 1. Robot detects the beacon signal and determines from which intersection it is coming from.<br>2. Robot logs the information within the system. |
| **Alternate Flows** | N/A |

Table 4: Detect/Identify Intersections use case

| | |
|---|---|
| **Use Case Name** | Detect/Identify Intersections |
| **Brief Description** | Robot is able to detect intersections and navigate its way to the destination |
| **Primary Actor** | Robot |
| **Secondary Actor** | N/A |
| **Precondition** | System is active and running, and the robot is turned on within the tunnels |
| **Postcondition** | Robot accurately identifies intersections and |

| | |
|---|---|
| | makes routing decisions |
| **Basic Flow** | 1. Robot approaches an intersection.<br>2. Robot identifies the intersection using sensors.<br>3. Robot makes a decision to turn or continue straight based on the destination. |
| **Alternate Flows** | N/A |

Table 5: Dock Robot use case

| | |
|---|---|
| **Use Case Name** | Dock Robot |
| **Brief Description** | Robot should be able to dock itself after the completion of a delivery, or when the battery is low |
| **Primary Actor** | Robot |
| **Secondary Actor** | N/A |
| **Precondition** | Robot has either completed a delivery or reached a low battery threshold |
| **Postcondition** | Robot successfully docks to recharge or reset, and its status is updated in the system |
| **Basic Flow** | 1. Robot detects low battery or task completion.<br>2. Robot calculates and navigates an optimal path to the nearest docking station using map and sensor data.<br>3. The robot uses its sensors to align and connect accurately with the docking station.<br>4. The robot sends a status update to the system, indicating that it has reached the docking station and is recharging.<br>5. Robot initiates the charging process, and the system logs and displays the docking and battery recharge status. |
| **Alternate Flows** | a) If docking fails, the robot retries docking or sends an error update to the system. |

Table 6: Update Status use case

| **Use Case Name** | Update Status |
|---|---|
| **Brief Description** | Robot is able to send a status update to the system |
| **Primary Actor** | Robot |
| **Secondary Actor** | System |
| **Precondition** | Robot is operational and running |
| **Postcondition** | System displays the updated status |
| **Basic Flow** | 1. Robot sends the current status to the system.<br>2. System updates information such as delivery progress, battery levels, and docking status.<br>3. Status is accessible by users. |
| **Alternate Flows** | a) If there is no internet connection, then keep trying |

Table 7: Notify User use case

| **Use Case Name** | Notify User |
|---|---|
| **Brief Description** | User is notified of any updates to the robot, their delivery, or of the system. |
| **Primary Actor** | System |
| **Secondary Actor** | User, Robot |
| **Precondition** | User has an active account on the web application and robot is running |
| **Postcondition** | User receives delivery updates |
| **Basic Flow** | 1. System detects a status change in the robot.<br>2. System sends an update notification to the user. |
| **Alternate Flows** | a) If system fails to update user, continue trying until success |

Table 8: Create User Account use case

| | |
|---|---|
| **Use Case Name** | Create User Account |
| **Brief Description** | User has access to an account through which they can see information about the robot |
| **Primary Actor** | User |
| **Secondary Actor** | System |
| **Precondition** | User has access to the web application |
| **Postcondition** | User account is created successfully |
| **Basic Flow** | 1. User accesses the account registration page.<br>2. User fills out registration details.<br>3. System verifies details and creates the account. |
| **Alternate Flows** | a) If the system fails to create a user account, redirect until the account is successfully created. |

Table 9: Notify Robot use case

| | |
|---|---|
| **Use Case Name** | Notify Robot |
| **Brief Description** | System notifies the robot of any new updates or data. This includes new requests, navigation data, etc. |
| **Primary Actor** | System |
| **Secondary Actor** | Robot |
| **Precondition** | System is running and robot is operational |
| **Postcondition** | Robot receives updates from the system |
| **Basic Flow** | 1. System prepares a new update for the robot.<br>2. System sends the update to the robot. |
| **Alternate Flows** | a) If the system fails to send updates, terminate requests, and update users. |

Table 10: Display Location use case

| **Use Case Name** | Display Location |
|---|---|
| **Brief Description** | System updates and displays the location for the user to access |
| **Primary Actor** | Web App |
| **Secondary Actor** | User, Administrator |
| **Precondition** | Robot is active and tracking is enabled |
| **Postcondition** | Web application displays the approximate location of the robot |
| **Basic Flow** | 1. Robot updates its location.<br>2. System receives location data.<br>3. Web application displays the location on the user's screen. |
| **Alternate Flows** | N/A |

Table 11: Monitor System use case

| **Use Case Name** | Monitor System |
|---|---|
| **Brief Description** | System is monitored by the admin, who receives and tracks the data sent by the robot. |
| **Primary Actor** | Admin |
| **Secondary Actor** | Robot, System |
| **Precondition** | Admin has access to the system dashboard, and the system is operational |
| **Postcondition** | Admin successfully monitors the robot's activities and statuses |
| **Basic Flow** | 1. Admin logs into the system's dashboard.<br>2. Admin views real-time data on robot activities, battery levels, docking status, and obstacle encounters.<br>3. Admin can review detailed logs of all robot actions and system notifications.<br>4. If necessary, Admin can send commands or alerts to the robot (e.g., for maintenance or rerouting). |

| | |
|---|---|
| **Alternate Flows** | a) If the system encounters connectivity issues, Admin is notified of the communication error and can troubleshoot. |

Table 12: Create Delivery Request use case

| | |
|---|---|
| **Use Case Name** | Create Delivery Request |
| **Brief Description** | The request manager from the system, creates and validates request from the sending user |
| **Primary Actor** | System |
| **Secondary Actor** | Robot |
| **Precondition** | Robot is operational |
| **Postcondition** | Delivery request is successfully submitted to the system and queued for the robot |
| **Basic Flow** | 1. User logs in and accesses the delivery request form.<br>2. User provides delivery details, including pickup and drop-off locations.<br>3. User submits the request, which is then sent to the system.<br>4. System notifies the robot of the new delivery request.<br>5. User receives a confirmation of the request, and the system logs the delivery request in the system queue. |
| **Alternate Flows** | a) If there is missing or invalid data, the system prompts User to correct the information.<br>b) If the robot cannot accept new requests (e.g., due to maintenance), User receives a notification |

# 2 Background

### 2.1 Robot Operating System

By Denis Cengu

The Robot Operating System 2 (ROS 2) is a significant redesign of the original ROS, developed to meet the evolving demands of modern robots. Unlike the predecessor, ROS 2 includes critical features such as enhanced security, real-time capabilities, reliability in non-ideal environments and scalability. [1] Built on the Data Distribution Service (DDS) communication protocol, ROS 2 is compatible with diverse hardware platforms, including support for Ubuntu 20.04 that will be run on the Raspberry Pi 4. ROS 2 includes a unified set of core libraries that support multiple programming languages, such as C++, Python and Java. Moreover, ROS 2 is well suited for integration with the iRobot Create 3 and 2, through microcontroller communication, which will be used to enable effective multi-robot communication and autonomous navigation. ROS 2's modularity and interoperability make it a powerful tool for deployment of autonomous systems.

### 2.2 iRobot Create

By Denis Cengu

The iRobot Create 3 is the latest version of the iRobot Create series, offering several improvements over its predecessor. It comes with built-in ROS 2 support which means it will not require an external microcontroller to send instructions resulting in a straightforward integration process.[2] The Create 3 is equipped with six IR obstacle sensors to detect objects and follow walls. Additionally, the custom faceplate allows for easy mounting of additional boards. The Create 3 also offers a storage compartment for the Raspberry Pi and additional power options including a 14.4V/2A battery connection and a 5V/3A USB-C port for powering and connecting components. The built in ROS 2 provides a comprehensive set of commands for sensor data collection, wall following and positional navigation, making it compatible with code used for the Create 2 as well.

The iRobot Create 2 is an educational version of an autonomous iRobot vacuum that connects over a serial cable to a microcontroller to receive commands. [3] As mentioned in section 2.1, ROS is used once more to send commands from the Raspberry Pi to the serial port. The Raspberry Pi receives and processes all of the sensor data and then sends the robot commands to drive. It also supports the use of the "Virtual Wall", which acts like a Bluetooth beacon to create a boundary that the robot will not cross. The Create 2 requires more customization in comparison to the Create 3, including use of an external microcontroller and additional hardware to achieve the level of integration offered natively by the Create 3.

### 2.3 LiDAR Sensor
By Cholen Premjeyanth

The LiDAR sensor is integral to the robot's navigation by allowing it to use an accurate mapping of its surroundings. This sensor emits laser pulses that emit off objects and it measures the time it takes for each pulse to bounce back after it hits the objects. This allows for a highly accurate mapping of the robots surroundings. The data the robot receives through this sensor allows it to detect obstacles, navigate complex environments, and avoid collisions. In this project, the LiDAR sensor will be used to enable an accurate 360 degree view of the environment; which is the Carleton tunnels. This sensor can be paired with the Create 3's built in ROS 2 functionality, which will allow the data to be processed in real time, ensuring the robot adheres to safety and performance requirements. Due to the sensor having compatibility with ROS, it allows for it to be easily integrated.

The specific LiDAR sensor being used is the Slamtec RPLIDAR A1 which measures 8000 times per second with a range of 0.15-12m with a resolution of 0.5mm [7]. This sensor, which was recommended from last year has a scanning frequency of 10Hz and a range of up to 12m.

### 2.4 Spring Framework and Spring Python
By Cholen Premjeyanth

The Spring framework is a powerful Java framework that can be used for large softwares. It provides various infrastructure support such as integration, testing, and aspect oriented programming [8]. Using the Spring framework will allow for flexibility and reusability. For this project, the spring framework will be used to manage the backend of the system for the application that interacts with the robot. Integrating it with Thymeleaf will allow for real time updates. Spring Python, an extension of the Spring Framework, allows users to write in Python rather than Java. Spring Python will be instrumental in integrating the Python-based control scripts with the Spring-driven web application. The ability to use Spring's features in Python allows for interaction between the robot's Python-based ROS 2 systems and the Java-based web interface.

### 2.5 Thymeleaf
By Douglas Lytle

Thymeleaf is a modern java template engine, which allows HTML templates to be easily populated with data from an application. Thymeleaf is fully integrated with the Spring framework [4], allowing views to be constructed with the needed information from the model in a very straightforward manner. Thymeleaf will be used in conjunction with Spring in order to continue the development of the web app.

### 2.6 Microsoft Azure
By Douglas Lytle

Microsoft Azure is a cloud service which can be used to host web applications, and was used in past years for this project. The team will continue to use Microsoft Azure to deploy the web application for several reasons. Azure offers free plans, which will help to maintain the project budget for any hardware upgrades or replacements which may be needed. Also, Azure supports continuous deployment via GitHub [5], and experience will be gained in using Azure by the team during SYSC 4806, which will facilitate the deployment of the web app.

### 2.7 Gazebo
By Douglas Lytle

Gazebo is a simulation software developed by Open Robotics, the same company which develops ROS. Gazebo features an extensive set of development libraries made to facilitate simulations and testing for robotics projects [6]. ROS is fully integrated with Gazebo, so it is relatively simple to import the control systems for the robot and to construct an extremely accurate simulation. Gazebo even has libraries for the simulations of a wide variety of sensors which can accurately read information from the simulated environment and pass this information to the control system. The exact LiDAR sensor used in the project, as well as the iRobot models used, are represented in these libraries, which will allow for accurate testing. Using Gazebo for simulation will greatly improve the ability to test any modifications to the robot control system, as it will no longer be necessary to be physically present to test changes in many cases.

## 3 The Engineering Project

### 3.1 Health and Safety
By Cholen Premjeyanth

The testing, development, and future upgrades of the robot will continue in the Canal lab room (CB 5101) and the Carleton University tunnels, as in previous years. All experiments will be conducted in controlled environments, ensuring there are no variables that may obstruct the robot’s operation.

The safety of both the team and the public will be prioritized during testing. In areas where the robot may operate near people, clear communication and safety measures will be implemented to inform the surrounding individuals and minimize risks.

When handling the robot's hardware, strict safety protocols will be followed. This includes removing loose clothing and jewelry, maintaining a safe distance from electrical probes, and ensuring that all power sources are disconnected prior to working on any electronic components. These precautions will ensure a safe working environment and protect both the team and equipment.

### 3.2 Justification of Suitability for Degree Program

By Denis Cengu and Douglas Lytle

The team has two students in computer systems engineering and a student in software engineering. These programs are similar and both have major focuses on topics in software and computer engineering. Both programs cover topics related to embedded development, requirements engineering, web development, real-time systems and software development among others. The scope of the project encompasses all of the above.

The emphasis of the project revolves around a code base for the robot developed in Python using the Robot Operating System (ROS). The robot itself is driven through use of a Raspberry Pi 4. The Raspberry Pi is used in SYSC 3010 (Computer Systems Development Project) , where students in groups use multiple Pi's that communicate with one another to create an embedded system primarily with a Python code base as well. As well as that course for Computer Systems students, Software Engineering students will have also covered at a basic level these concepts in ECOR 1051 (Fundamentals of Engineering I) and ECOR 1052 (Fundamentals of Engineering II). Furthermore, SYSC 3303 (Real Time Concurrent Systems) heavily applies itself in this project where we will seek to replace a state-machine based system, with a subsumption based architecture which may make use of multi-threading, which are topics all covered in the course.

The hardware required to operate the robotics for this project requires knowledge of circuitry as well experience with hardware and software integration. As previously mentioned, in SYSC 3010 students have learned how to integrate hardware and read data in the project. As well, circuit knowledge as well as electronic theory was learned in ELEC 2501 (Circuits and Signals) along with ELEC 2507 (Electronics I). Additional experience in integration with hardware sensors and software was acquired in SYSC 4805 (Computer Systems Design Lab).

Developing the web application will draw heavily from techniques learned in SYSC 4504 (Fundamentals of Web Development) and SYSC 4806 (Software Engineering Lab). Also, to support the functionality of the web application, there will be a database constructed according to principles learned in COMP 3005 (Database Management Systems). Throughout the project, the material covered in SYSC 4120 (Software Architecture and Design) will be useful to manage the structure of the project. Also, testing knowledge gained from SYSC 4101 (Software Validation) will be applied here.

## 4 Group Skills

By Denis Cengu, Douglas Lytle, and Cholen Premjeyanth

The project team consists of three members, with members from both the computer systems engineering and the software engineering programs. This gives the team collectively a broad knowledge base of engineering principles, design experience, and tools which will be needed to complete the project.

- **Denis Cengu:** Denis is a fourth-year computer systems engineering student with experience in C++, Java and in particular Python. He has previous experience developing a smart device operated with multiple Raspberry Pi's which involved substantial work in embedded systems and hardware integration. He also has experience using Firebase for backend services and real-time data management. His background in embedded systems and hardware integration should serve useful for close interaction between software and physical components.

- **Douglas Lytle:** Douglas is a fourth-year software engineering student with a good knowledge of software design and experience in various programming languages such as Java, C, and Python, which will provide a great background for working on the robot. Douglas also has experience with web development using tools and languages such as HTML/CSS, JavaScript, and PHP, which should help with the continued development of the web application.

- **Cholen Premjeyanth:** Cholen is a fourth-year computer systems engineering student with an experienced background in Java, Python, and C. He has previous experience developing a 3D printer monitor which involved a Raspberry Pi, and focused on hardware integration. He also has experience using Arduino in a snow remover based project. His background in both software design and hardware implementation will serve useful in this project.

# 5 Methods

### 5.1 Software Development Methodology

By Douglas Lytle

For the continued development of this year's project, the team will be working using an Agile software development model. This will allow any design flaws to be discovered as early as possible, and allow for the goals of the project to be iterated upon as more work is completed. The many benefits of an Agile methodology were learned in SYSC 4106 (Software Economics and Project Management). The team will also try to follow the practices of Continuous Integration, which were learned about in SYSC 4806 (Software Engineering Lab). Continuous Integration will allow for changes to individual modules in the system to be made more freely and frequently, without fear of a long and difficult integration period.

### 5.2 Improving Robot Navigation

By Douglas Lytle

In order to accomplish the project objective of heavily simplifying the state machine used for navigation, which is a necessary goal for the current and future development of the project,

the team will transition the project away from the state pattern. The benefits and limitations of the state pattern were learned in SYSC 3110 (Software Development Project), and in the context of this project it adds more complexity than it alleviates. Thus, the robot control system will be modified to use a subsumption architecture, which is extremely suitable for a mobile robot. In a subsumption architecture, the system is split into modules which are each responsible for a certain type of navigation or response to sensor readings. Each module will constantly output some instructions for the robot, based on the type of behavior it represents, and what is detected by sensors. Then, a central control module will decide which of the other modules' instructions should be listened to and performed, based on a priority system. This will make use of principles learned in SYSC 3303 (Real Time and Concurrent Systems).

### 5.3 Improving Reliability of Intersection Detection/Traversal

By Cholen Premjeyanth

To enhance the reliability of intersection detection in our autonomous robot system, the team will utilize Gazebo for extensive simulation testing. Gazebo is a powerful robotics simulator that allows for realistic 3D environments to be modeled, enabling us to test various navigation algorithms while not putting the robot at risk of damage. By simulating various intersection scenarios, we can analyze the robots behavior in various conditions. Furthermore, by using simulations to test the robot we can analyze the robot more efficiently, rather than consistently trying to set up the same tests physically. This approach is similar to an approach I learned in SYSC 3020 (Intro to Software development), which emphasizes the importance of testing and validation within the software development life cycle.

Furthermore, the use of the LiDAR sensor will be important to improving the reliability of the intersection detection.We can use sensor fusion even to improve the detection of intersections, as through combining sensor data, the robot will be more accurate. In SYSC 3303 (Real Time and Concurrent Systems) we learned how to incorporate synchronization for multiple sensor inputs and explored strategies relating to real-time systems and their responses. This will be helpful in improving intersection detection. Ultimately, Gazebo testing will lead to the development of better algorithms for intersection detection and traversal, ensuring that the robot can navigate complex environments effectively and safely.

### 5.4 Improving the Web Application

By Douglas Lytle

To reach the project's goal of continuing to improve the web application, some new features will be added to the web application, and all existing features will be tested rigorously to ensure they function as expected. Namely, the team plans to add a visual map of the tunnels which will display the robot's approximate location. Due to not being able to rely on Wi-Fi or GPS in the tunnels, this will be represented as the tunnel segment the robot is currently in (what two beacons it is between). This feature will enhance the user experience by displaying delivery progress in real time. This will make use of techniques learned in SYSC 4504 (Fundamentals of Web Development). A mock-up of such a map display can be seen below in figure 2.

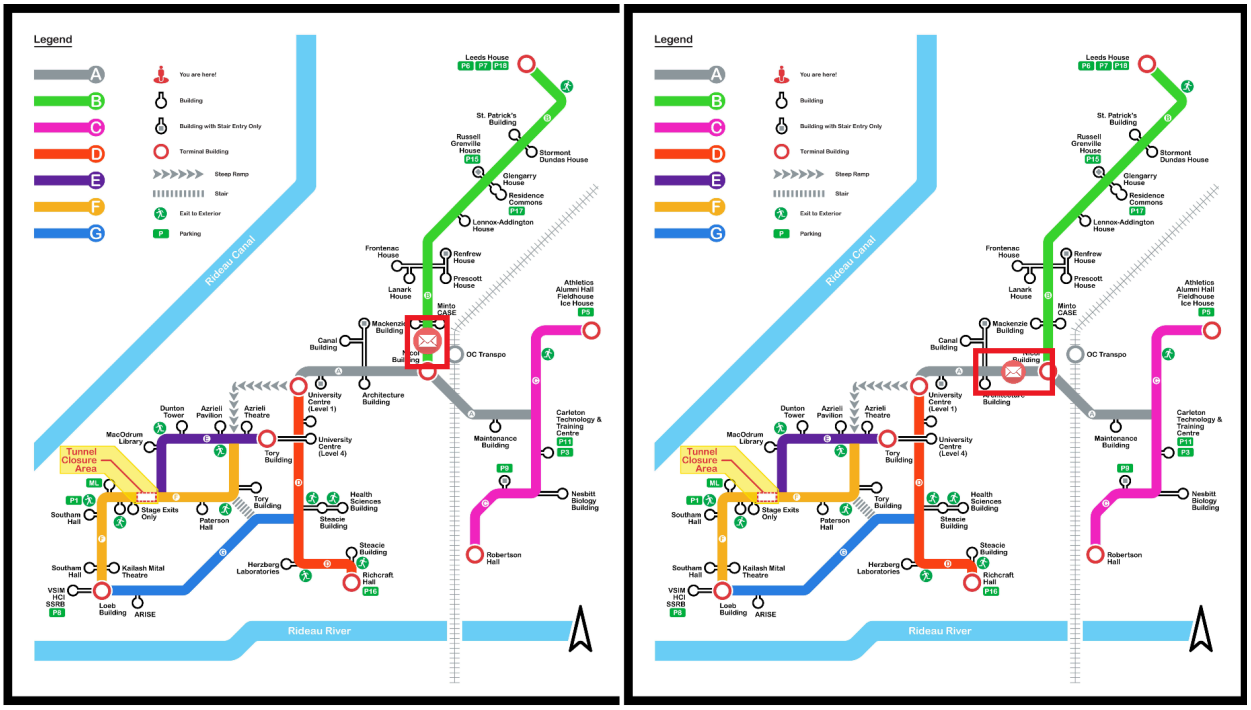


Figure 2: Web app map mockup. The robot is seen in between Minto CASE and the Nicol Building (Left). Then, as it moves it is seen in between the Nicol Building and the Architecture Building (Right).

# 6 Testing and Metrics

## 6.1 Testing Plan

By Denis Cengu

A robust testing plan will be employed for the shift to a subsumption model to verify all modules function correctly in various scenarios and that the subsumption framework itself properly handles task prioritization, such as obstacle avoidance preempting navigation.

The testing plan will be divided into two stages. Unit tests will be used for individual modules to ensure that each meets their functional requirements. For example, the wall following module is tested in isolation to verify the ability of that individual module. Afterwards, integration tests will be used to verify how the modules will work with each other within the subsumption framework. These tests are meant to verify how well modules interact with one another while simulating a real world scenario mainly focusing on proper behavior switching and task execution. Each test will be automated where possible, using ROS 2 available tools such as pytest for unit and integration tests. The goal of the testing plan is to create a simple, yet comprehensive test system which will simplify deployment of upgrades and changes.

### 6.2 Performance Metrics
By Denis Cengu

In addition to verifying functionality, it is imperative to measure how well the robot performs each task, whether it is done efficiently or effectively. Performance metrics will play a key role in this evaluation, providing objective and subjective measures of success. The focus will be on creating and using factual metrics along with qualitative metrics.

Factual metrics will include objective measurements such as task completion time, accuracy in navigation, battery consumption and how often the task is successfully completed. These metrics will allow the measurement of efficiency in the system and pinpoint areas which may require improvement.

Qualitative metrics will assess less tangible aspects, such as how smoothly the system transitions behaviors, how reliably dynamic environments are handled as well as how well a turn is made, among other things. These qualitative assessments will help determine how the system is performing in a real world application, beyond just satisfying technical requirements.

Through a combination of both factual and qualitative performance metrics, it will give a comprehensive understanding of not just whether the system can achieve its goals but how well it achieves them, allowing for diligent identification of areas where improvements can be made.

#### 6.2.1 Specific Metrics
By Douglas Lytle

Detailed below are some empirical metrics which will be used to evaluate the performance of the system, by comparing the actual performance with these targets. These metrics are designed to be able to be automatically tested and evaluated in Gazebo. A mock-up of a dashboard view which will automatically display an evaluation of system performance against these metrics can be seen below in figure 3.

- **M1**: The robot should stray no more than 30 cm from the wall while following it.
    - This value was chosen because it has been found that the IR sensors on the robot have an effective range of about 20 cm. While the robot is in that range, the IR sensors will be used for wall following. However, a threshold of 30cm will allow the robot to enter that range using the technique developed by previous teams which uses the LiDAR sensor to control the angle of the robot with the wall. If the robot strays more than 20 cm from the wall, this will allow it to correct itself without being considered a failure.

- **M2**: The robot should detect intersections after having traveled no more than 1 meter into the intersection.

    - This value is selected based on tests done with the LiDAR sensor. Through multiple trials, it was seen that the intersection could only be clearly recognized after having traveled some distance into the intersection, typically 50 cm to 1 m.

- **M3**: The robot should complete turns through an intersection by following a path that is not longer than the sum of the width and length of the intersection.
    - This value was chosen since a path any longer than the sum of the width and length of the intersection being traversed would constitute unnecessary backtracking.

- **M4**: The robot should not collide with any stationary obstacle during a delivery.
    - This metric is simple: it should be unacceptable for the robot to collide with any non-moving obstacle for obvious reasons.

- **M5**: The robot should complete deliveries in an amount of time proportional to $t = d/v$, where $t$ is the time elapsed during delivery, $d$ is the total distance from the starting location of the delivery to the destination, and $v$ is the operating speed of the robot, currently estimated to be around 0.2 m/s.
    - This metric's purpose is to ensure the robot wastes no movement. As much as possible, all movements the robot takes should bring it closer to its destination.

- **M6**: The robot should be able to automatically undock to begin a delivery, and automatically dock itself again after having completed a delivery.
    - These are both required behaviors, so if either docking or undocking fails, this metric will be considered not met.

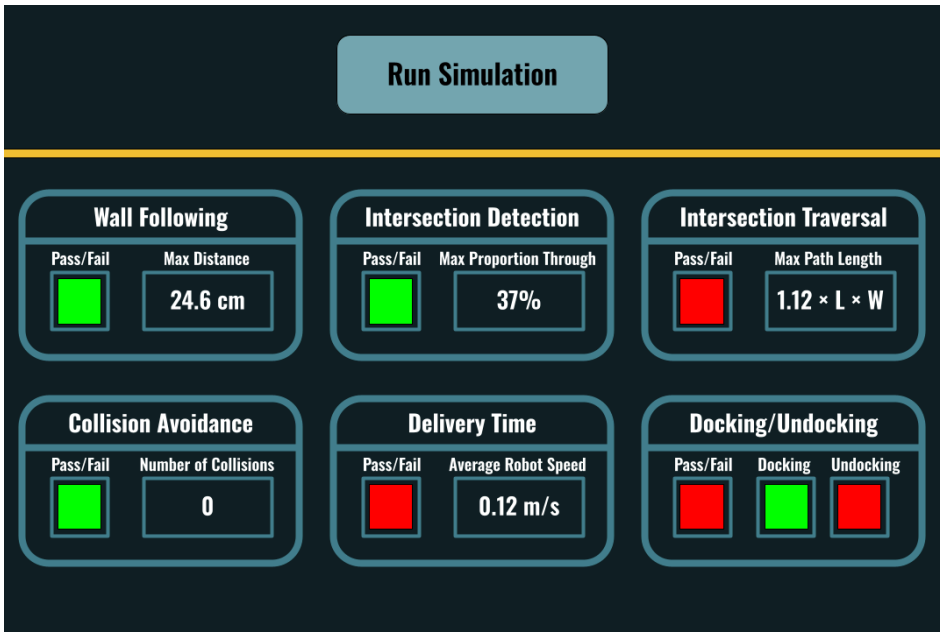


Figure 3: Mock-up of the Automated testing dashboard displaying system performance

# 7 Analysis

### 7.1 Analysis of Project Starting Point

By Douglas Lytle

As this is the fourth year of the Mail Delivery Robot project, there is significant work which has already been completed by past project teams. To begin this year's iteration of the project, the team began by examining in detail the report left behind by last year's project group.

It can be seen that last year at the conclusion of the project, the robot was able to follow walls well, navigate through intersections with reasonable consistency, avoid simple obstacles, follow its internal map to reach its destination, and communicate with the web application. All of this together allowed the robot to complete its first delivery.

However, there are also several issues which have been described by the previous project group which must be solved. There are significant improvements to be made in autonomously docking and undocking the robot, as well as reliability of intersection detection. Furthermore, the use of the state pattern seems to greatly complicate the development of the robot control system, and thus should be heavily simplified, or if this is not feasible, removed. Finally, for more accurate testing of the robots and their sensors, Gazebo should be used for simulation.

### 7.2 Analyzing the Capabilities of the Equipment

By Denis Cengu and Douglas Lytle

To better understand the initial capabilities of the robot and equipment, several tests were conducted. The results of these tests can be seen in the following subsections.

#### 7.2.1 Analyzing the LiDAR

By Denis Cengu

The LiDAR was tested in various intersections using RViz to provide a visualization. It was determined to be very reliable in ranges of 3m-5m and still providing useful information upwards of its maximum of 12m. In particular for the scope of the project, the LiDAR is more than capable of providing the robot with sensor data for wall following and intersection detection, with the latter requiring further refinement in its methods. This can be seen in the figures below which depict the robot first coming into an intersection and then roughly 50cm into an intersection. The red lines represent the walls detected by the LiDAR and the blue X represents the robot's location within the visualization.

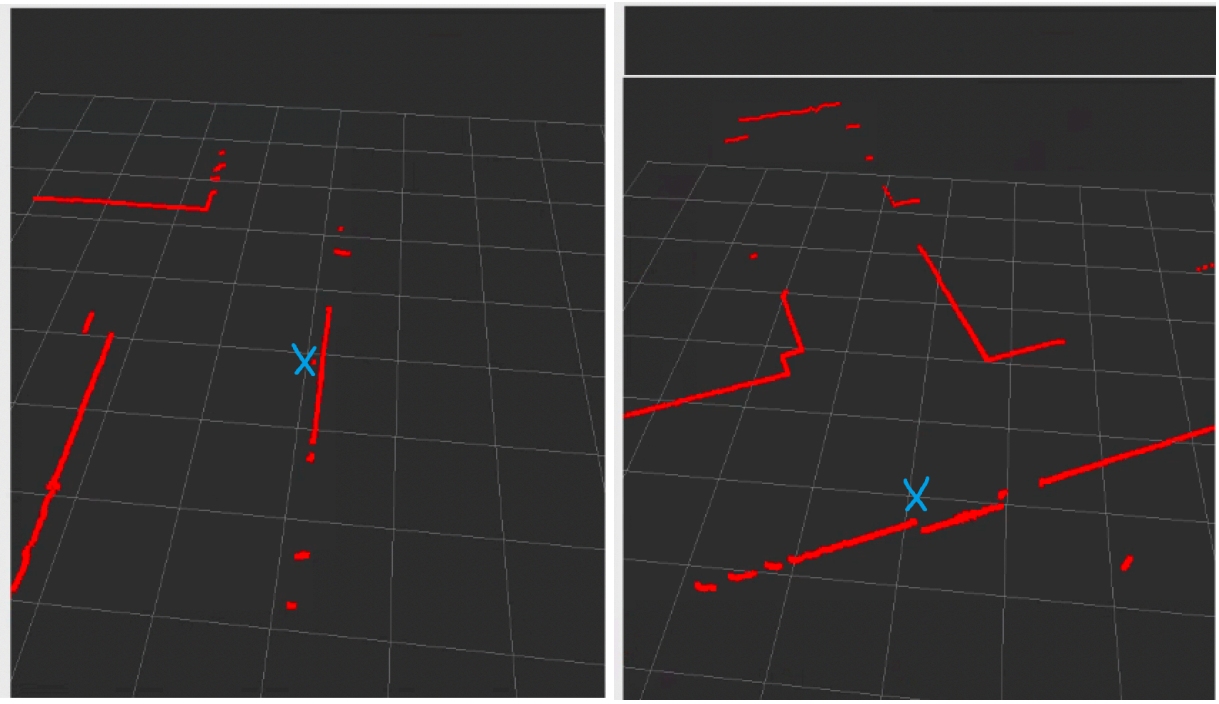

Figure 4: LiDAR Visualization at the Beginning of an intersection (left) and 50cm into an intersection (right)

### 7.2.2 Analyzing the Beacons

By Denis Cengu

The beacons similarly to last year's team were found to be somewhat unreliable in their readings potentially due to noise, low battery power and distance among other factors. Despite this, the data represents a clear enough trend that should allow the team to further build on last year's method of reading a gradient to determine whether a beacon is being approached or not. The strength of the beacons was measured five times at each distance, and in the following table the averages of those readings can be seen.

Table 13: Measured beacon signal strength at 1 and 10 meters

| Beacon Address | 1 Meter Strength | 10 Meter Strength |
|---|---|---|
| 78:46:f6:40:db:5b | -67dB | -84dB |
| 40:83:3d:c7:ec:98 | -69dB | -82dB |
| fb:ef:5c:de:ef:e4 | -74dB | -88dB |
| 51:c6:33:c1:41:94 | -67dB | -85dB |
| ee:16:86:9a:c2:a8 | -64dB | -82dB |
| e4:87:91:3d:1e:d7 | -73dB | -90dB |
| df:2b:70:a8:21:90 | -72dB | -89dB |
| ef:08:20:32:8d:a0 | -64dB | -91dB |

# 8 Project Timeline

The proposed timeline for the project can be seen below in table 2, as well as in the form of a Gantt chart in figure 4.

### 8.1 Proposed Timeline

By Denis Cengu and Douglas Lytle

Table 14: Proposed project milestones and target completion dates

| Milestone | Target Completion Date |
|---|---|
| Introduce Gazebo simulations for testing the system | October 16th, 2024 |
| Complete the implementation of last year's system in Gazebo to gather performance data | October 25th, 2024 |
| Proposal | October 28th, 2024 |
| Redesign mail holder | Nov 1st, 2024 |
| Complete the design of the new system architecture | Nov 5th, 2024 |
| Implement stubs for everything so that any new code can be tested in Gazebo in the context of the new architecture | Nov 12th, 2024 |
| Implement modules for individual behaviors | Nov 26th, 2024 |
| Implement module responsible for behavior switching | Dec 3rd, 2024 |

| | |
|---|---|
| Refine behavior switching | Jan 21st, 2024 |
| Oral Presentation | TBD (January 20th-24th) |
| Improve performance of modules for individual behaviors | Feb 11th, 2024 |
| Implement the new web app features | Feb 25th, 2024 |
| New system is able to complete a delivery with performance close to the targets | March 18th, 2024 |
| Poster Fair | March 28th, 2025 |
| Final Report | April 8th, 2025 |

### 8.2 Gantt Chart

By Douglas Lytle

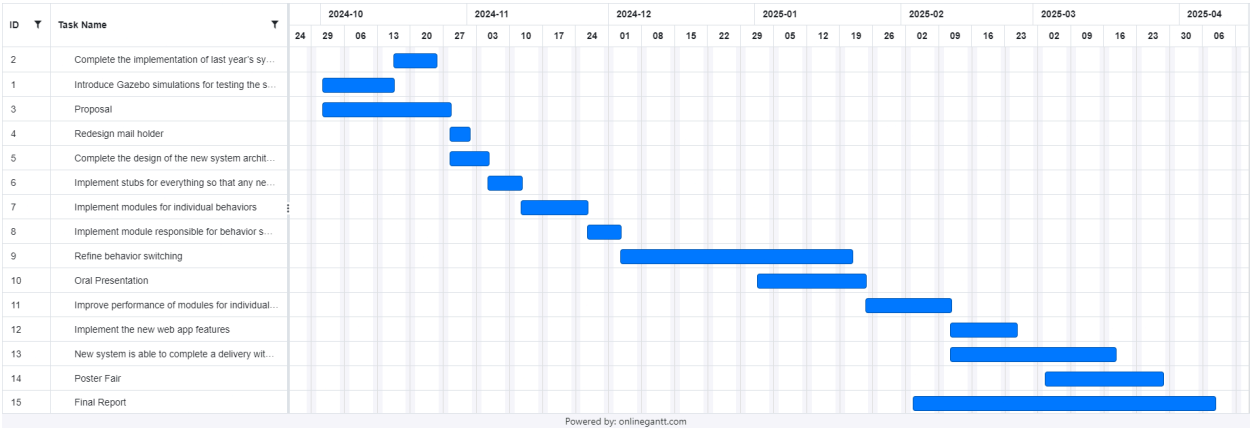


Figure 5: Gantt chart showing project timeline

## 9 Risks and Mitigation Strategies

By Douglas Lytle

Table 15: Risks to the project and strategies to mitigate them

| Project Risk | Mitigation Strategy |
|---|---|
| The existing Bluetooth beacons might not have sufficient signal strength/reliability or battery capacity to be effective for navigation through the tunnels. | The team will take care to test the beacons and ensure they are functional for their required purpose. If they are found to not be sufficient, enough project budget will be maintained so as to potentially buy more powerful beacons. |

| | |
|---|---|
| The robot could be damaged during testing or operation. | The team will take every possible precaution in order to avoid both damage to the robot, as well as its surroundings. |
| Integration of code could prove difficult and very time-consuming if the team has different or incorrect assumptions about the functions of modules of the system. | Practices of continuous integration will be used to ensure no code can be published to the main branch without first being rigorously tested in the context of the complete system. |

# 10 List of Required Components and Facilities

By Douglas Lytle

Table 16: List of required components/facilities, purposes, and estimated costs

| **Required Component/Facility** | **Purpose** | **Estimated Cost** |
|---|---|---|
| Tunnel Access | As the purpose of the robot is to deliver mail in the tunnels, tunnel access will be needed at times to test the robot in a real environment. | $0 |
| Robots/Sensors | Already purchased in past years. | $0 |

## 10.1 Justification of Purchases

By Douglas Lytle

The team has not yet identified any purchases which will be necessary for the continued development of the project. This will allow us to maintain enough project budget to replace any needed hardware in the event of a catastrophic failure, and also to upgrade certain components such as the Bluetooth beacons if it is deemed necessary.

## 11 References


[1] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, and W. Woodall, "Robot Operating System 2: Design, Architecture, and Uses In The Wild," *Science Robotics*, 2024. Available: https://www.science.org/doi/10.1126/scirobotics.abm6074.

[2] "iRobot Create 3 Hardware Overview," iRobot Education, Accessed: Oct. 8, 2024. [Online]. Available: https://iroboteducation.github.io/create3_docs/hw/overview/

[3] "iRobot Create 2 Programmable Robot," iRobot, Accessed: Oct. 8, 2024. [Online]. Available: https://www.irobot.ca/en_CA/irobot-create-2-programmable-robot/RC65099.html#:~:text=Create%202%20is%20ready%20to,controlled%20via%20Open%20Interface%20Commands

[4] “Thymeleaf”. Thymeleaf Team. Accessed: Oct. 13, 2024. [Online]. Available: https://www.thymeleaf.org/

[5] “Azure App Service”. Microsoft. Accessed: Oct. 13, 2024. [Online]. Available: https://azure.microsoft.com/en-ca/products/app-service/

[6] “Gazebo”. Open Robotics. Accessed: Oct. 12, 2024. [Online]. Available: https://gazebosim.org/home

[7] RPLIDAR A1: “RPLIDAR A1 Datasheet.” SLAMTEC. Accessed: Oct. 15, 2024. [Online]. Available: https://bucket-download.slamtec.com/d1e428e7efbdcd65a8ea111061794fb8d4ccd3a0/LD108_SLAMTEC_rplidar_datasheet_A1M8_v3.0_en.pdf

[8]
“Spring Framework.” Pivotal Software. Accessed: Oct. 15, 2024. [Online]. Available: https://spring.io/projects/spring-framework