Autonomous Mail Delivery Robot

Final Report

Max Curkovic 101139937
Cassidy Pacada 101143345
Bardia Parmoun 101143006
Matt Reid 101140593

Supervisor: Dr. Babak Esfandiari

Department of Systems and Computer Engineering
Faculty of Engineering
Carleton University

April 10th, 2023

Table of Contents

List of Tables

List of Figures

1 Objectives

1.1 Project Objectives

By Max Curkovic

In this term, the objectives of this project are to develop the robot in a way that it will be able to autonomously move through the tunnels, be able to detect and avoid anything in its path, (e.g. carts, people) and avoid hitting walls. The robot should be able to maintain the power necessary to complete its journey to its destination. The robot should have a way of carrying mail and should ensure that it does not fall off during the travel time. The robot should also be controlled by a user-friendly web application that is able to set the desired location.

The project itself should remain under the given $500 budget when completing these tasks. The team will measure each objective iteratively through a series of tests that can competently consider each case. Our group believes that the robot will be in a state in which it can successfully complete a mail delivery trip to a building by the end of the Winter 2023 term.

1.2 Project Description

By Bardia Parmoun and Matt Reid

This will be the third year of this project. As stated in the original proposal for the project, its main goal is to create an autonomous robot that can deliver mail between the different buildings at Carleton University through the tunnels. The project will also include a web application that easily allows the users to place their orders and manage the deliveries.

The previous team focused mostly on the functionality aspect of the robot as such a lot of their work was focused on ensuring the robot's movements were as expected. This task included ensuring the following: wall following, passing through intersections, right turns, and left turns. To achieve this task they also added a new state machine design for the system.

Based on the conclusions in last year's report, it can be seen that although the robot's movements have improved since its first iteration, there still needs to be more upgrades in that regard. Based on their conclusions the main reason for these issues is the existing sensors on the robot. As a result, this year the team will primarily focus on ensuring the functionality of the robot is working perfectly. This task may involve obtaining new sensors and redoing some of the work for the wall following and turning movements. The goal is to ensure that the robot will have reliable movements and readings. Additionally, there was very little work completed on the web application for the robot. As such some of the development for the project will be allocated to working on the web application side of the robot.

2 The Engineering Project

2.1 Health and Safety

By Max Curkovic

All experiments and test runs with the robot are conducted in two specified locations: the Canal lab room (CB 5101) and the Carleton tunnels. Our team ensures that all tests are done in secluded areas, whether within the empty lab room or ensuring that there are no people obstructing the path of the robot if in operation. By doing this, the team takes every possible precaution to ensure the risk to public safety is minimal.

The team also ensures to take all possible precautions when working with the hardware. When working with voltmeters and converters, all team members ensure that hands stay away from the probes, and there are no watches or jewelry being worn. It is also extremely important for the Pi to be powered off and unplugged prior to changing the hardware and cables.

2.2 Engineering Professionalism

By Max Curkovic

ECOR 4995 is a course taken by fourth-year undergraduate students that aims to teach the practice of professional engineering. It equips students with knowledge of various topics, such as engineering ethics, professionalism, communication skills via reports or presentations, and legal obligations. Bearing this in mind, the team has vowed to uphold these principles over the duration of the project:

- Clear, open, and honest communication amongst team members during meetings and work sessions.
- Strong writing skills upon writing the proposal, progress report, and final report, and presenting these reports in a clear and concise manner to the project supervisor.
- Considering public safety as the highest priority during project development.
- Adhering to all ethical regulations and practices during project development, including respecting intellectual property rights through utilizing correct references.
- Using a project development methodology (in the team's case, an agile development process), and adapting to any changes that may arise within the planning or scheduling of milestones.

2.3 Project Management

By Max Curkovic

The size and scope of the project required for the team to implement several project management techniques over the course of the term. Each technique is described in succeeding sections.

2.3.1 Project Timeline

By Max Curkovic, Cassidy Pacada, Matt Reid, and Bardia Parmoun

Table 1: Project milestones and proposed target completion dates in table form

Milestone	Target Completion Date
1 - Design a new chassis that accommodates better stands for the components and print it.	September 30th, 2023
2 - Prepare a prototype using the lidar sensor and compare performance with existing sensors	October 13th, 2023
3 - Proposal	October 20th, 2023
4 - Introduce unit tests and integration tests	October 27th, 2023
5 - Improve the state machine and implement new designs	November 4th, 2023
6 - Improve wall following and the PID controller	November 18th, 2023
7 - Fix the existing turn behavior and finalize movements	January 1st, 2023
8 - Replace the hard-coded navigation with a dynamic one	January 8th, 2023
9 - Create a simple prototype for the web app with simple commands: start, stop, status log, and delivery	January 22nd, 2023
10 - Oral Presentation	January 29th, 2023
11 - Add simple collision handling OR integrate iRobot's collision handling	March 10th, 2023
12 - Poster Fair	March 17th, 2023
13 - Final Report	April 12th, 2023

2.3.2 Gantt Chart

By Max Curkovic, Cassidy Pacada, Matt Reid, and Bardia Parmoun

Each project milestone is divided as its own separate task, with Figure 4 (Gantt chart) being an agile, visual representation of when each milestone should be completed for. Overall, each task is divided into its own “milestone”, subsumed by different cycles. The cycle typically starts with implementing any new states that were designed as part of the state machine, then considers all unit and integration testing for each task, and then allows for time to work on the project report. If this cycle is completed, then the team progresses to the next milestone.

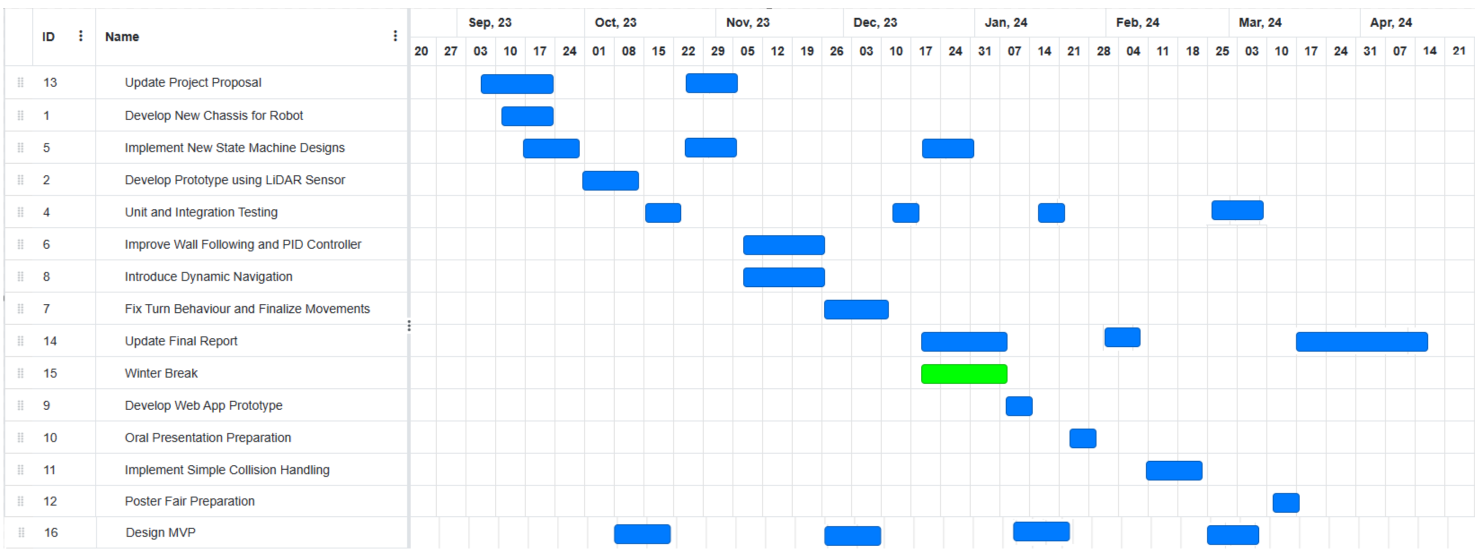


Figure 1: Project milestones and proposed target completion dates in Gantt chart form

2.3.3 Risks and Mitigation Strategies

By Cassidy Pacada

Table 2: Possible risks and their mitigation strategies

Project Risk	**Mitigation Strategy**
Potential hardware failure can slow project progress as it takes time to re-solder circuitry. Should a larger, more important piece fail, such as the iRobot itself, it may take an extended period of time to replace.	The current faulty circuitry will be replaced with new equipment to minimize the risk of failure. Additionally, the team has extra pieces of critical hardware on hand in case a failure does occur.
Making incorrect changes to the code can potentially set the project back if there is no way to recover previously functional code.	The team will use a version control system, Github, to ensure that changes can be reverted if necessary.

There is a risk that the existing sensors will not be sufficient to navigate the tunnels effectively. Should this implementation strategy fail with no backup, it would leave the project directionless.	The team has come up with alternatives to fall back on if the initial navigational implementation does not work. These include using Lidar sensors and computer vision to navigate.
Any test of functionality where hitting an obstacle can cause damage to the robot or its hardware can potentially lead to costly replacement of said hardware.	The team will supervise and monitor all tests of functionality whenever they are performed, as someone can step in and stop the robot if absolutely necessary. All precautions to protect the robot (e.g. ensuring the chassis is in place) will also be taken prior to testing.

2.4 Justification of Suitability for Degree Program

By Max Curkovic and Bardia Parmoun

The team consists of four people. Three of the team members, Bardia Parmoun, Cassidy Pacada, and Max Curkovic are software engineering students. Matt Reid, the fourth member, is a computer systems engineering student. Both programs are closely related and focus on major areas in the field of software and computer engineering. These are namely: embedded development, requirements engineering, web development, real-time systems, etc. The development of this project will span all of these areas.

The majority of the code base for the robot has been developed in Python and the Robot Operating System (ROS). The code for the robot itself is being run on a Raspberry Pi. These align very closely with two of the engineering courses that all engineering students take, namely ECOR 1051 (Fundamentals of Engineering I) and ECOR 1052 (Fundamentals of Engineering II). In addition, there are various state machines involved in implementing the main functionalities of the robot. This is closely related to the curriculum for SYSC 3303 (Real Time Concurrent Systems).

The robotics hardware for the project requires knowledge of circuit design as well as integration between hardware and software systems. A fundamental understanding of circuit design and electronics theory was acquired in ELEC 2501 (Circuits and Signals) and ELEC 2507 (Electronics I). Experience with communicating between hardware sensors and software including communication protocols and end-to-end testing was acquired through various design projects and labs in the Computer Systems Engineering courses SYSC 3010 (Computer System Development Project) and SYSC 4805 (Computer Systems Design Lab).

Furthermore, the web application portion of the project is related to the material covered in SYSC 4504 (Fundamentals of Web Development) and SYSC 4806 (Software Engineering

Lab). It is also worth mentioning that requirement analysis and software architecture will be covered in every step of the project which are the main focus of the following courses, SYSC 3020 (Introduction to Software Engineering), SYSC 3120 (Software Requirements Engineering), and SYSC 4120 (Software Architecture and Design). Finally, there will be some 3D design work done for the physical aspects of the robot such as its chassis which was covered in ECOR 1054 (Fundamentals of Engineering IV).

As illustrated, the mail delivery system project spans various aspects of the software engineering and computer systems engineering programs at Carleton University.

2.5 Individual Contributions

By Max Curkovic

The sections below detail the contributions made from team members, on both the overall project and the final report.

2.5.1 Project Contributions

By Max Curkovic

- Max Curkovic: Navigation, integration and unit testing.
- Cassidy Pacada: Integration and unit testing.
- Bardia Parmoun: Controller and state machine implementation.
- Matt Reid: Hardware, electrical and chassis work.

2.5.2 Report Contributions

By Max Curkovic

Contributions for each report section are given in the sub-heading beneath each section title. Each section is divided equally amongst team members.

3 Background

By Max Curkovic

The autonomous mail delivery service robot project is intended to streamline the transportation of mail across Carleton University, through the use of autonomous robots driving through the tunnels. Professors often send mail through manual methods, usually by means of walking across campus to another building. The robot is designed with the idea of operating without any interference from humans and shortening the overall time to traverse the tunnels and deliver mail to a specified location.

Currently, the robot relies on Bluetooth connectivity to traverse the tunnels. Various sectors of the tunnels do not have cellular service, thus the robot cannot utilize typical navigation methods such as a GPS. The robot utilizes Bluetooth-supported beacons to support navigation by creating a graph representation of the tunnels. IR sensors are used to sense the robot's movement and detect any obstructions in its path, as the tunnels are full of different walls and pillars.

3.1 Robot Operating System

By Matt Reid

The Robot Operating System (ROS) is a set of open-source libraries and tools used for robotic applications [1]. The provided drivers and algorithms for a wide range of sensors and actuators allow for much faster development and simpler source code. The mail delivery robot uses ROS2 which is an updated version of the original ROS1, and can be used on a microcontroller to communicate with the iRobot Create 2 platform. The current robot uses the Foxy Fitzroy distribution of ROS2 which supports the Ubuntu 20.04 platform used on the Raspberry Pi device being used as a controller. The ROS library is documented in both C++ and Python, and the current project code is written using Python. There are many pre-made ROS packages to support any new sensors or actuators that may be added to the project including LIDAR.

3.2 iRobot Create

By Matt Reid

The current robot is built on the iRobot Create 2 which is an educational version of an autonomous iRobot vacuum that connects over a serial cable to a microcontroller to receive commands [2]. As discussed in Section 2.1, ROS is used to send commands from a Raspberry Pi to the serial port on the iRobot Create 2. 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 works like a Bluetooth beacon and creates a wall that the robot will not pass.

A new version of the platform, called the iRobot Create 3 has ROS2 built-in and can connect with WiFi or Bluetooth instead of using a microcontroller [3]. The robot has 7 IR obstacle sensors built in to detect objects and follow walls. It also has a custom faceplate which allows for easy mounting of boards through many screw holes, and cable passthrough to the storage compartment. The built-in ROS2 includes a full interface of ROS commands to take sensor data, follow walls, and positional navigation. Inside the storage component, it provides a 14.4V/2A battery connection as well as a 5V/3A usb-c port that can be used to power and can connect to a microcontroller if needed. This microcontroller connectivity would allow for a Raspberry Pi to be connected so the code run currently on the Create 2 could be easily adapted without major architectural changes, and both robots can be used at once.

3.3 LiDAR Sensor

By Matt Reid

A Light Detection and Ranging (LiDAR) sensor rapidly emits many light pulses that reflect off of objects and measures data such as the time elapsed and reflection angle in order to generate a 3D map [4]. This map can be used by the robot to navigate and avoid obstacles in the tunnels. By using a LiDAR sensor, the robot will be able to detect the tunnel walls at a much farther distance (~30cm with the IR sensors vs. 12m with LiDAR), allowing for navigation straight through an intersection without losing the walls of the tunnel. A LiDAR sensor has a much higher cost than other distance sensors, however with just one LiDAR sensor, an accurate 360-degree map of the robot's surroundings can be made.

For the robot, the team is proposing using the Slamtec RPLIDAR A1 which can measure 8000 times per second with a range of 0.15-12m with a resolution of 0.5mm [5]. The RPLIDAR A1 uses a modulated infrared laser which reflects off of the object and is processed by a vision acquisition system and DSP to acquire distance and angle measurements. It outputs the distance and heading measurements over UART meaning that it can be connected to the Raspberry Pi microcontroller currently being used to control the robot. Although a premade SDK is provided by Slamtec, in order to build on the current system, it will likely be needed to use the measurements sent over UART to create our own map of the surroundings and to follow walls as is currently done using the two IR sensors.

4 Group Skills

By Bardia Parmoun

As previously mentioned the team consists of members both from the software engineering and computer systems engineering program. This allows the team to collectively obtain all the skills required to complete the project. Here is a summary of the skills of each member:

- **Bardia Parmoun**: As a 4th-year software engineering student, Bardia has a lot of experience working with different languages such as Python, Java, C, and C++. Bardia also has some experience working with web development tools and languages such as HTML/CSS and JavaScript. Bardia also has some embedded software development experience and some electronics knowledge. Finally, Bardia also has some experience working with 3D designs and modeling.

- **Cassidy Pacada**: Cassidy is also a 4th year software engineering student with a lot of experience with Python and Java. Cassidy also has experience with front-end development using HTML/CSS and JavaScript. She also has some experience working in the field of cybersecurity which could be useful with regards to the security of the robot. Finally, Cassidy has a lot of experience working in Linux environments.

- **Max Curkovic**: Max is a software engineering student with knowledge of programming languages such as Python, Java, and C. Max also has a lot of experience with full-stack web development, which can help with the development of the web application associated with the robot. Max also has experience with Raspberry Pis and Linux.

- **Matt Reid**: Matt is a computer systems engineering student. As such he has a lot of experience working with embedded systems and hardware design. Matt has also done some projects with Raspberry Pis and Arduino microcontrollers. In addition, Matt has previously worked with ROS which is the operating system that is used to control the iRobot's movement. Finally, Matt also has some electrical engineering knowledge which could help with the circuit designs for the robot.

5 Methods

By Cassidy Pacada

To accomplish the project objective of improving the robot's navigational capabilities, the group has chosen to implement several changes to the robot. Primarily, the existing sensors will be replaced with LIDAR which will provide more accurate data for the robot to work with. During testing, it was noted that the current sensors are ineffective after a distance of 30cm. This will be problematic in a larger tunnel area as the robot may lose the wall and be unable to navigate to its destination. As well, the robot's sensors are currently both on the right side. The team determined that the robot does not follow the wall effectively when the wall is on its left side. It also struggles with making turns consistently and is unable to prevent collisions with obstacles directly in front of it. LIDAR would provide a 360° view which would remediate the robot's sensing limitations.

Additionally, the team aims to implement a web application so that users can interact with the robot. This is to expand upon the user interface portion of the web application that was created during the previous year's project. The team will create the back-end of the application using Spring Boot as it is taught in SYSC 4806 (Software Engineering Lab) and will be easier for future students to manage.

Throughout the project, the group will be working using Agile methodology to allow for continuous improvement. This will aid in validating the project design and implementation choices since each iteration has a testing phase. Using Agile should prevent the team from discovering critical design failures towards the end of the project. The team learned about the benefits of this method in SYSC 4106 (The Software Economy and Project Management) and decided to follow it for this project.

The team will use many problem-solving methods to complete this project. The knowledge acquired during our degree program is essential and will be applied throughout the entire project. Requirements analysis, which was learned in SYSC 3120 (Software Requirements Engineering) has already been crucial in determining what the main focus of the project should be. As well, the team will need to figure out how to implement the new features in a way that is clean and maintainable for future groups which can be done using skills obtained in SYSC 4120 (Software Architecture and Design).

6 Analysis

By Max Curkovic, Cassidy Pacada, Matt Reid, and Bardia Parmoun

6.1 Analyzing the Equipment

By Max Curkovic, Cassidy Pacada, Matt Reid, and Bardia Parmoun

Firstly, the speed of the robot was measured. On average, it traveled approximately 0.19 m/s at its default setting.

Table 3: Measuring the speed of the robot

Distance (m)	Time (s)	Speed (m/s)
1	7.18	0.14
2.5	14.89	0.17
5	25.94	0.19

Overall, it can be concluded that at its current average speed, the robot is quite slow which could affect delivery times. The current metric for the delivery time of the robot was designed by this speed; however, the speed of the Roomba may be increased and the metric might be updated at a later date, once the team is able to develop a reliable navigation system that the robot can safely follow.

Secondly, the strength of the robot's behaviour implemented by last year's team was evaluated. This included: wall finding, right turn, and a left turn. The team's testing showed that the robot's current ability to find the wall and follow it entirely depends on the distance of the robot from the wall. Testing showed that, once the robot was further than 30cm away from the wall, its wall-following capabilities were not sufficient. The team believes that this is an issue that a LiDAR sensor will be able to resolve (see Background, section 2.3).

Table 4: The ability of the robot to find the wall and maintain it

Distance	Description (Missed, OK, GOOD, PERFECT)
10cm	Good
30cm	Good
50cm	Missed (sensor distance may not be high enough)
1m	Missed (sensor distance may not be high enough)

Similarly, the strength of the robot's right turns was tested. Overall, the right turns were solid. The robot can currently maintain a safe distance from a wall, and perform the right turn as it enters an intersection state. Trials 3 and 4 are suspected to also be an issue with the sensors, which will likely be corrected when the LiDAR sensor is in place.

Table 5: The ability of the robot to make a successful right turn consistently

Trial #	Description (Missed, OK, GOOD, PERFECT)
1	Perfect
2	Good
3	Missed (too close to the wall)
4	Missed (too far from wall)

Next, the strength of the robot's left turns was tested. This is the robot's biggest weakness - it was unable to perform a single left turn in any trial. The team believes that, once again, this stems from an issue of having two IR sensors, both on the right side of the room. Once the LiDAR sensor is in place, the left turns should be significantly more reliable.

Table 6: The ability of the robot to make a successful left turn consistently

Trial #	Description (Missed, OK, GOOD, PERFECT)
1	Missed
2	Missed
3	Missed
4	Missed

Overall, it can be observed that the turn behaviour for the robot needs a lot of improvement. The inaccuracy that is being seen in the behaviour is mostly due to the fact that the IR sensors are a bit unstable and have a small range. This is why the team is hoping to improve the behaviour using LiDAR. On the other hand, the robot's wall following behaviour is very stable whenever it is within a short distance of the wall meaning that the PID controller is quite effective and with a better input such as LiDAR the wall following behaviour will be perfect. As a result, the team is planning to reimplement the IR distance module to use LiDAR but copy over the PID controller to improve the output.

The team also measured the strength of the beacons at different distances. There are no concerns or reliability issues with the current beacons, and they should suffice for the team's implementation of the autonomous mail delivery service robot.

Table 7: Measuring the beacons at known distances

Beacon ID	Beacon Mac Address	Distance	RSSI
1	E2:77:FC:F9:04:93	1m	-73
1	E2:77:FC:F9:04:93	10m	-90
2	EA:2F:93:A6:98:20	1m	-69
2	EA:2F:93:A6:98:20	10m	-87
3	FC:E2:2E:62:9B:3D	1m	-71
3	FC:E2:2E:62:9B:3D	10m	-88.4
4	E4:87:91:3D:1E:D7	1m	-67
4	E4:87:91:3D:1E:D7	10m	-87
5	EE:16:86:9A:C2:A8	1m	-71
5	EE:16:86:9A:C2:A8	10m	-86.2
6	D0:6A:D2:02:42:EB	1m	-91
6	D0:6A:D2:02:42:EB	10m	-85
7	DF:2B:70:A8:21:90	1m	-69
7	DF:2B:70:A8:21:90	10m	-93
8	FB:EF:5C:DE:EF:E4	1m	-61
8	FB:EF:5C:DE:EF:E4	10m	-89

In conclusion, the beacons all seem functional and have a really good range. They can still be detected from a 10m range which is more than enough for the applications of this project; however, for the full scale of the project, the team is planning to purchase more of the same beacons.

Finally, the strength of the Wi-Fi signal of different sections of the Carleton University tunnels were measured. This was done to allow us to determine what is the best location for the various docking stations for the robot. These points also give us a rough estimate of where the beacons should be placed.

Table 8: Measuring the strengths of the internet signal in Carleton University tunnels

Location	Wi-Fi Speed (None/Bad/Good)	Comments
Entrance to Athletics	Good	40-50 mbps.
Athletics-Tunnel A	None	Measured at intersection.
Nesbitt-Pigiarvik	None	Both building entrances have minimal Wi-Fi speed, 10 mbps.
Entrance to Maintenance	Good	35-40 mbps.
Entrance to CTTC	None	N/A
Nicol-B-A	Good	50-60 mbps (use the intersection)
Minto-Mackenzie	Bad	Both entrances have decent Wi-Fi (30-40 mbps)
St Patrick's-Residence Commons	None	N/A
Architecture-Canal	Good	50-60 mbps. Main area.
Mackenzie-Canal	Good	70-80 mbps. Main study area.
Tunnel E	None	N/A
Tory-UC-Azrieli Entrance	Good	70-80 mbps.
Steacie Entrance	None	No service..
Richcraft-Loeb	None	No service.
Entrance to Steacie/Richcraft	Good	50-60 mbps.
Entrance to Herzberg	Good	70-80 mbps.
Entrance to Loeb	None	No Wi-Fi in tunnel F
Entrance to Southam	Bad	10-20 mbps.
Entrance to Library	Good	70-80 mbps. Main study area.

6.2 Beacon and Dock Station Placement

By Max Curkovic, Cassidy Pacada, Matt Reid, and Bardia Parmoun

The final project will utilize numerous docking stations scattered across campus grounds. When placing the beacons and the docking stations, it is important to account for various factors such as the range of the beacons, Wi-Fi strengths, and possible traffic on the campus. To overcome these issues, the team decided to group certain buildings together. This approach helps reduce the cost of the beacons and docking stations. It also ensures that the robot will always have reliable internet connections at its destinations so it could easily communicate with the web app. Since the team already established that the beacons are easily recognizable within 10m radius, they can be easily detected in the tunnels. In addition, the robot's built-in wall following behaviour should easily overcome the slight turns in the tunnel so there is no need to place any beacons there; however, it is important to have beacons at every intersection that that robot might encounter even if there is not a destination there. This will allow the robot to easily pass through the intersection. Finally, the team considered a beacon for each docking station so the robot could recognize them upon getting close to them. This will allow the robot to easily dock.

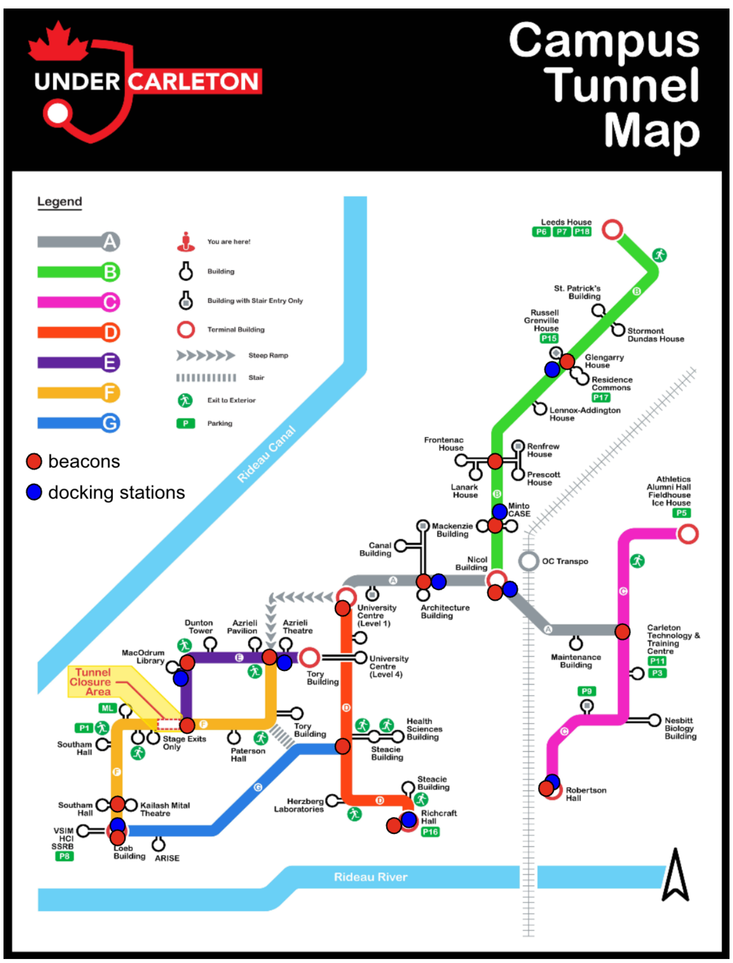


Figure 2: Proposed locations for the beacons and docking stations in the tunnels.

As shown in the diagram, the beacons used to identify the docking stations will also double as indicators of intersections for the robot. In other words, upon reaching the intersection the robot will check to see if it needs to dock using its builtin map and the ID of the docking station (or the beacon). If not, the robot will only note the intersection. Based on the Wi-Fi analysis that the team conducted, it can be guaranteed that the robot will have a reliable connection at every place that a docking station has been placed. In addition, the majority of the beacons that are placed at the extra intersections in the tunnels should also have reliable internet connection. This will allow the robot to easily send status updates to the user.

The team believes that docking stations should be placed in the following locations:

1. Residence common intersection for St. Patrick's Building and Residence Commons.
2. In front of Minto entrance, Minto Building.
3. At the Nicol Building intersection for the Nicol Building.
4. Near Robertson Hall for Maintenance, CTTC, and Nesbitt Buildings and Robertson Hall.
5. In front of the Architecture building for Mackenzie, Canal, and Architecture buildings.
6. In front of Richcraft hall for Steacie, Health Sciences, Herzberg, and Richcraft buildings.
7. At the tunnel E/F intersection for the Tory and Azrieli Buildings and University Centre
8. In front of the MacOdrum library for the library and Dunton Tower
9. In front of the Loeb building for Southam Hall, Loeb Building, and Paterson Hall.

In each individual tunnel, there will be numerous beacons to assist the movement of the robot as it navigates. The team believes that beacons should be placed in the following locations:

- Tunnel B:
    - Lanark/Renfrew House Entrance

- Tunnel C:
    - Tunnels A and C intersection

- Tunnel D:
    - Long ramp/Tunnel D turn
    - Tunnels D and G intersection

- Tunnel F:
    - Tunnels E and F intersection
    - Southam Hall/Kailash Mital Theatre

In addition to these, there will be a beacon above each docking station so the robot can easily distinguish them. This brings the total number of required beacons to (6 + 9) 15.

7 Design

7.1 Requirements

By Max Curkovic and Bardia Parmoun

Below is a list of all functional and non-functional requirements that the robot should accomplish in order to satisfy the project objectives.

7.1.1 Functional Requirements

- **R1:** The robot shall be able to send mail from one destination to another using the Carleton University tunnels.
- **R2:** The robot shall be able to retrieve mail from one destination to another using the Carleton University tunnels.
- **R3:** The robot shall be able to navigate the tunnels with great precision.
- **R4:** The robot shall be able to notify the systems of its status.
- **R5:** The robot shall be able to handle any collisions with the people and objects in the tunnels.
- **R6:** The robot shall be able to communicate with the Bluetooth beacons installed in the tunnels.
- **R7:** The system should allow the administrator to monitor its behavior by providing detailed logs.
- **R8:** The robot shall be able to dock itself upon reaching its destination or whenever it is low on battery.
- **R9:** The system should allow the users to make delivery requests.
- **R10:** The system shall notify the robot of new requests and provide it with a path.
- **R11:** The system shall notify the user with information regarding their deliveries.

7.1.2 Non-Functional Requirements

- **R12:** The robot control software shall be interoperable with the web display application.
- **R13:** The robot shall be able to travel at a speed of at least 0.15m/s through Carleton University's tunnels.
- **R14:** The beacons shall be able to be interfaced with by the robot within 14.2 meters.
- **R15:** The robot's battery shall maintain at above 50 percent while it performs a delivery.
- **R16:** The robot shall ensure that only intended recipients can access mail.
- **R17:** The project shall remain within the specified $500 budget.
- **R18:** The expected features of the project, as proposed in the timeline, shall be completed on their respective dates.
- **R19:** The web application shall only be accessible for users connected to the Carleton University Wi-Fi or VPN.

7.2 Use Cases

By Max Curkovic and Bardia Parmoun

For each functional requirement, a use case was designed in order to illustrate how each requirement will be performed, whether by the robot, the system, or the user.

7.2.1 Use Case Diagram

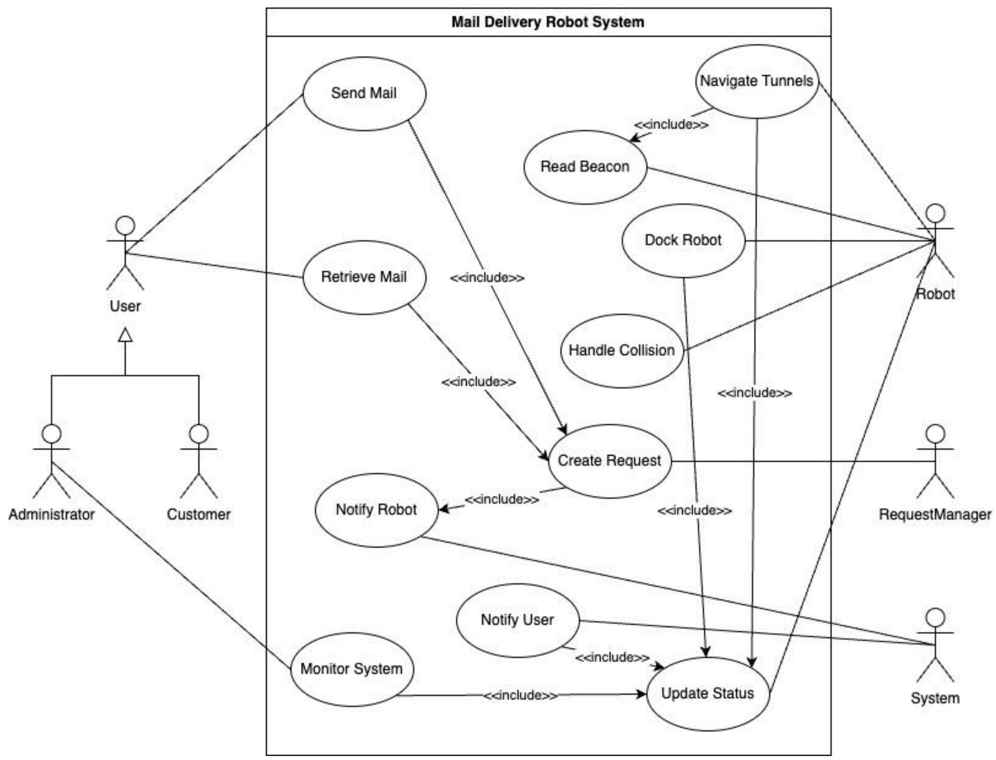


Figure 3: The use case diagram for the Mail Delivery Robot System.

7.2.2 Individual Use Cases

Table 9: Send Mail use case

Use Case Name	Send Mail
Brief Description	The user sends a request to send mail to a destination.
Primary Actor	User
Secondary Actor	RequestManager
Precondition	The system is running, the robot is turned on, and the beacons are installed.
Dependency	INCLUDE USE CASE Create Request
Basic Flow	Steps: 1. The sender opens a request to send mail. 2. The RequestManager receives and validates the request. 3. RequestManager asks for an available robot. 4. The sender places the package in the robot. 5. The system notifies the recipient of the mail. **PostCondition:** Mail is in the robot being delivered.
Specific Alternative Flows	RFS Basic Flow 2. 1. IF There are no available robots THEN the request will be placed in a pending queue ENDIF **PostCondition:** Request will be placed in a pending queue.

Table 10: Retrieve Mail use case

Use Case Name	Retrieve Mail
Brief Description	The user sends a request to retrieve mail from the robot.
Primary Actor	User
Secondary Actor	RequestManager
Precondition	The system is running, the robot is turned on, and the beacons are installed.
Dependency	INCLUDE USE CASE Create Request
Basic Flow	Steps: 1. Users (sender and recipient) are notified by the System that mail has arrived at the destination. 2. The recipient retrieves the mail from the robot. **Postcondition:** The recipient User has successfully received mail from the Robot, sent by the sender User.
Specific Alternative Flows	RFS Basic Flow 2. Steps: 1. IF The Recipient User does not retrieve the mail from the robot THEN record the error and provide the robot status ENDIF **Postcondition:** The robot returns to its home dock and proceeds with other requests.

Table 11: Navigate tunnels use case

Use Case Name	Navigate Tunnels
Brief Description	The robot is able to navigate the tunnels based on its given path. This includes navigating various intersections and turns.
Primary Actor	Robot
Secondary Actor	System
Precondition	The system is running, the robot is turned on and the beacons are installed.
Dependency	INCLUDE USE CASE Read Beacon INCLUDE USE CASE Update Status
Basic Flow	Steps: 1. The system provides the robot with a destination and a navigation path. 2. The robot uses its sensors to receive information from its surroundings. 3. The robot uses the beacon sensors to determine its required next action. 4. The robot calculates its next move 5. The robot reaches its destination. **Postcondition:** The robot successfully navigated the hallway.
Specific Alternative Flows	RFS Basic Flow 2. Steps: 1. IF The robot is unable to detect the sensor THEN update the system ENDIF **Postcondition:** The robot notifies the system.
Specific Alternative Flows	RFS Basic Flow 3. Steps: 1. IF The robot is unable to detect the beacons THEN update the system ENDIF **Postcondition:** The robot notifies the system.

Table 12: Update Status use case

Use Case Name	Update Status
Brief Description	The robot is able to send a status update to the System.
Primary Actor	Robot
Secondary Actor	System
Precondition	The system is running, the robot is turned on, the beacons are installed and a delivery must be in progress.
Dependency	N/A
Basic Flow	Steps: 1. The sender user requests the System for a status update. 2. The robot sends a positional update, as well as a delivery status, to the System. 3. The system notifies the sender user. **Postcondition:** The sender user and the System are now both aware of the delivery status.
Specific Alternative Flows	RFS Basic Flow 3. Steps: 1. IF there is no Internet in the Carleton tunnels, THEN save it as a cache and try again ENDIF. **Postcondition:** The sender user will still be updated on the delivery status.

Table 13: Handle Collisions use case

Use Case Name	Handle Collisions
Brief Description	The Robot collides with an object (wall, cart, etc.) and re-orients itself.
Primary Actor	Robot
Secondary Actor	N/A
Precondition	The system is running, the robot is moving through the tunnels.
Dependency	N/A
Basic Flow	Steps: 1. The Robot collides with an object. 2. The Robot bumpers detect that a collision has occurred. 3. The Robot reverses. 4. The Robot turns away from the object and re-orients in the right direction. **Postcondition:** The robot has successfully navigated through the collision.
Specific Alternative Flows	N/A

Table 14: Read Beacons use case

Use Case Name	Read Beacons
Brief Description	The robot detects a beacon signal traveling through the tunnels.
Primary Actor	Robot
Secondary Actor	N/A
Precondition	The system is running and the robot is moving.
Dependency	N/A
Basic Flow	Steps: 1. The robot detects the beacon and determines what intersection it is coming from. 2. The robot logs the information on the system. **Postcondition:** The robot has successfully provided the system with its beacon data.
Specific Alternative Flows	N/A

Table 15: Monitor System use case

Use Case Name	Monitor System
Brief Description	An administrator monitors the current state of the system.
Primary Actor	Admin
Secondary Actor	Robot
Precondition	The system is running, and the robot is operational.
Dependency	INCLUDE USE CASE Update Status
Basic Flow	Steps: 1. USE CASE Update Status is initiated. 2. The administrator requests to view the current state of the system. 3. The system replies with the current state. **Postcondition:** The administrator is informed of the current system state.
Specific Alternative Flows	RFS Basic Flow 2. Steps: 1. IF the system is not running OR the robot is not operational, THEN send an error ENDIF **Postcondition:** The administrator is informed of the current system state.

Table 16: Dock Robot use case

Use Case Name	Dock Robot
Brief Description	The robot is near a docking station and determines that it needs to dock.
Primary Actor	Robot
Secondary Actor	N/A
Precondition	The system is running and a docking station has been encountered.
Dependency	INCLUDE USE CASE Update Status
Basic Flow	Steps: 1. The robot gets near a docking station. 2. The robot determines that it needs to dock at the specific docking station. 3. The robot mounts the docking station. 4. The robot updates the system about its behavior. **Postcondition:** The robot is charging and the system is notified.
Specific Alternative Flows	RFS Basic Flow 3. Steps: 1. IF The robot is unable to mount the docking station THEN send an error ENDIF **Postcondition:** The system is notified of the error.

Table 17: Create Request use case

Use Case Name	Create Request
Brief Description	The request manager creates and validates a request for the sender user.
Primary Actor	RequestManager
Secondary Actor	Robot
Precondition	The system is running, and the robot is operational.
Dependency	INCLUDE USE CASE NotifyRobot
Basic Flow	Steps: 1. The sender user creates a request to the request manager to send or receive mail. 2. The request manager validates the user's request. 3. USE CASE Notify Robot is initiated to inform the robot of the request. **Postcondition:** The robot is notified of the request and will proceed to fulfill it.
Specific Alternative Flows	RFS Basic Flow 3. Steps: 1. IF the robot is not operational, THEN inform the user to try again at a later time. ENDIF **Postcondition:** The sender user is notified that the robot is not currently operational and will have to wait in order to put in a request.

Table 18: Notify Robot use case

Use Case Name	Notify Robot
Brief Description	The system notifies the robot of new updates. This includes new requests, navigation data, etc.
Primary Actor	System
Secondary Actor	Robot
Precondition	The system is running and the robot is operational.
Dependency	N/A
Basic Flow	Steps: 1. The system prepares a new update for the robot. 2. The system sends the update to the robot. **Postcondition:** The robot receives the update and acts accordingly.
Specific Alternative Flows	RFS Basic Flow 2. Steps: 1. IF The system is unable to contact the robot THEN inform the user and terminate the request ENDIF **Postcondition:** The system notifies the user of the error.

Table 19: Notify User use case

Use Case Name	Notify User
Brief Description	The user is notified of any updates to the robot, their delivery, or the system itself.
Primary Actor	System
Secondary Actor	Robot
Precondition	The system is running and the robot is operational.
Dependency	INCLUDE USE CASE Update Status
Basic Flow	Steps: 1. USE CASE Update Status is initiated. 2. The system posts the updated results for the user. **Postcondition:** The user is now informed of any updates to the robot.
Specific Alternative Flows	RFS Basic Flow 2. Steps: 1. IF The system is unable to contact the robot THEN inform the user ENDIF **Postcondition:** The system notifies the user of the error.

7.3 Metrics

By Bardia Parmoun and Cassidy Pacada

Here are some metrics which will be used to evaluate the system.

- **M1:** The system should cover all the major tunnel paths: A, B, C, D, E, F, and G:

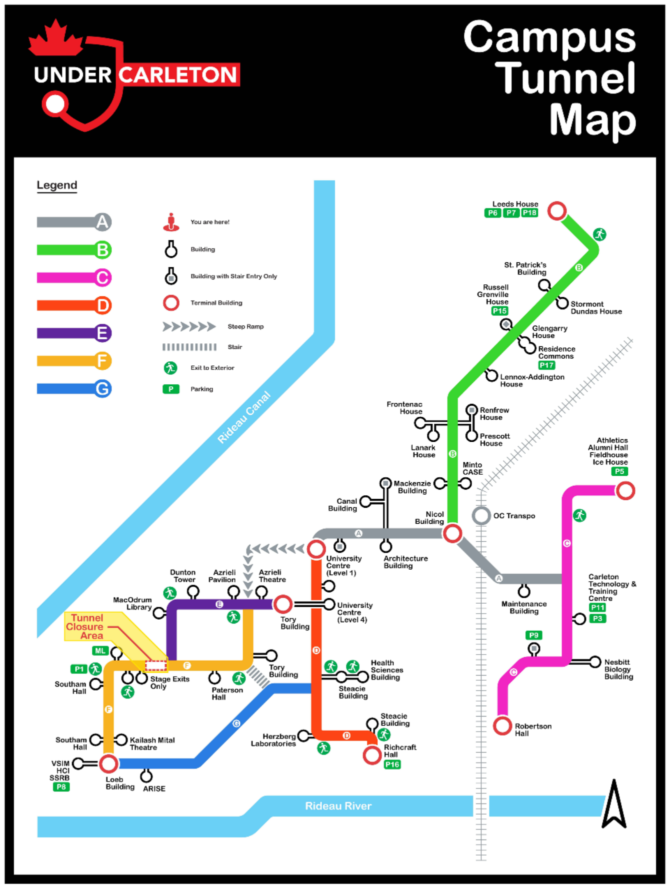


Figure 4: A map of the Carleton University tunnels.

To make the implementation easier for this iteration of the project, the robot is only going to traverse the mentioned paths meaning it will only recognize the major intersections and avoid the smaller detours along the path. For example, the robot for deliveries with destinations at the Canal and Mackenzie buildings the robot will only dock at the entrance of the detour leading to these buildings on path A.

- **M2:** The robot speed should be at least 0.2m/s and delivery time at most an hour.

- **M3:** The robot shall be able to deliver different envelope sizes: namely C5, C6, C7, DL, and Greeting Cards.

For reference here are the sizes of these envelopes:

- C5: 16.2cm x 22.9cm
- C6: 11.4cm x 16.2cm
- C7: 8.3cm x 11.2cm
- DL: 11cm x 22cm
- Greeting Card: 13.3cm x 18.4cm

- **M4:** The robot shall be able to deliver packages with a total weight of 1kg or less.

7.4 State Machine

By Max Curkovic, Cassidy Pacada, Matt Reid, and Bardia Parmoun

7.4.1 List of States, Events, and Actions

By Max Curkovic, Cassidy Pacada, Matt Reid, and Bardia Parmoun

Sources of Inputs:
The system contains of 4 separate source of inputs: the bumper sensor, the beacons, the LiDAR sensor, and any possible errors. Together these will help determine the states for the system.

States:

- OPERATIONAL: The overall state containing all the valid states that the robot can have.
    - NO_DEST: The robot is moving but has not received a beacon.
    - SHOULD_TURN_LEFT: The robot is moving and got a beacon to turn left.
    - SHOULD_TURN_RIGHT: The robot is moving and got a beacon to turn right.
    - SHOULD_PASS: The robot is moving and has received a beacon to pass.
    - SHOULD_DOCK: The robot has received a signal to dock.
    - HANDLE_INTERSECTION: The robot has arrived at an intersection.
    - DOCKED: The robot has reached a docking station.
    - COLLISION_NO_DEST: The robot received a collision with no signal.
    - COLLISION_TURN_LEFT: The robot received a collision with a left signal.
    - COLLISION_TURN_RIGHT: The robot received a collision with a right signal.
    - COLLISION_PASS: The robot received a collision while having a pass signal.
    - COLLISION_DOCK: The robot received a collision while having a dock signal.
    - COLLISION_INTERSECTION: The robot received a collision at an intersection.
- NOT_OPERATIONAL: The robot has encountered a fatal error and has stopped.

Events:

Since this is an embedded system, we should expect all the different inputs to occur at the same time. As a result, an event in the state machine is a snapshot of these input values.

Table 20: List of the transitions for the state machine based on the different inputs.

TRANSITION	ERROR	BUMPER	BEACON	LiDAR
1	FALSE	FALSE	NONE	FALSE
2	FALSE	FALSE	NONE	TRUE
3	FALSE	FALSE	LEFT	FALSE
4	FALSE	FALSE	LEFT	TRUE
5	FALSE	FALSE	RIGHT	FALSE
6	FALSE	FALSE	RIGHT	TRUE
7	FALSE	FALSE	PASS	FALSE
8	FALSE	FALSE	PASS	TRUE
9	FALSE	FALSE	DOCK	FALSE
10	FALSE	FALSE	DOCK	TRUE
11	FALSE	TRUE	NONE	x
12	FALSE	TRUE	LEFT	x
13	FALSE	TRUE	RIGHT	x
14	FALSE	TRUE	PASS	x
15	FALSE	TRUE	DOCK	x
16	TRUE	x	x	x

As shown in the table, the different sensors in the system could have different valus. The bumper sensor used for collison detection could either be activated(TRUE) or deactivated(FALSE). Similarly, the beacons could trigger 5 separate navigation events (NONE, LEFT, RIGHT, PASS, and DOCK). Finally, the LiDAR sensor could either indicate if a wall exists(TRUE) or not(FALSE). This table summarizes every combination of these inputs and marks them as transitions for the state machine. For certain transitions, some inputs are marked with x, meaning their value is not important in the transition.

Actions:

- WALL_FOLLOW: The robot will move at the angle given by the lidar sensor at 0.4m/s.
- L_TURN: The robot will make a 30 degree left turn with a 0.4m/s speed.
- R_TURN: The robot will make a 90 degree right turn with a 0.4m/s speed.
- FORWARD: The robot will go forward with no angle chang with a 0.4m/s speed..
- DOCK: The robot will go to the docking station.
- UNDOCK: The robot leaves the docking station.
- KILL: Stops the system.

To better demonstrate the relations between the various states, a state machine diagram was prepared as follows. Please note that in order to facilitate reading the state machine, the diagram was split into two separate diagrams. The first diagram shows all the events that do not involve collisions namely T1-T10 and T16. The second diagram shows the remaining T11-T15.

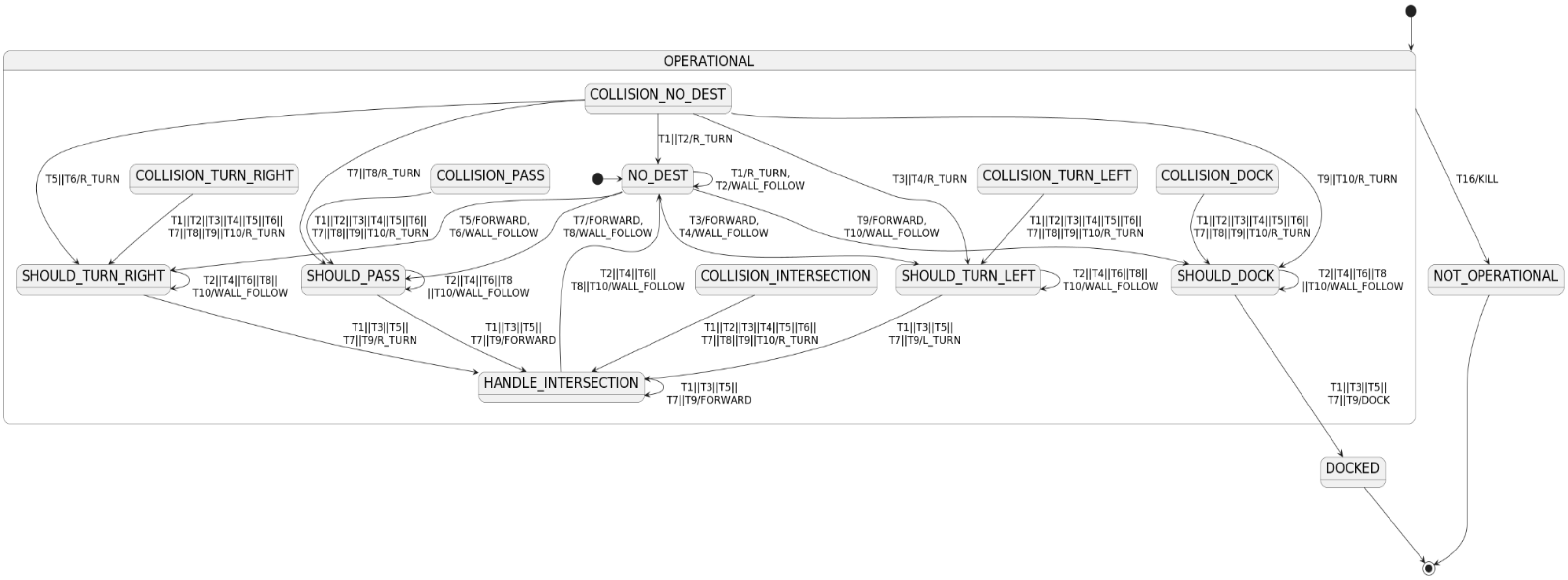


Figure 5: The proposed state machine for the system without the collision events

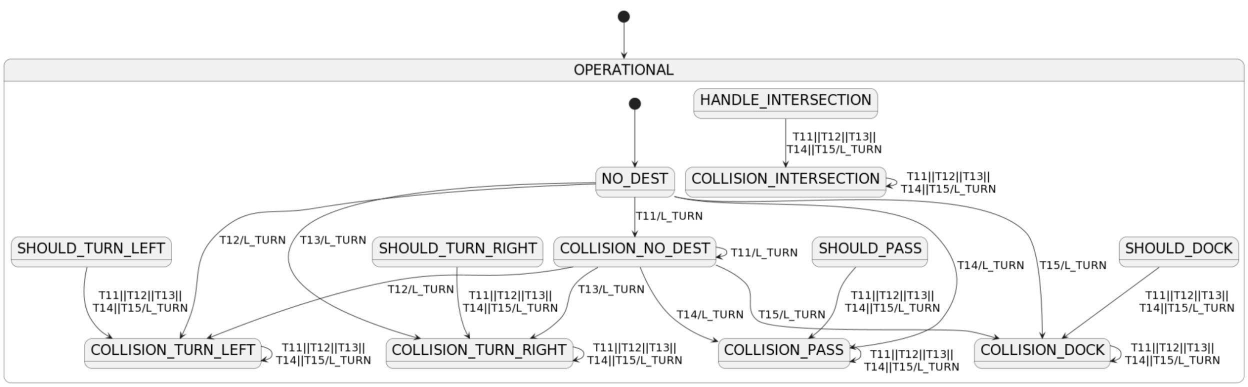


Figure 6: The proposed state machine for the system with the collision events

7.4.2 Justifications

By Bardia Parmoun

The typical flow of the system:

The robot starts at the NO_DEST state until it receives a navigation notification. Based on the navigation notification that it gets, the robot moves to one of the following states: SHOULD_DOCK for a docking, SHOULD_PASS for pass, SHOULD_TURN_LEFT for turning left, and SHOULD_TURN_RIGHT for turning right. From the SHOULD_PASS, SHOULD_TURN_LEFT, and SHOULD_TURN_RIGHT states the robot will enter the HANDLE_INTERSECTION state upon losing the wall. These transitions will be done with their proper actions: FORWARD, L_TURN, and R_TURN accordingly. During the

HANDLE_INTERSECTION, the robot will keep going FORWARD until it has found a wall. Finally, when the robot gets a wall, it will go back to the NO_DEST event to start a different intersection. Similarly, when the robot loses the wall at a SHOULD_DOCK state, the robot wil go to the DOCKED state which concludes its trip. Note that WALL_FOLLOW is an expected behaviour for the NO_DEST, SHOULD_TURN_LEFT, SHOULD_TURN_RIGHT, SHOULD_PASS, and SHOULD_DOCK states since the robot is expected to be following a wall during those states. In other words, during the states, if the robot gets a transition which has a wall, it will perform wall following. Also note that events that include losing the wall should not be expected for the NO_DEST state however in the case that it occurs the robot is instructed to turn right (in case this event was triggered by the robot getting too far from the wall accidentally). Finally, it is important to mention that at any time an event with error can lead the robot to enter NOT_OPERATIONAL with a KILL action.

Collision Handling:

In order to preserve the expected direction for the robot, a collision state has been dedicated for every state that can have a collision. Those states are as follows: COLLISION_NO_DEST, COLLISION_DOCK, COLLISION_TURN_RIGHT, COLLISION_PASS, COLLISION_INTERSECTION, and COLLISION_TURN_LEFT. The typical flow of collision handling is the same for all of these states. After the robot receives a an event with a bumper sensor, it will do a L_TURN and enter its specific collision state. Once it receives an event without the bumper, it will do a R_TURN to get back to its original direction and goes back to the state that called it. The idea is that through these constants left and right turns the robot will slowly move past that the obstacle while always maintaining its original direction and angle. Please note that this method of collision detection only works for the obstacles that are either attached to the wall on one side or are removed after the collision (ie a human). In other words, with the current design if there is empty space all around a static obstacle, the robot will constantly circle around it. This will be addressed in future designs. Finally, the current design of the robot is capable of detecting any obstacle in front of it and performing a turn accordingly. Meaning it should be capable of avoiding most obstacles that are in front of it by default.

7.5 Node Structure

By Max Curkovic, Cassidy Pacada, Matt Reid, and Bardia Parmoun

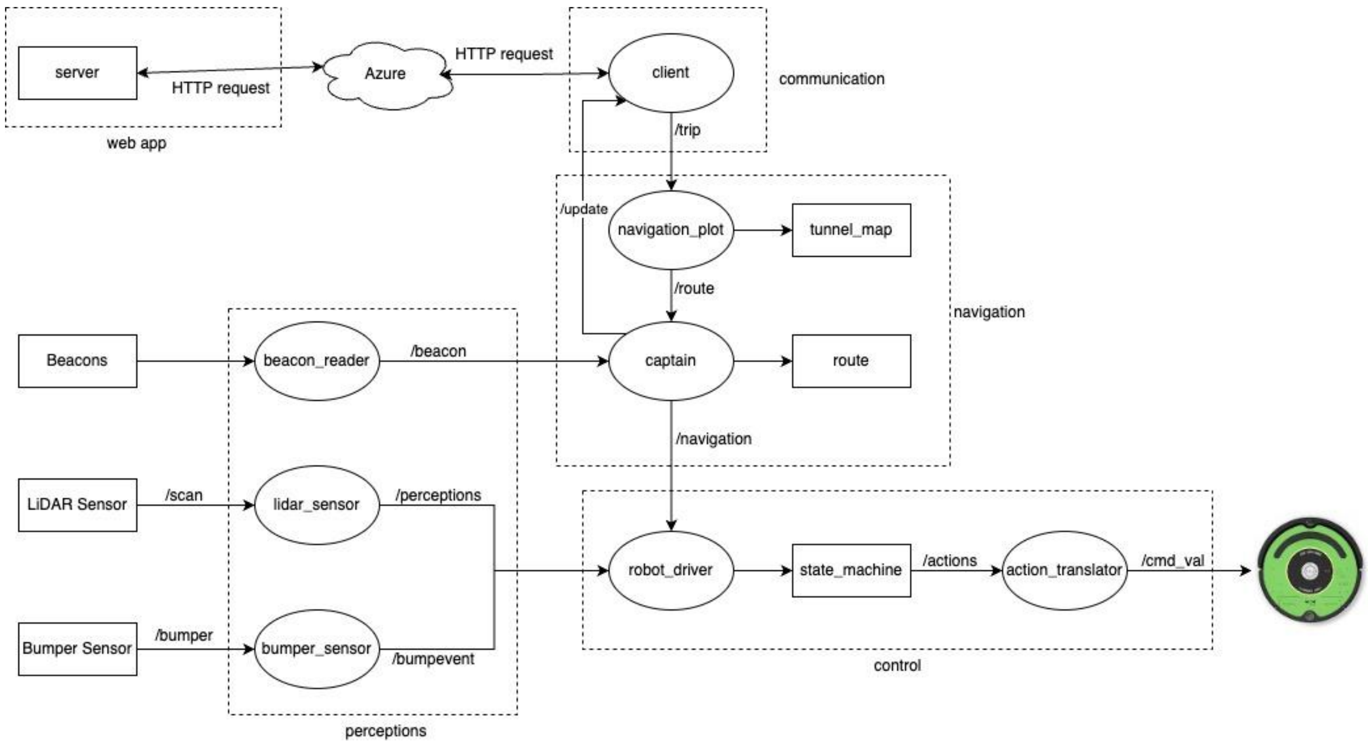


Figure 7: The proposed ROS node structure for the system.

7.5.1 List of Nodes, Messages and Entities

By Max Curkovic, Cassidy Pacada, Matt Reid, and Bardia Parmoun

Nodes:

- Server: Defines a node which describes the server (from the web application).
- Client: Defines a node which describes the client.
- Navigation_plot: Defines a node which describes the navigation plotting.
- Captain: Defines a node which describes the captain - controlling the robot en route.
- Beacon_reader: Describes a node which allows for reading the detected beacons.
- Lidar_sensor: Describes a node for the LiDAR sensor.
- Bumper_sensor: Describes a node for the bumper sensor.
- Robot_driver: Describes a node for the robot driver, which takes in state machine actions.
- Action_translator: Describes a node for the action translator, which translates state machine actions into valid serial commands.

Messages:

- /trip: Sends a trip message from the client node to the navigation_plot node.
- /route: Sends a route message from the navigation_plot node to the captain node.
- /update: Sends an update message from the captain node to the client node.
- /navigation: Sends a navigation message from the captain node to robot_driver node.
- /actions: Sends an action message from the robot_driver node to the action_translator node.
- /cmd_val: Valid command that represents a state machine action sent to the robot.
- /bump_event: Bumper sensor event message sent to the robot_driver node.
- /perceptions: LiDAR sensor event message sent to the robot_driver node.
- /beacon: Beacon reader message sent to the captain node.

Entities:

- Server: Defines the web server used in the web application.
- Tunnel_map: Defines a map of the tunnels.
- Route: Defines a route for the robot to take.
- Beacons: Defines beacons used by the robot when traveling.
- LiDAR sensor: Defines the LiDAR sensor used on the robot.
- Bumper sensor: Defines the bumper sensors used on the robot.
- State_machine: Defines the state machine used by the robot.

7.5.2 Justifications

By Max Curkovic

The typical flow:

The above figure depicts the proposed node structure, which would be detailed using ROS. Essentially, a client node would receive a request, which would then be sent through the captain during the trip action. The captain node utilizes a beacon reader node, which aids in sensing the direction of the robot, and utilizes a robot driver to assist in sensing for obstacles. The state machine entity (see Figure 3) uses an action translator which tells the robot what state to enter. The navigation_plot node utilizes a tunnel map entity to assist in the robot's proper navigation through the tunnels. The beacons assist in ensuring that the robot follows this particular route.

There are two sensor nodes that are accounted for: the LiDAR sensor and the bumper sensor. The LiDAR sensor sends perception messages to the robot driver depending on where the robot is on its path. The bumper sensor should recognize if the robot has hit a wall, and should perform the expected "handle collision" state.

8 Implementation

By Bardia Parmoun

This section describes the current state of the project along with detailed explanations for the various core features of the project along with their corresponding implementation.

8.1 Minimum Viable Product

By Max Curkovic, Cassidy Pacada, Matt Reid, and Bardia Parmoun

To evaluate the design choices outlined in the previous section, the team has decided to develop a minimum viable product which contains a subset of the robot's responsibilities. For the latest iteration of the MVP, the team has considered the following functionalities:

1. The ability for the robot to reliably wall follow using LiDAR. This includes detecting and finding a wall and maintaining the distance with the wall using the PID controller. In other words, if the robot starts off closer or further from the expected distance it should autocorrect itself to account for the set distance.
2. The ability for the robot to reliably make left and right turns. This includes recognizing an intersection, making a turn, and going back to the wall following.
3. The ability for the robot to properly detect beacons. In this iteration all the robot needs to do is to log the beacons it has seen; if possible, this behaviour can be further improved by having a test intersection and an expected beacon so the robot will use the beacon to properly identify the intersection.
4. The system should implement the state machine proposed in 1.4 and act accordingly. For this iteration, the system could bypass certain states such as docking, charging, and collision handling and instead provide stubs for them.

8.2 State Machine Implementation

By Bardia Parmoun

The state machine is core part of the robot's behaviour as such it is important to provide a proper implementation for it that closely resembles the state machine diagram. A file named "state_machine.py" was created under the control module to encapsulate the all the state behaviours. The state design pattern was utilized to implement the state machine. This allows the robot_driver class to simply includes a state parameter that holds the current state of the robot. That state parameter will include an action_publisher which is the same as the action publisher for the robot_driver node. This action_publisher will be used by the state for publishing its actions. As a result, the robot_driver can simply send updates to the current state. Upon receiving each update, the current state will publish the corresponding action and returns a state (either itself or a new state indicating a transition).

As explained in 7.4.1, the robot has various sources of perception that can be triggered at various times. As such the team decided to define events as a snapshot of these perception events. To achieve this, the robot_driver includes specific variables for each source of perception (LiDAR sensor, bumper sensor, and the beacon). These variables are constantly updated by the call backs that for each source of perception. To send an event to the state machine, the robot_driver includes a timer that will trigger every 0.1 seconds. Once the timer triggers, the value of all three variables are passed to the state machine. From there the state will use these values to figure out which transition has been triggered and it will call it.

8.3 Implementing Wall Following

By Bardia Parmoun

The previous team for this project implemented a simple PID controller that was able to maintain a specific and angle with the wall. This controller worked perfectly when the robot was close to the wall however the team noticed some issues when the robot was placed further from the wall. This was mostly due to the fact that the PID was only maintaining the distance with the wall by sending specific rotation commands for the *Twist* command in ROS. Since the PID was not aware of the current orientation of the robot, the effects of the these rotation commands would easily compound and result in robot completely missing the wall or doing a U-Turn.

To overcome this issue, the team came up with a much easier implementation that involved simply having the robot maintain a specific angle with the wall. To implement this solution, the team first finds the robot's angle and distance with the wall using the LiDAR sensor. This is achieved by reading the positive side of the sensor with a big range such as 60 to 170 degrees. The robot's angle with the wall is calculated as follows:

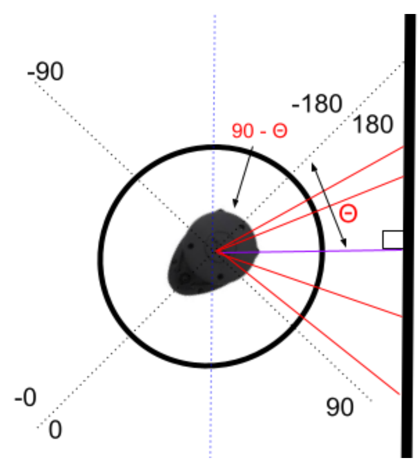


Figure 8: Calculating the robot's angle with the wall using LiDAR scan

As shown in the diagram, the program will first look through the scan to find the smallest one. That indicate the one that was perpendicular with the wall (ie in figure the scan that was marked with purple). The program will then take a look at the angle that resulted in the shortest scan, Θ, and calculates 90 - Θ. That determines the angle of the robot with he wall. This is due to the fact that the robot and the wall create a right angle triangle.

Now using this angle, the wall following behaviour can easily make the robot maintain a set angle with the wall. The main approach is to constantly calculate the robot's angle with the wall using the described method and then provide the robot with an angular velocity to correct its current angle so it is set to the preset angle. This correction process will continue until the robot reaches a certain threshold with the wall from which it will just follow the wall in parallel until it leaves that threshold and more correction is needed. Please note that to avoid constant minor adjustments around the threshold, an error region was defined on either side of the threshold as acceptable distance with the wall. This prevents the robot from doing any angle corrections in that region. That error region is calculated based on the robot's maximum speed and the set angle. The error formula is as follows:

$$error = (speed \cdot sin\,(angle) \cdot time) \,/\, 2$$

This formula allows for a small error region around the threshold. This region is equivalent to the distance that the robot will travel during a given interval. The error value is half of that area since it is being applied to both ends of the threshold.

8.4 Robot Simulator

By Bardia Parmoun

To simulate wall following and the robot's other core behaviours, the team decided to create a simple simulator. This simulator was developed using the *Turtle* library in Python.

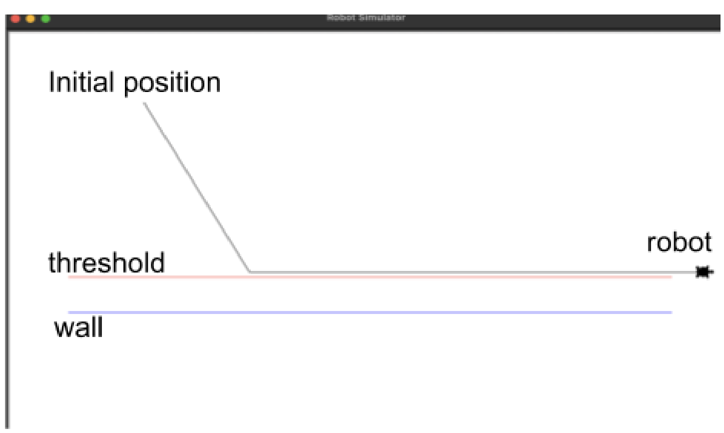


Figure 9: Simple simulator show casing the robot's wall following behaviour.

As illustrated in the diagram, the simulator allows for the robot to be initialized at a custom distance and angle with the wall. Similarly, the simulator also allows the user to set a threshold for the distance that the robot should maintain with the wall. From there the simulator will calculate the error region and give the robot the proper commands. The simulator also simulates the ROS twist command by giving appropriate linear and angular velocities to simulate its behaviour.

Since there is *ROS Gazebo* support for iRobot CREATE 3, the team is planning on implementing a more advanced simulator to properly simulate the state machine and the remaining core functionalities of the robot.

9 Testing

By Cassidy Pacada

9.1 Testing Strategy and Justification

The team's testing strategy is to individually test the action_translator, the robot_driver, and the captain by making ROS nodes that will stub the other components and provide controlled test data. This allows the team to test the program without the need to run the physical robot which is much more efficient and convenient. The team will be able to simulate sensor data that may be difficult to obtain in the real world, particularly with the amount of traffic in the tunnels. Additionally, this method of testing ensures that each component test is decoupled from the other components. This prevents issues such as test failures that are caused by other areas in the program as opposed to faults in the specified component.

The team has created a set of unit tests for the state machine. This ensures that the transition of states occurs as expected. As well, an action publisher stub was created that collects a list of every action that the state machine sends allowing the team to verify that they are correct. This was done using *colcon test* which is a package that identifies test files and allows us to run them without the need to create launch files. However, this only works when testing the functions inside each class independently and that no actual nodes are being run. To test how the Robot Driver, the Action Translator, and the Captain interact with each other and with the state machine, integration tests were made. This does involve the creation of nodes and as such a launch file must be made for every test. The integration tests are currently being limited by the inability to stop the processes as destroying the node at the end of the test is ineffective. The team is looking into ways to kill the node process directly through Linux commands.

9.2 Workflow Specification

For integration testing,the team will make launch files for each of the test nodes so that it can be opened with the *ros 2 launch* command. A bash script containing all the test commands will be created for ease of testing. Furthermore, a Github workflow will be created to automatically run the unit tests every time a pull request or a commit to main occurs. This will ensure that the program is functional at all times and is meant to keep the project aligned with the practice of continuous integration. The team is researching to determine if the integration tests can also be added to the workflow.

10 Documented Issues

By Matt Reid

This section outlines issues with the system that have been identified and need to be addressed. For each issue, a temporary solution that will be used so that development is not blocked, a permanent solution that will be implemented by the end of the year, and any other potential solution(s) for if the permanent solution does not work are provided. Section 11.1 outlines an issue with automatic undocking where the robot cannot be activated by the serial port, and Section 11.2 outlines an issue with the power system not providing enough power for the Raspberry Pi microcontroller to power the LiDAR module.

10.1 Automatic Undocking

By Matt Reid

Since the robot must dock in order to recharge between trips, it is desirable to be able to dock and undock the robot on command from the Raspberry Pi to the robot. An issue was identified where it becomes impossible to talk to the robot over the serial port when it is on the docking station. It was found that this was due to the robot sending charge information over the serial port, constantly filling the data lines while it charges.

Temporary Solution:

A temporary solution to this issue is to manually undock the robot when it is done charging and is ready to go on another trip. This solution is not ideal as the robot is no longer fully autonomous, requiring manual intervention to move it off of the docking station.

Permanent Solution:

A permanent solution that was provided by iRobot when a similar question was asked about the robot failing to receive the undock command, was to contact them by email to get an updated firmware for the robot processor [9]. This update should fix the sleep/wakeup functions so that they can be used at any time as expected.

Other Solutions:

If an update cannot be obtained from iRobot, or the firmware update is unsuccessful, the team can connect a relay to the Raspberry Pi which pushes the “Clean” button on the robot twice to undock it. By pushing the “Clean” button, the robot will undock and start moving so that it can clean. Then by pushing it again, it will stop trying to clean and will be off the dock and ready to receive commands again.

Another solution to this problem is to get the iRobot Create 3 platform discussed in Section 2.2. Since it has ROS2 built in, it does not have the communication issue and would be able to dock and undock autonomously without issue. Because of the cost of a new Create 3 robot, the team decided that this solution should not be entertained unless all other solutions fail.

10.2 Powering the entire system using the robot

By Matt Reid

Currently, the system is being powered by the serial port of the iRobot connected to a DC to DC converter. Currently, the measured output of the DC to DC converter is 5V with 1A. The recommended power input for a Raspberry Pi 4 Model B is 5V with 3A (15W), but a 2.5A supply can be used if downstream components do not exceed 500mA [10]. This means that the Pi is always underpowered by the iRobot serial port and is unable to provide enough current to power any downstream components such as the LiDAR.

Temporary Solution:

A temporary solution to this issue is to use a battery pack that can provide 5V with 2.5A since the current draw of the LIDAR will not exceed 500mA. The problem with this solution is that the power bank would either have to be manually recharged, or slowly charged by the robot during operation, reducing the robot's range.

Permanent Solution:

Another option for powering the pi from the robot is to use the main vacuum brush's motor driver power. The main motor driver on the robot is capable of providing 1.45A at 12V which gives a capacity of 17W. Using a DC to DC converter, this means that the motor driver would be able to provide the Pi with 5V and 3.4A [11]. A hardware schematic of this setup can be seen in Figure 12 in Appendix B. The problem with this solution is that there is no power to the motor driver until the robot is turned on, which requires a signal to be sent through the serial port (unless done manually). One solution for this would be to use a microcontroller that requires less power such as the Pi Zero to send a start command over the serial so that the Pi 4B receives power and takes over from there with the LIDAR. A hardware schematic of the serial port power setup for the Pi Zero can be seen in Figure 13 in Appendix B.

With the iRobot Create 3, the robot provides a 5V 3A usb-c port that can provide all of the power needed for the system [3]. This means that for future robots added to the fleet that are Create 3 instead of Create 2, this problem will be solved out of the box.

Other Solutions:

If it is not possible to get power from the motor driver, or have the Pi Zero W start the robot, another solution would be to charge the power bank through the serial port. The problem with this solution is that the Pi would be drawing more power from the power bank than the serial port can put in, causing the bank to drain and eventually die, potentially before the robot dies.

As with the undocking issue, this issue would also be solved with the iRobot Create 3 platform. As was discussed in Section 2.2, the Create 3 provides a 5V/3A usb-c output to the storage bay which would be enough to power the LiDAR module (the Pi would also no longer be necessary due to the built in ROS). If the LiDAR were connected to the usb-c port, it could communicate with the built in microcontroller and be sufficiently powered. As discussed previously, this solution will not be entertained this year due to the large added costs.

11 List of Required Components/Facilities

By Cassidy Pacada

Table 21: List of required components/facilities, their objectives of the project, and estimated cost

Required Facility / Equipment	**Purpose / Rationale**	**Estimated Cost**
Tunnel Access	Necessary to test the robot's obstacle-avoidance capabilities in its intended environment	$0
LiDAR Sensor	Will be used to improve the robot's navigational capabilities	$167
Power Bank	Will be used to power the LiDar Sensor	$35

11.1 Justification of Purchases

By Cassidy Pacada

Sensors

Table 22: Comparison of sensor candidates based on power usage, range of effectiveness for both distance and direction, and price

	Meng Jie TF-Luna	**Existing IR Sensors**	**Slamtec RPLIDAR AM18**
Power Usage	< 0.35W	Around 0.01W	0.5W
Range of Effectiveness (Distance)	Effective at a range of up to 8 meters.	Effective at a range of approximately 20 cm.	Effective at a range of up to 12 meters.
Directional Range of Effectiveness	Effective in only one direction.	Effective in only one direction per sensor.	Effective range of 360°.
Price	$56.17	Free (re-used from last year's project.)	$167

Multiple LiDAR options were considered such as the Meng Jie TF-Luna which is effective up to 8 meters and is advertised to have low power consumption. [6] This was an asset as the LiDAR draws power directly from the robot and the team did not want it to drain the battery too quickly between home bases. Additionally, the MengJie LiDAR was on the cheaper end of LiDAR options which was valuable as the team wanted to ensure the given budget was used effectively.

Another option that was considered was to retain the existing IR sensors but to add more of them around the robot for better coverage. The team already had a number of IR sensors so the added cost would be minimal. Additionally, as the previous team had been working with the sensors, this option meant that it would be possible to retain much more of the existing code.

In the end, the team chose to use the more expensive LiDAR option as its benefits outweighed the benefits of the other options. One of the major problems with the IR sensors was that they were only effective from a distance of approximately 20cm. Even if the team had more directional coverage, the robot would need to remain extremely close to the wall in order to navigate properly. This may have been an issue in the middle of wide intersections where the robot could lose the wall and become disoriented.

The MengJie LiDAR was not selected as it can only range in a single direction. Even with its 8 meter range, this would give us the same issue that the robot has with the single IR sensor where the robot would have several blind spots on its left, front, and back sides. The LiDAR that was selected has 360° capabilities, meaning that multiple purchases would not have to be purchased to obtain full coverage. As well, it has a range of 12 meters which ensures that it should be able to sense walls from the middle of intersections with no difficulty. These advantages made it the clear option for our project and justified the additional cost.

Battery Bank

Table 23: Comparison of battery pack candidates based on supplied power, size, and price

	Anker Portable Charger [7]	**Anker PowerCore 10000 Portable Charger [8]**
Supplied Power	15W	12W
Size	6.2in x 2.9in x 0.8in	3.6in x 2.3in x 0.9in
Price	$56.50	$38.42

The main candidates considered for the battery bank that would be used to power the Raspberry Pi and the LiDAR were the Anker Portable Charger and the Anker PowerCore 10000

Portable Charger. The biggest determining factor for our decision to choose the Anker PowerCore was that it was smaller than the Anker Portable Charger. The robot has a small surface area and a lot of the space is already being taken up by the Raspberry Pi, the LiDAR, and various wires. Although a chassis has been designed to hold everything in place, the team prefers to preserve as much space as possible so the robot will have space for the mail holder.

Even though the Anker Portable Charger supplies more power than the Anker PowerCore, the team calculated the amount of power drawn by the Raspberry Pi and the LiDAR and determined that a supply of 12W was sufficient. A secondary benefit to the Anker PowerCore is that it is the cheaper option which allows the team to save the budget for any other supplies that may be needed.

12 References

[1] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, W. Woodall, “Robot Operating System 2: Design, architecture, and uses in the wild,” Science Robotics vol. 7, May 2022. (accessed Sep. 22, 2023).

[2] “Create 2 Robot: What We Offer” edu.irobot.com. https://edu.irobot.com/what-we-offer/create-robot (accessed Sep. 22, 2023).

[3] ”Overview - Create 3 Docs” edu.irobot.com. https://iroboteducation.github.io/create3_docs/hw/overview/ (accessed Sep. 22, 2023).

[4] “What is LiDAR?” Synopsys. https://www.synopsys.com/glossary/what-is-lidar.html (accessed Sep. 30, 2023).

[5] “Slamtec RPLIDAR A1 Low Cost 360 Degree Laser Range Scanner. Introduction and Data Sheet” Slamtec. https://bucket-download.slamtec.com/d1e428e7efbdcd65a8ea111061794fb8d4ccd3a0/LD108_SLAMTEC_rplidar_datasheet_A1M8_v3.0_en.pdf (accessed Oct. 8, 2023).

[6] “Meng Jie TF-Luna small size and Low Power, lidar range finder sensor module, single-point micro ranging module, ranging from 0.2-8m, Resolution 1cm, compatible with UART / I2C communication interface,” Amazon.ca: Electronics, https://www.amazon.ca/Single-Point-Resolution-Compatible-Communication-Interface/dp/B09DCLPYV1/ref=sr_1_18?keywords=lidar&qid=1696789560&s=electronics&sr=1-18 (accessed Oct. 8, 2023).

[7] “Anker Portable Charger, Power Bank, 20K” Amazon.ca: Electronics, https://www.amazon.ca/Anker-PowerCore-Technology-High-Capacity-Compatible/dp/B07S829LBX/ref=sr_1_6?crid=1Z7F36BX965BO&keywords=anker%2Bpower%2Bbank%2B5v%2B3a&qid=1697405645&sprefix=anker%2Bpower%2Bbank%2B5v%2B3a%2Caps%2C85&sr=8-6&th=1 (accessed Oct. 15, 2023).

[8] “Anker powercore 10000 Portable Charger,” Amazon.ca: Electronics, https://www.amazon.ca/Anker-PowerCore-Ultra-Compact-High-speed-Technology/dp/B0194WDVHI/ref=sr_1_29?crid=AKV3WJBMCKR8&keywords=anker%2Bpower%2Bbank%2B5V%2F2.4A&qid=1697245268&sprefix=anker%2Bpower%2Bbank%2B5v%2F2%2B4a%2Caps%2C73&sr=8-29&th=1 (accessed Oct. 15, 2023).

[9] “Create 2 losing serial communication after toggling full to passive while charging” Robotics Stack Exchange. https://robotics.stackexchange.com/questions/15433/create-2-losing-serial-communication-after-toggling-full-to-passive-while-chargi (accessed Oct. 27, 2023)

[10] “Raspberry Pi 4 Tech Specs” Raspberry Pi. https://www.raspberrypi.com/products/raspberry-pi-4-model-b/specifications/ (accessed Oct. 15, 2023)

[11] “Battery Power from Create 2” iRobot Education. https://edu.irobot.com/learning-library/battery-power-from-create-2. (accessed Oct. 15, 2023)

Appendix A: Chassis Design

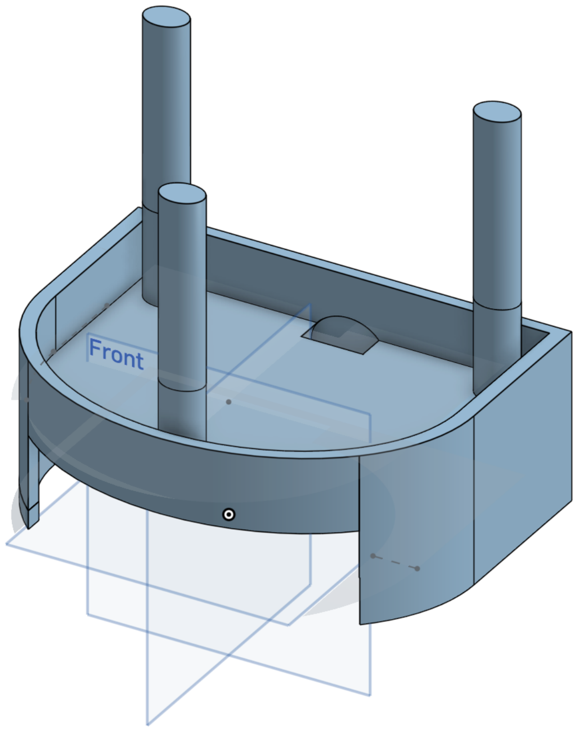


Figure 10: The chassis design for the robot holding the circuits for the robot.

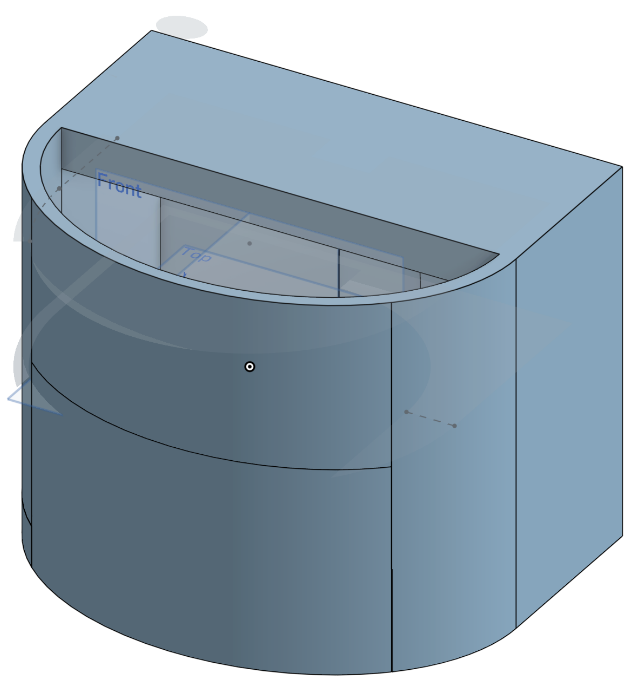


Figure 11: The design of the mailbox for the robot.

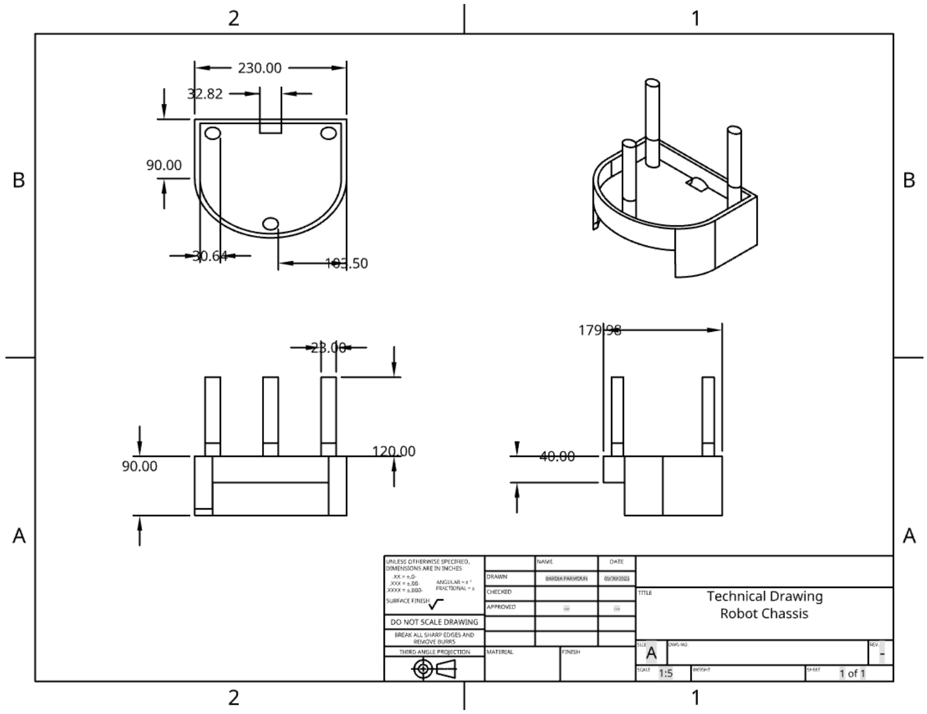


Figure 12: The technical drawing for the robot chassis detailing its measurements.

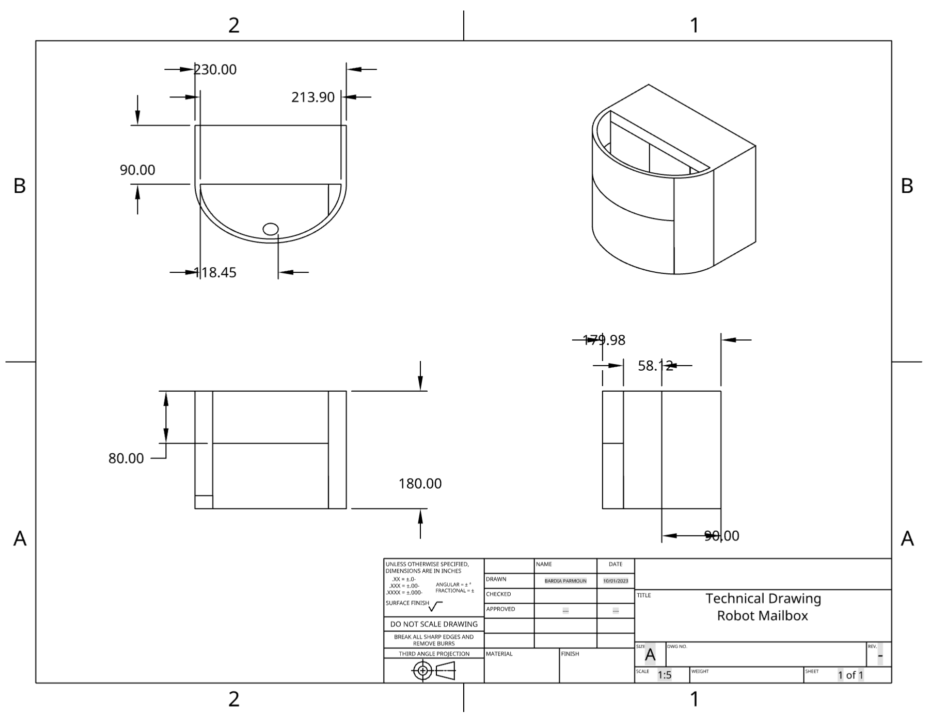


Figure 13: The technical drawing for the mailbox detailing its measurements.

Appendix B: Hardware Setup

Setting up the hardware for CREATE 2

Connecting the RPLIDAR A1M8 LIDAR to the Raspberry Pi 4B

The LIDAR can be connected to the Raspberry Pi 4B using either a USB to TTL adapter or by connecting it to the GPIO pins of the Pi. The following steps will provide a guide on how to connect to the GPIO pins on the Pi:

1. Connect the 5V motor and 5V Vcc pins on the LiDAR to the 5v power from the Pi
2. Connect the ground pins for the motor and main LiDAR ground to the ground of the Pi
3. Connect Tx of the LiDAR to Rx of the Pi which is on GPIO pin 14
4. Connect Tx of the Pi which is on GPIO pin 15 to the Rx of the LiDAR
5. Connect the motor control pin of the LiDAR to any GPIO pin on the Pi (in this case pin 25 is used)

The hardware schematic of this implementation is shown in Figure 13 below.

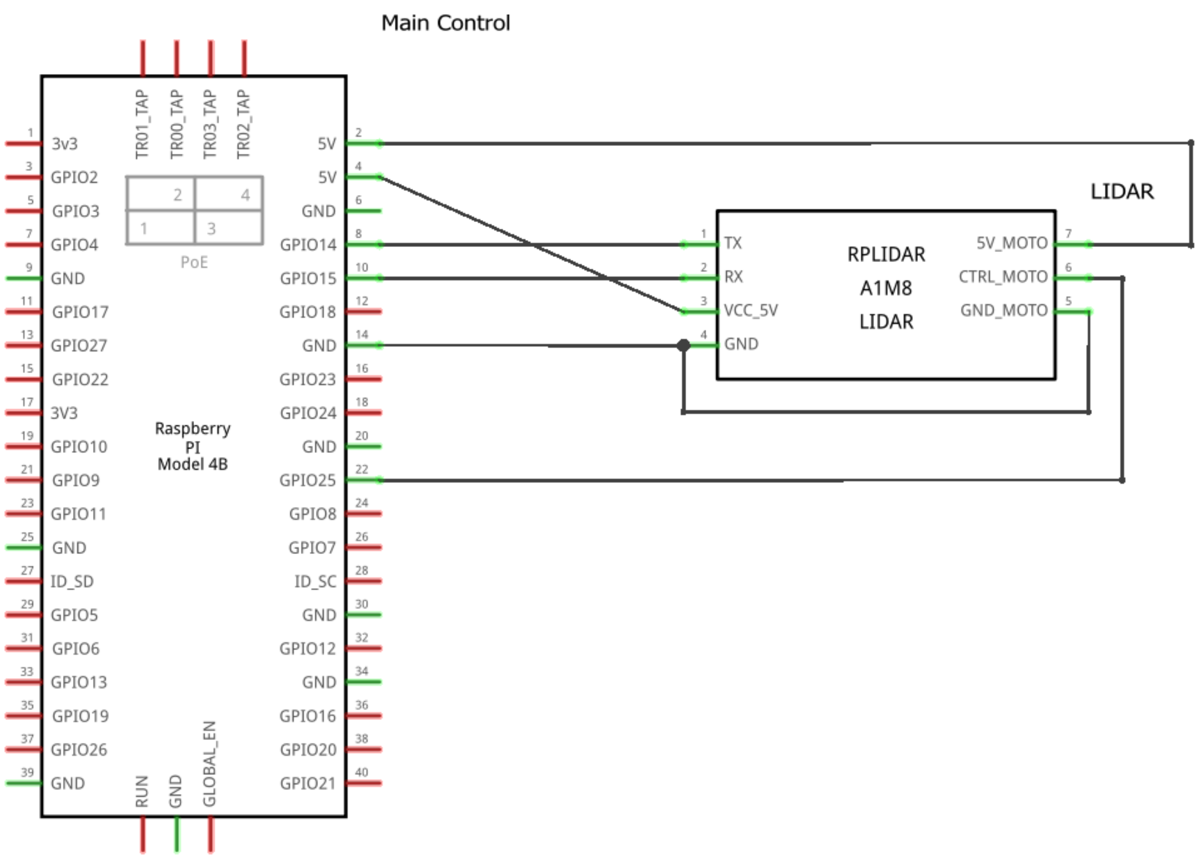


Figure 14: Hardware Schematic for the LiDAR connection

Connecting the Raspberry Pi 4B to the iRobot Create 2 Main Brush Motor Driver

As discussed in Section 10.2, in order to have enough power to run the project code as well as powering the LIDAR device, the Raspberry Pi 4B needs to be connected to the main brush motor driver on the iRobot Create 2 (providing 5V 3A to the Pi). This will require a 2.2mH, 1.4A inductor and a 17W DC-DC converter. The steps to connect the motor driver power to the Raspberry Pi 4B are as follows:

1. Remove the main vacuum brush and motor from the iRobot Create 2
2. Connect the red wire to a 2.2mH, 1.4A inductor and then to the positive input of the DC-DC converter
3. Connect the black ground wire to the negative input of the DC-DC Converter
4. Cut one end of a USB C cable and connect the ground to the negative output and the power to the positive output of the DC-DC Converter
5. Connect the USB C cable to the Raspberry Pi 4B
6. Start the robot to power the main brush driver and start the Raspberry Pi 4B

A hardware schematic of this implementation can be seen in Figure 14 below, and pictures of the real hardware connection on the iRobot Create 2 can be seen in Figure 15.

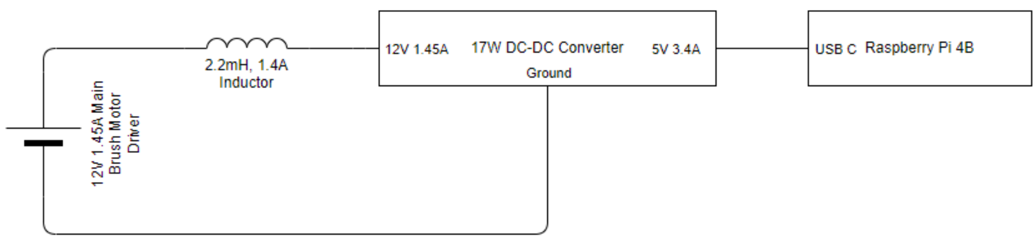


Figure 15: Hardware Schematic for Main Brush Motor Driver power supply to the Pi 4B

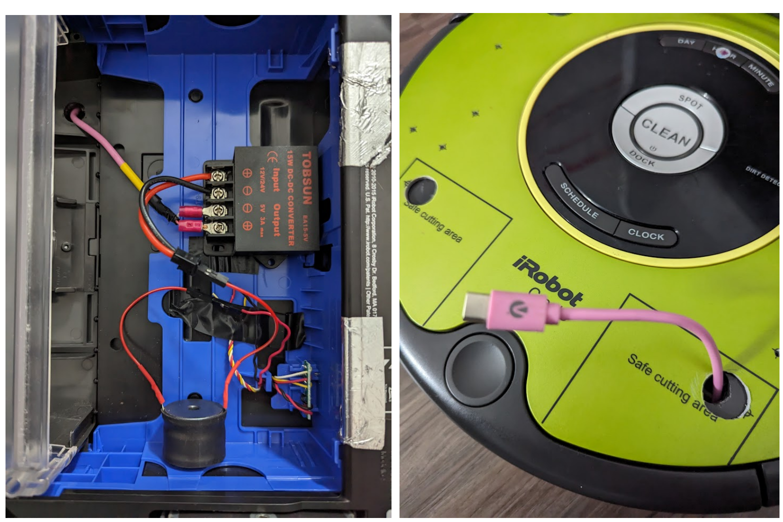


Figure 16: Hardware Connection for the Main Brush Motor Driver power to USB C output

Connecting the Raspberry Pi Zero to the iRobot Create 2 Serial Port Power Source

As discussed in Section 10.2, a Pi Zero will also be required to start the motor driver and power the main Raspberry Pi 4B. Since a Pi Zero requires much less power than the Raspberry Pi 4B (5V 1A instead of 5V 3A) it can be directly connected to the serial output of the iRobot Create 2. This setup will only require a micro USB cable and a 5W DC-DC converter. The steps to connect the hardware are as follows:

1. Connect the positive and negative inputs of the DC-DC converter to the positive battery terminal and negative battery terminal pins of the serial connector
2. Cut a micro usb cable and connect the ground to the negative output of the DC-DC converter, and the power to the positive output
3. Connect the micro USB to the Pi Zero
4. The Pi Zero will be powered as long as the robot battery has power

The hardware schematic of this implementation is shown in Figure 15 below.

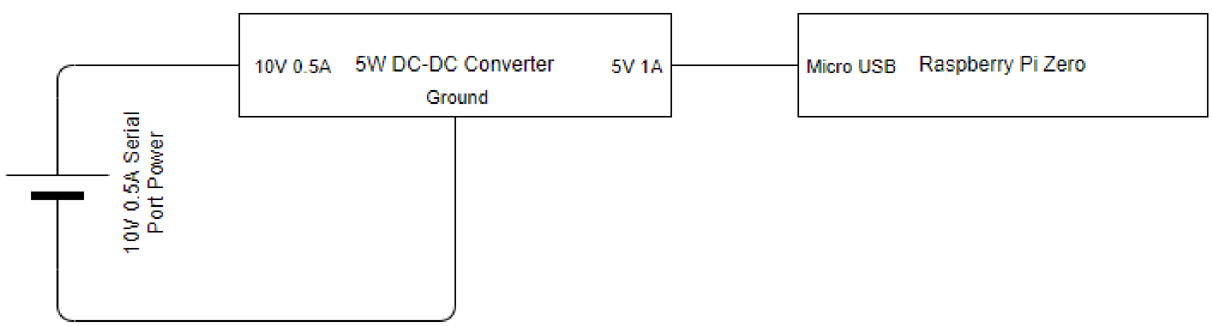


Figure 17: Hardware Schematic for Serial Port power supply to the Pi Zero

Appendix C: Software Setup

Setting up the Project for CREATE 2

Installing the Ubuntu Image

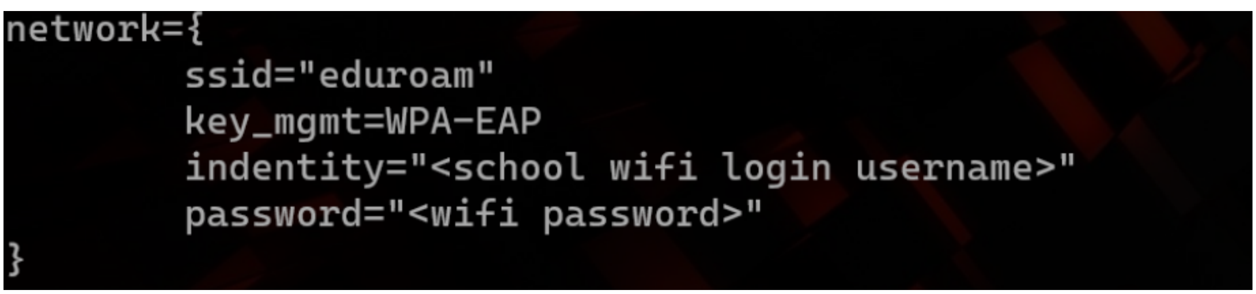
1. Using the Raspberry Pi Imager, install Ubuntu Server 20.04.x
2. SSH is already enabled, login with user:ubuntu, password:ubuntu and change the password.

Creating an account on the image

1. ***sudo su - root***
2. ***hostnamectl set-hostname***
3. ***adduser robot***
4. ***su - robot***

Setting up Wifi

1. Install wpasupplicant using: ***sudo apt install wpasupplicant***
2. Initialize the wpasupplicant file using: ***wpa_passphrase your-ESSID your-wifi-passphrase | sudo tee /etc/wpa_supplicant.conf***
3. Edit the wpasupplicnt file using: ***sudo /etc/wpa_supplicant.conf***



```
network={
        ssid="eduroam"
        key_mgmt=WPA-EAP
        indentity="<school wifi login username>"
        password="<wifi password>"
}
```

Figure 18: The contents of the wpa_supplicant file.

Configuring ROS

1. Install ROS Foxy by following this link: https://docs.ros.org/en/foxy/Installation/Ubuntu-Install-Debians.html
2. Run ***sudo rosdep init***
3. Run ***source /opt/ros/foxy/setup.bash*** (must be run with every new terminal)

Creating a local codespace and clone all the necessary code

1. Run ***mkdir -p ~/cmds_ws/src***
2. Run ***cd ~/cmds_ws/src***
3. Run ***git clone git@github.com:bardia-p/carleton-mail-delivery-robot.git***
4. Follow this link to the get create driver: https://github.com/AutonomyLab/create_robot/tree/foxy
5. Get the LiDAR package by running: ***git clone git@github.com:Slamtec/sllidar_ros2.git***
6. Install bluepy by running: ***sudo apt install bluepy***
7. Run ***cd ~/cmds_ws***
8. Run ***colcon build***

Running the code

1. Run ***sudo -s***
2. Run ***cd ~/cmds_ws***
3. Run ***source /opt/ros/foxy/setup.bash***
4. Run ***source /home/ubuntu/cmds_ws/install/setup.bash***
5. Run ***ros2 launch mail_delivery_robot robot.launch.py 'robot_model:=CREATE_2'***

Running the unit tests

1. Run ***cd ~/cmds_ws***
2. Run ***colcon test --packages-select mail_delivery_robot --event-handlers console_cohesion+***

Running the integration tests

1. Run ***sudo -s***
2. Run ***cd ~/cmds_ws***
3. Run ***source /opt/ros/foxy/setup.bash***
4. Run ***source /home/ubuntu/cmds_ws/install/setup.bash***
6. Run ***ros2 launch mail_delivery_robot [testfile].launch.py***