Autonomous Mail Delivery Robot

By

Chase Scott, Emmitt Luhning, Favour Olotu, Jacob Charpentier

Supervisor: Dr. Babak Esfandiari

A report submitted in partial fulfilment of the requirements

of SYSC-4907 Engineering Project

Department of Systems and Computer Engineering

Faculty of Engineering

Carleton University

April 12, 2023

Abstract

By Chase Scott and Favour Olotu

The autonomous mail delivery robot system was developed to streamline the transportation of mail across Carleton University's campus through the tunnel system. The system is intended to operate without human intervention and aims to enhance the schools current solution for transporting mail between professors. The robots consist of an iRobot toolkit and a Raspberry Pi 4 equipped with IR sensors to detect obstacles in their surroundings. As the robots operate underground, they cannot rely on GPS for navigation. To address this, Bluetooth beacons were strategically placed in the tunnel system to construct a graph representation of the tunnel map. By sensing the beacons, the robots can estimate their location, enabling them to navigate effectively. This cost-effective and efficient system presents an excellent opportunity to replace Carleton's existing mail delivery system. The report below provides a comprehensive overview of the system's design process, implementation details, and recommendations for future improvements.

Table of Contents

List of Figures

List of Tables

1.0 Introduction

By Chase Scott and Favour Olotu

The project aims to develop an efficient mail delivery system through the use of robots that will distribute mail across the Carleton tunnel system. The proposed system will provide users with the flexibility to select their desired station for mail delivery and track the delivery process through a user-friendly web application. To facilitate the mail delivery process, the robots will operate autonomously, leveraging Bluetooth beacons to navigate between stations and communicate their status to the central server. By introducing mail delivery robots that can seamlessly navigate through the Carleton tunnel system, this project aims to optimize the mail delivery process, offering a reliable and efficient solution for the transportation of mail.

The proposed Mail Delivery Robot system offers a promising solution for streamlining mail delivery within the Carleton tunnel system. However, several challenges exist in developing a reliable and efficient system that can meet the diverse needs of users. One key challenge is ensuring the robots can navigate through the complex tunnel system and locate the correct destination stations without error. Additionally, the system must be scalable and adaptable to accommodate future growth and changes in user requirements. To address these challenges, this project draws on related work in robotics and software design, while also developing novel approaches in several key areas to ensure the system's reliability, scalability, and consistency. Technical details on the proposed solution will be discussed in Section C and relevant references will be cited throughout the document.

The problem being addressed in this project is the development of a reliable and efficient mail delivery system using autonomous robots that can navigate through the Carleton tunnel system without GPS services. Specifically, the system needs to handle traversal between points within the tunnel system, allowing the robot to move from point A to B with minimal human interaction. To meet this goal, the system must employ a sophisticated navigation system and wall following algorithms that enable the robot to operate autonomously while ensuring safe and efficient delivery of mail. The proposed solution will empower users to request services and mail pickup through a user-friendly web app, while the robots will handle the complex task of navigating the tunnels, ultimately optimizing the mail delivery process within the Carleton tunnel system. We have primarily focused our efforts on creating a robust navigation system, which can handle different intersections and pathways fully autonomously.

To address the lack of GPS services within the tunnels, the system will employ a checkpoint system implemented with Bluetooth beacons, as outlined in section 5.2 on navigation design. The robots will use these beacons to determine their location within the tunnel system and navigate to their desired destination. The ability for the robots to traverse between points within the tunnels will be accomplished by following walls between intersections and points along a path, as described in section 5.4 on wall following. This approach ensures that the robots can move safely and efficiently through the complex, obstacle-rich tunnel environment.

Our team successfully designed and implemented an autonomous robot that is capable of wall following, navigation, and intersection handling. The robot's wall following capabilities were improved through the implementation of a control element, as described in section 5.4. We

also made significant progress in the robot's ability to navigate through tunnels using Bluetooth beacons, which is detailed in section 5.2. The robot was able to effectively interact with Bluetooth beacons at checkpoints, as outlined in section 5.3. These functionalities were all designed, implemented, and thoroughly tested, as described in their corresponding sections within the report. While we fell short of achieving full autonomy due to hardware issues, which are documented in section 7, we made considerable progress in this regard. Our team was unable to complete the web application functionality, as our priority was achieving full autonomy for the robot.

The following report is the final deliverable for the fourth year engineering project, and the culmination of all of our hard work this semester. It documents the engineering process for the mail delivery robot, a multi-year initiative that has made significant progress over the previous academic year. The report includes several sections, each providing important information for the project. Section 2 gives an overview of the team members' qualifications and project management concepts that contributed to the project's organization. Section 3 documents the analysis of the previous year's work on the final product. In section 4, the functional and nonfunctional requirements of the system are described. Section 5 details the designs and implementations the team developed to meet these requirements. Section 6 outlines the testing of the implemented designs. The recommendations and reflections upon the project's conclusion are contained in section 7. Finally, the appendices provide information on the setup and configuration of the robot's hardware and software, which we highly recommend groups inheriting the project study closely.

2.0 The Engineering Project

2.1 Health and Safety

By Chase Scott

Since our experiments and testing were conducted in the Carleton tunnels, it was important to ensure that there was no risk to public safety. We ensured that anyone passing by our experimentation areas were notified to stand clear of the robot if it was operating, or it was moved out of the way of any passersby if possible. We also performed most of our experimental runs in enclosed locations if possible, to avoid risk of any impact to the public. We also made sure in accordance with 6.3 of the Health and Safety Guide that any material we used was cleaned up after, and that access to any exits and emergency equipment was not blocked [8].

In the case of any electric soldering that was done, masks were worn by all group members to avoid exposure to flux fumes. We also ensured that proper protective clothing was worn during soldering, including eye protection, and a vice was used to hold work in order to avoid burns. Finally in accordance with section 6.12 of the Health and Safety Guide, all exposed electronics were only handled or repaired by us and other trained, qualified personnel to ensure the safety of everyone involved [8].

2.2 Engineering Professionalism

By Chase Scott

ECOR 4995 is designed to equip students with the knowledge and skills necessary to succeed as professional engineers. The course covers topics such as engineering ethics,

communication skills, teamwork, project management, and professionalism. We endeavoured to to uphold the following principles over the duration of the project:

- Adhering to ethical principles, such as ensuring safety as detailed in section 2.1, and respecting intellectual property rights through proper references.
- Demonstrating effective communication skills, such as writing clear and concise technical reports, presenting our work to our supervisor in a polite and informative manner, and collaborating with team members in all aspects of development.
- Working effectively as a team, such as dividing tasks, setting goals, resolving conflicts, and providing constructive feedback.
- Applying project management skills, such as planning, scheduling, monitoring progress, and adapting to changes as detailed in section 2.3.4.
- Demonstrating professionalism, such as dressing appropriately, being punctual, respecting deadlines, and maintaining a positive attitude.

2.3 Project Management

By Chase Scott

Due to the size of this project, we employed several different project management techniques as detailed throughout this section in order to ensure everything was completely smoothly and without delay.

2.3.1 Timeline

By Chase Scott

Here is our timeline for the project. We initially had planned to subdivide our team into developing the web app, but development on the robot required too much attention so these features focused on instead.

Table 1: Project Objectives and Proposed Completion Dates

Objective	**Earliest Finish (EF)**
1. Robot movement prompted by project code.	September 27th, 2022
2. Autonomous movement without obstacles	October 19th, 2022
3. Proposal	October 21st, 2022
4. Wall following code corrected	November 2nd, 2022
5. Full unit test suite introduced	November 23rd, 2022
6. New State machine introduced	November 29th, 2022
7. Progress report	December 9th, 2022
~~8. Signal from app to robot - start/stop~~	~~December 22nd, 2022~~
9. PID Controller designed and integrated	January 18th, 2023
~~10. Robot location map UI completed~~	~~January 18th, 2023~~
11. Oral Presentation	Jan 24th, 2023
12. Autonomous navigation using wall following and beacons.	March 10th, 2023
13. Poster Fair	March 17th, 2023
14. Final Report	April 12th, 2023

2.3.2 Required Equipment

By Chase Scott

The list of equipment required for the additions and improvements we plan to make are as follows:

Table 2: Required Facilities and Equipment Costs

Required facility/equipment	**Purpose/Rationale**	**Estimated Cost**
Tunnel Access	Necessary for tests of intersection and obstacle avoidance	$0
Breadboards/Electrical Equipment	Replacement of unsatisfactory circuitry	$80-$100
Soldering Equipment	Affixing circuitry to prevent damage and loss + ensure effective operation	$30-$50
IR Sensors	Increase effective range of wall following behaviour.	$135

2.3.2 Risks and Mitigation Strategies

By Emmitt Luhning and Favour Olotu

Strategies we developed for dealing with identified risks are detailed in the following table:

Table 3: Risk and Mitigation Strategies

Project Risk	**Mitigation Strategy**
The inconsistent behaviour from the IR sensors presents a risk to the existing project implementation as it can cause the robot to veer off course at any time.	The types of sensing data utilized to complete a given task will be diversified such that the failure of one sense does not impede the system as a whole, as well as contingency plans to handle the situation should multiple sensory failures occur.
The robot's inability to determine whether or not it is docked may pose a threat to the ability of the robots to recharge and disembark autonomously.	The team will introduce a check to determine if the robot is docked before attempting to launch. In the event of being docked, control will be handed over to the IRobot's default docking and undocking behavior to prevent interference from the project code.
Potential failure of equipment is a threat to the timely completion of the project, should it not be promptly replaced and reconfigured.	A portion of the budget shall be set aside for the replacement of any vital parts, and all components will be developed with ease of setup accounted for, such that no part of the project should depend on one specific piece of hardware.

Losing important portions of the code, or corrupting the code base with broken/ineffective code has the potential to bring development to a halt	The project team intends to utilize GitHub for version control to prevent lost code, and have enacted strict review requirements for all new merged pieces of code to ensure all new functionality is as intended.
Most software assets that are required to build a high quality "observation" for autonomous development require high computational power that may be far above the power of the current on-board computer (Raspberry Pi).	The project team intends to keep software solutions simple to reduce dependency on high processing power.

2.3.4 Development Methodology

By Emmitt Luhning

In order to maximize the efficiency of the development process, the Agile methodology was selected as a way to manage project activities. To this end, the team has employed the use of the GitHub project board for task management, where tasks are self-assigned and moved to the review slot upon completion or the blocked slot if there is a block in development. Each work product must be reviewed by at least one other group member before a pull request to the repository can be made. In order to minimize blocks, weekly meetings were held to assess the current progress of the project and determine the next steps. This meeting is where new actionable items will be created and added to the board, whereupon the process will repeat. The choice to structure the project in this manner was to emphasize iterative development and ensure

all team members are involved in the development of and knowledgeable on all aspects of the project.

2.4 Justification of Suitability for Degree Program

By Jacob Charpentier and Emmitt Luhning

The code previously created for this project was written in Python, as is ROS, the library which facilitates communication between the project code and the robot. Therefore, the team must be able to read existing python code and write new functionality with a high degree of proficiency. Project team members have completed courses requiring Python programming, such as Introduction to Software Development (SYSC 1005) and, in some cases, Algorithms and Data structures (SYSC 2100). Additionally, all members have been educated in applicable programming best practices and principles from a language-agnostic perspective through completing courses at Carleton such as Programming Languages (SYSC 3101) and the Software Design Project (SYSC 3110).

This project utilizes the Raspberry Pi, running a Linux distribution called Ubuntu 20.04. As much of the development for this project requires configuring these devices correctly and the management of the local version of the code base on them through an ssh connection, or other remote connection techniques, it is necessary that team members have a robust knowledge of the operating system, and how their code interacts with it, as well as the required command line tools for navigation and file system manipulation. Enrollment in Operating Systems (SYSC 4001) provided members with the knowledge required in this domain to complete this project effectively.

The robotics-oriented nature of this project indicates a high degree of knowledge concerning the boundaries and communication between software and hardware will be a necessity in its completion, as well as circuit design and electronic principles. Understanding the physical effects of running code and how the team's software deals with and controls analog input and output will require the knowledge base that the team acquired through taking courses in Real-Time Systems (SYSC 3310) and Computer Organization and Architecture (SYSC 2320). Understanding of circuitry and electronic engineering has been acquired through Computer Systems Engineering courses in Circuits and Signals (ELEC 2501), Electronics 1 (ELEC 2507) and Computer Systems Development Project (SYSC 3010).

Finally, member's completion of Requirements Engineering and Software Project Management courses (SYSC 3120, and SYSC 4106 respectively) have provided the team with the skills necessary to define the requirements for this project, identify risks, as well as construct a sound budget and time table.

2.5 Individual Contributions

2.5.1 Project Contributions

Chase Scott: Created requirements and use-case diagrams, ran integration tests every system that was developed, designed and implemented PID algorithm for wall-following, and created simulator for testing and analysis. Created genetic algorithm for finding PID constants. Created poster for poster fair and slides for presentation. Worked on state machine implementation and integration, and created unit tests. Ran tests for initial analysis of IR sensors, beacons, and wall following.

Emmitt Luhning: Helped to formulate requirements. Created unit tests for existing ActionTranslator node, Implemented and tested state machine for intersection handling, and beacon detection system, Ran integration tests, and refactored captain and navigation nodes. Created video for poster-fair and slides for presentation. Configured and maintained environments on the onboard Raspberry Pi's. Ran tests for initial analysis of IR sensors, beacons, and wall following.

Jacob Charpentier: Created state machine design. Ran testing for beacon signal strength. Created testing for magicNumbers implementation. Implemented and tested state machine for wall following, intersection handling, and beacon handling. Created the hardware circuit design and implementation. Designed and printed the chassis for the robot. Created an improved beacon layout and implemented the code.

Favour Olotu: Performed analysis of the pre-existing implementations of the captain, plot navigation and bluetooth sensor ROS nodes coupled with the pathfinding algorithm. Created unit tests for the ROS nodes to ensure the channel of communication is as described. Adapted the previous year's recommendation of the navigation design, Implemented and tested the updated navigation sub-system as a unit.

2.5.2 Report Contributions

Contributions to each report section are given by the subheading beneath each section title. For contributions to actual work deliverables, see above section.

3.0 Analysis

By Chase Scott

This section of our report serves as a critical component in the evaluation of our project. In this chapter, we present a detailed and comprehensive analysis of the data collected during our project's initialization phase as we investigated the current state of development. The purpose of the analysis is to draw meaningful insights and conclusions that will inform our recommendations for future action, including requirement elicitation and design.

3.1 Wall Following Functionality

By Chase Scott

When analyzing the previous year's wall following implementation, their documentation stated that upon activation of the robot's launch command, it should enter wall following mode automatically and attempt to follow any wall it spots. This behaviour was accomplished by using two IR Sensors, the primary sensors guiding the general movement of the robot. These sensors are located on top of the robot, affixed at a 30-degree angle from each other, and point at the wall. This data is then sent to the system by the robot state machine.

3.1.1 IR Sensor Functionality

By Chase Scott

In order to ascertain the effectiveness of the system as is, we ran a series of tests on the robot to determine how well they performed their primary directive. Through the wall following tests, it was apparent that the consistency and reliability of the IR sensors were of great concern. Firstly, it was necessary to know the distance the robot could be from a wall and still detect it as present. To test this, the robot was activated for a series of runs. For each run, the distance was

altered, and if the robot could follow the wall, it could be determined that the IR sensor could find that wall at the given distance. The results of these tests are as follows:

Table 4: IR Sensor Test Results

Distance	Signal
Less than 10 centimeters	No signal
10 centimeters - 15 centimeters (inclusive)	Signal
Greater than 15 centimeters	No Signal

Secondly, it was imperative to analyse the consistency of the sensors. To do this, 10 more runs were completed with the sensors within the determined distance between 10 and 15 centimetres from the wall. Of these tests, in 3 of 10, the sensors maintained sight of the wall, and the wall following operated normally for the duration of the test. However, for the remaining 7 runs, the wall following was disrupted at one point due to the lost wall. It could be determined through analysis of the output offered by the program that the ability of the sensors to maintain signal throughout the entirety of the execution was not sufficient enough to satisfy our consistency and reliability requirements. They were constantly skewing the robot's understanding of its angle and distance from the wall, making for unreliable and erratic traversal.

3.1.2 IR Sensor Calculations

By Chase Scott

The issue of the consistency and reliability of the IR sensors is only exacerbated by the implementation of the IR Sensor calculations in the current project code. The current

implementation has the IR Sensors set up in an isosceles triangle formation, where the ideal angle between the robot and the wall is 90 degrees. Figure 1 below shows the current setup:

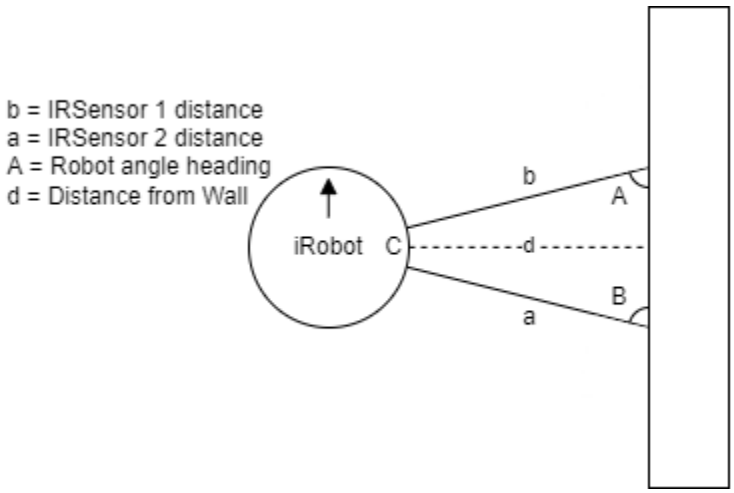

Figure 1: Positioning of the IR sensor on the robot relative to the wall

Current calculations implemented tries to achieve an angle of A between 60 degrees and 120 degrees. This range is far too large to ensure smooth movement of the robot since no action will be taken unless the robot's heading is outside this range. The current implementation also aims for a range between 10cm and 20cm from the wall, which seems to be the optimal operation range of the robot based on our experiments. The assumption of an isosceles triangle is inaccurate, as it breaks down when the robot is not moving perpendicular to the wall. Therefore, these calculations must be altered to work with a scalene triangle to return the correct value. However, the main issue with the current implementation is the lack of any control theory in the system. Without a feedback loop, the robot cannot properly react to differences in the robot's position, which leads to stuttering in the wall following behaviour. Because the robot updates to hard-coded movement operations when different flags are triggered, the robot experiences oscillatory behaviour when traversing walls which causes a significant increase in travel times

between locations. This analysis will be the main motivation into the design conducted in section 5.4.

3.2 Speed Analysis

By Favour Olotu, Emmitt Luhning, and Chase Scott

Due to the wall following analysis yielding less than ideal results, it was necessary to determine the limitations of the hardware, specifically with respect to speed. To test the speed of the existing iRobot Create 2, the robot was set to follow a wall for a set distance for five timed runs so that an average speed could be determined. The speed of the robot without wall-following enabled was also recorded to determine the absolute maximum of the hardware so that the disparity between regular operation and wall-following could be determined. Each run was conducted for a distance of 4.04 meters, with the robot starting at 15 centimeters from the wall.

Table 5: Measured Speed and Maximum Robot Speed (Not Wall Following)

Run	**1**	**2**	**3**	**4**	**5**	**6 (Not Wall following)**
Time (seconds)	20.46	21.28	20.56	20.53	20.14	12.74
Speed (meters / second)	0.1975	0.1898	0.1965	0.1968	0.2006	0.3171

Calculated Average Speed Across All Runs:

$$speed_{avg} = \frac{\sum_{0}^{5} speed_i}{N}$$

$$speed_{avg} = 0.196 \text{ m/s}$$

Calculated Standard Deviation:

$$\sigma = \sqrt{\frac{\sum(speed_i - speed_{avg})^2}{N}} = 0.004\, m/s$$

Calculated Deviation from Maximum Speed:

$$deviation\ from\ max = \frac{|speed_{max} - speed_{avg}|}{\sqrt{2}} = 0.085\, m/s$$

In conclusion, the speed analysis has shown that while the robot travels in a straight line, it has an average speed of 0.196 m/s with a threshold of +/- 0.004 m/s. This knowledge of the average speed of the robot will be useful when devising requirements for our system as we work towards improving the wall following behaviour.

3.3 Beacon Analysis

By Chase Scott

The Bluetooth beacons used for navigation are an imperative part of the operational system, and thus it was necessary to understand the operational range. To determine this, a series of tests were conducted with the beacons at several distances from the robot. At each distance, the signal strength was measured by recording the Received Signal Strength Indicator (RSSI) the raspberry pi detected onboard the robot. Each beacon was tested without obstructions between the beacon and pi, and with people blocking the signal by standing between them. Graphs were then constructed to represent the average signal strength across the distances that our robot will commonly interact with the beacon. These graphs are shown in figures 2 - 6 below:

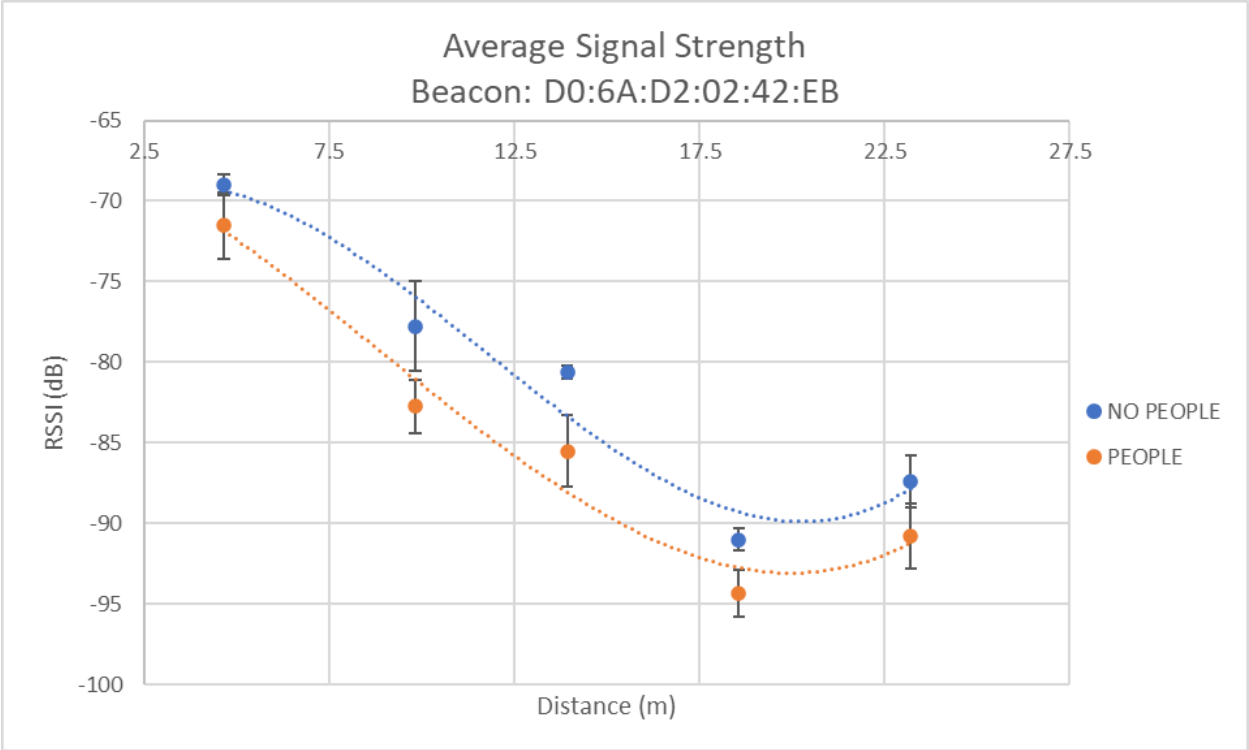

Figure 2: Beacon Strength Measurements for Beacon 42:EB

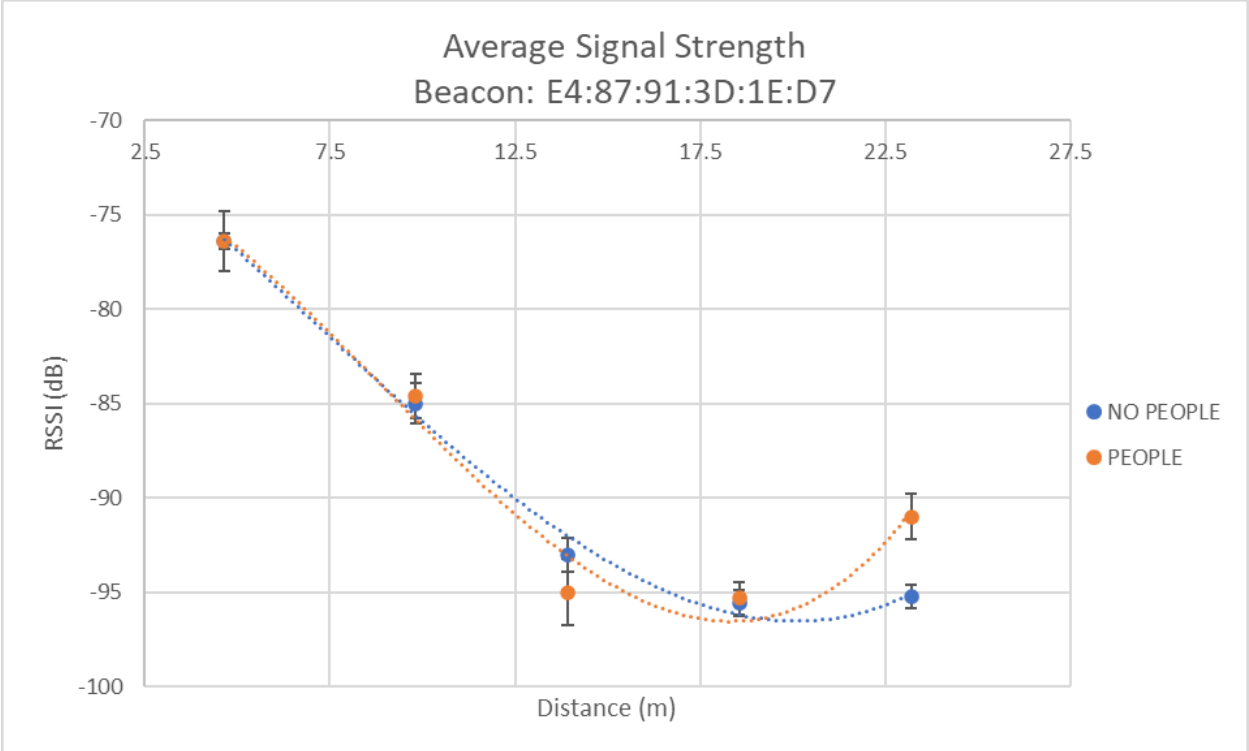

Figure 3: Beacon Strength Measurements for Beacon 1E:D7

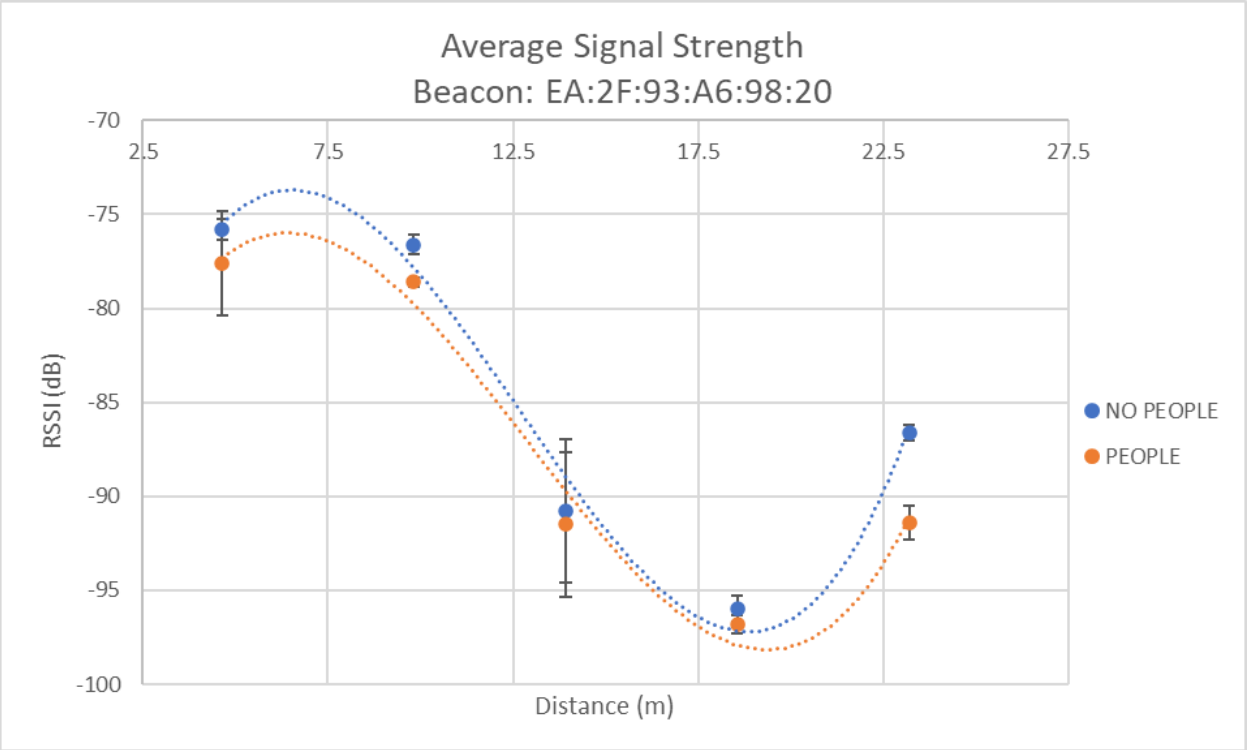

Figure 4: Beacon Strength Measurements for Beacon 98:20

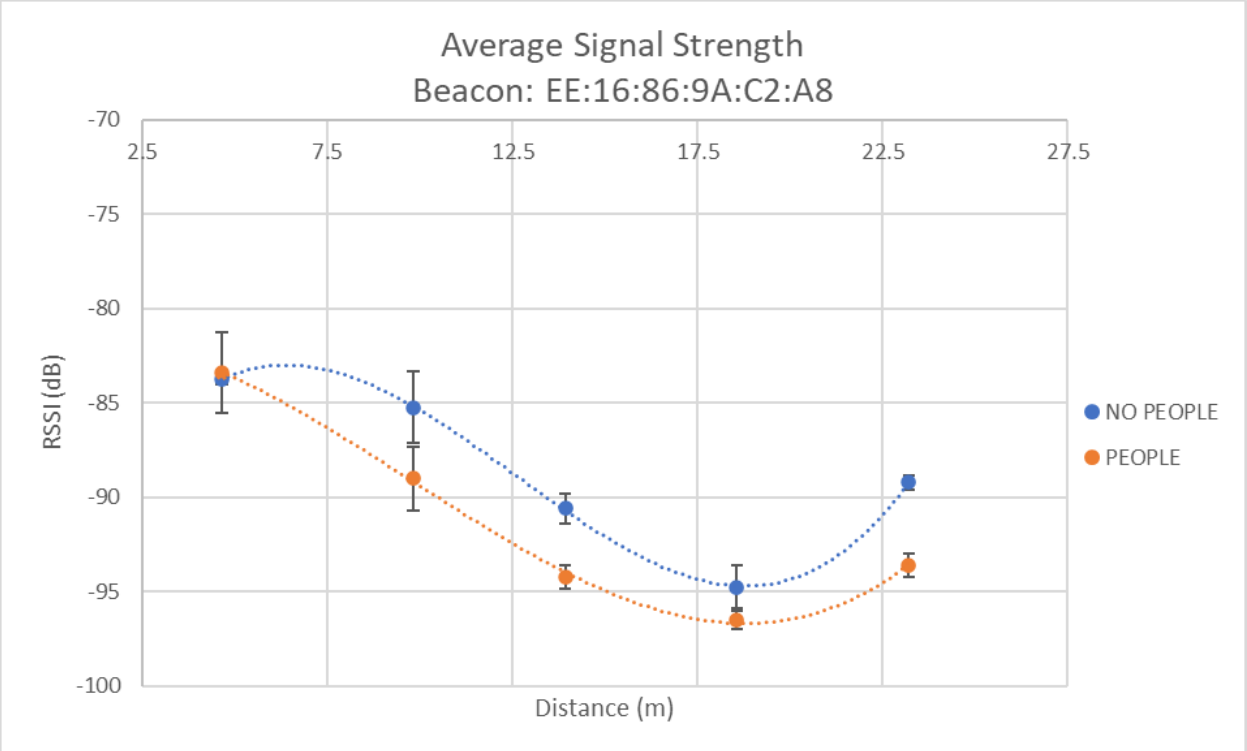

Figure 5: Beacon Strength Measurements for Beacon C2:A8

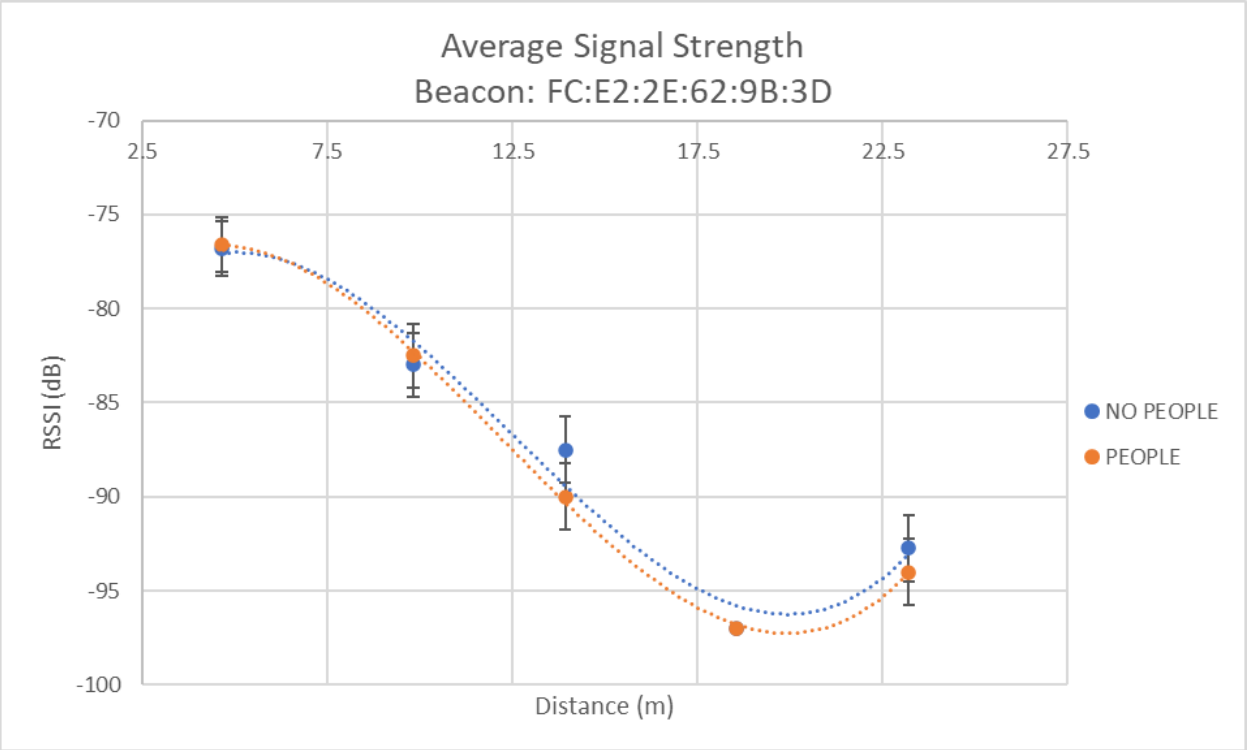

Figure 6: Beacon Strength Measurements for Beacon 9B:3D

These graphs above display the strength of the signal measured from five distances, with and without people walking between the beacon and the sensor. These graphs provide insight into the strength and consistency of the beacons, which is important information when considering the future of the project, as the beacons are required to work at a distance inside the congested tunnels. Across beacons, the average reading from each beacon deviated from each other with a standard deviation of 2.69 dB. This is a relatively high standard deviation, which could lead to consistency issues with the robot if using the current beacons. This data was graphed to obtain a trend for the signal strength of the beacons on average as shown in figure 7:

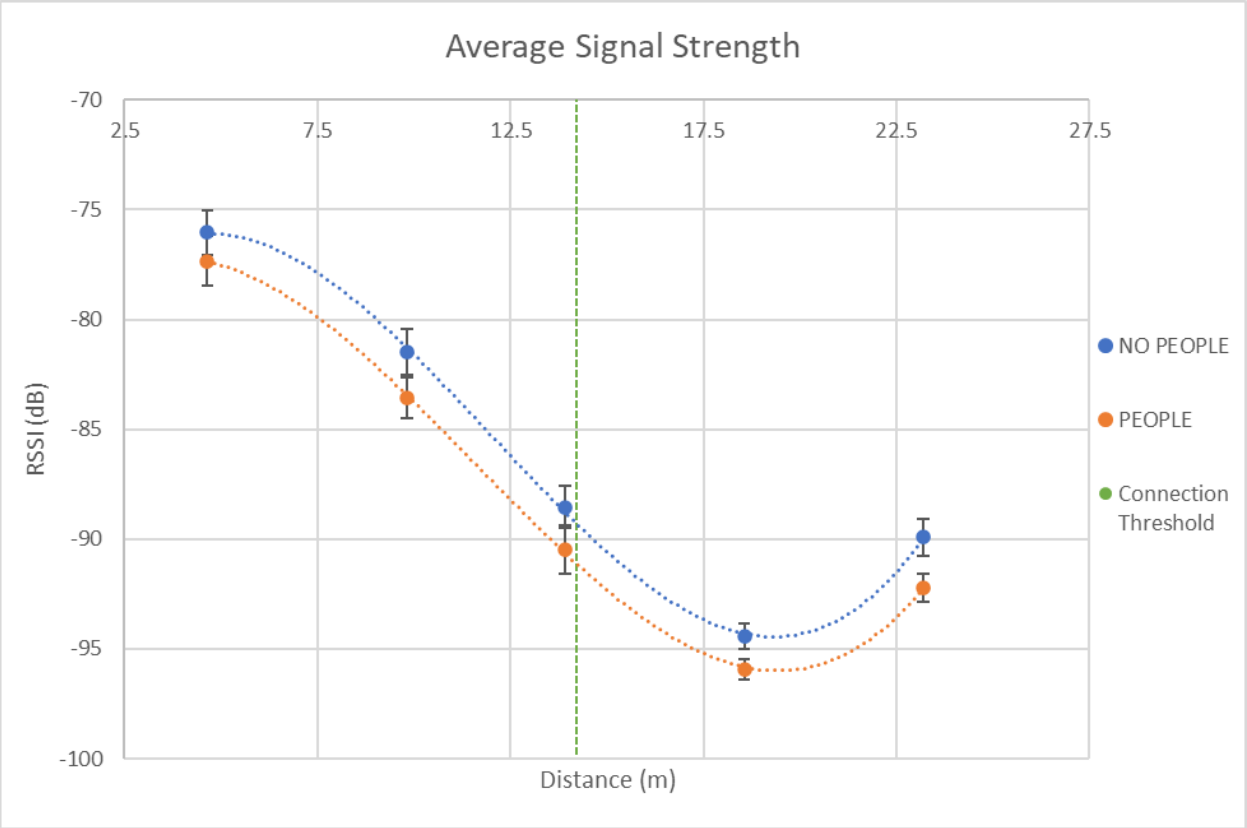

Figure 7. Average Signal Strength of Tested Beacons With and Without People

From this data, a connection threshold was established at 14.2m. Closer than this number, we achieve an average RSSI of above -90 dB, which is needed to effectively interact with the pi. Another factor considered was how often beacon connectivity was dropped, meaning no signal at all was received by the raspberry pi. A consistency factor was calculated for each beacon by dividing the number of the received signal by the number of total pings as follows:

$$CF = \frac{numRecievedSignals}{numPings} x\ 100\%$$

This was calculated for both people and no people present in front of the beacons, and then averaged across the beacons. For no people, we successfully receive a signal back 90.4% of the time. And for when there are people present obstructing the signal, we successfully receive a

signal back only 84.8% of the time. This is a difference of 5.6%, which is significant when considering the reliability requirements of the robot. Overall, this is an average consistency factor of 87.6% for the beacons.

3.3.1 Bluetooth Beacon Placement

By Favour Olotu and Emmitt Luhning

The existing implementation for navigation placed these beacons at intersections which allowed the robot to determine what intersection it was at, and where in the intersection it is situated, as well as the routing algorithm by which the robot determined which turn to make at any given intersection.

The existing placement outlined for beacon placement utilised four beacons at a four way intersection, however it was found that only two beacons were being utilised for a given turn. The robot knew it was in an intersection when it passed the first intersection, then knew it was through the interaction when it detected the second beacon that was placed along the destination hallway. It would ignore the beacons placed in destination hallways that didn’t align with its path.

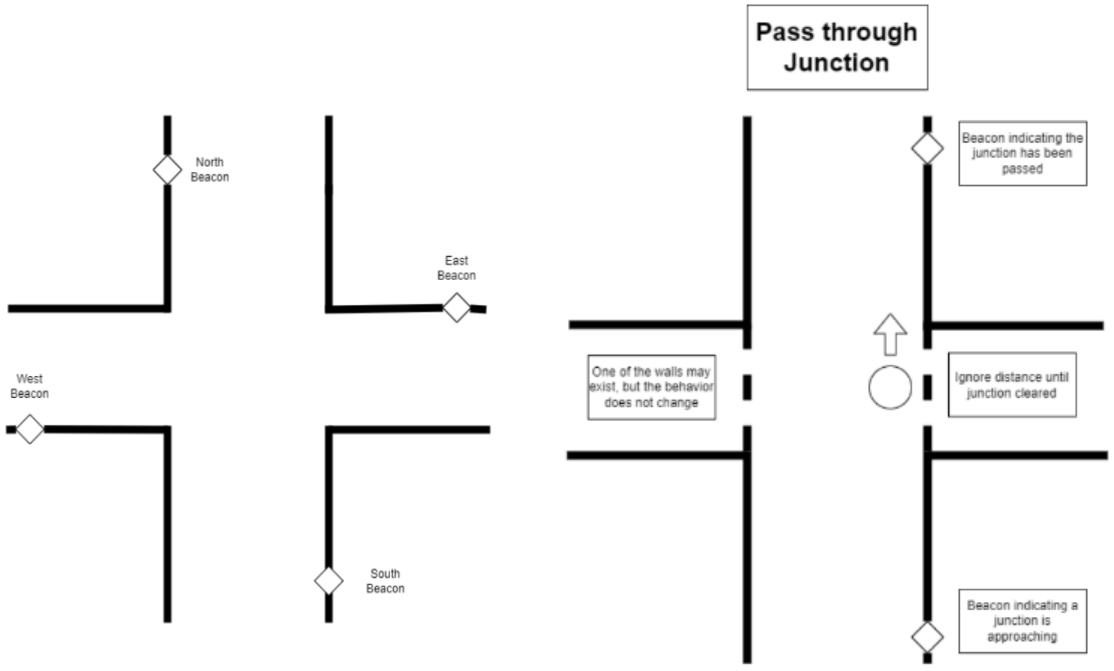

Figure 8. Diagrams of Existing Beacon Placements from Existing Work [7]

3.3.2 Slope Calculation Analysis

By Emmitt Luhning

The existing implementation allowed for the robot to know it entered the intersection when the slope of the RSSI values from the first beacon transitioned from positive to negative, and knew it left the intersection when the values for the second beacon transitioned from positive to negative.

A challenge addressed by this implementation is that of noise. The values recorded from the beacon have been seen to not be entirely consistent, which leads to expected disruptions causing a short term reversal of the measured slope. To combat this, an average of the RSSI values are calculated, and the slope of these averages are then used to determine the direction. The calculations utilised are as follows:

$$RSSI_{avg,\,i} = \frac{1}{n}\sum_{n=1}^{n} RSSI_n,\ for\ n = 5 \quad (1)$$

Where, $RSSI_{avg}$ represents the current average of the measured RSSI values, $RSSI_n$ represents the newest measured RSSI value, and i designates the number of averages calculated. Slope can then be calculated easily as:

$$Slope = RSSI_{avg,3} - RSSI_{avg,1} \quad (2)$$

The slope is calculated under the assumption that points are equidistant, resulting in a division by 1. The robot was run using this mechanism for determining its location relative to the beacon 15 runs to determine its efficacy, and of those runs, the robot was correctly able to identify that a beacon was near, and that it was approaching, and consequently passing the beacon every time.

3.4 Navigation Scheme Analysis

By Emmitt Luhning and Favour Olotu

The work from previous years proposed a check-in system for navigation, wherein a complete route is determined at the start of each run, but instead of sending that information at once, the Captain node requests it on reception of beacon data, and then relays then receives the information on a per-intersection basis. This direction is then meant to be published to the state machine.

The junction database outlined in previous years has four sections: north, south, east, and west, as well as a junction ID. These sections serve to illustrate the relationship between junction exits and beacon placements, not as actual markers of the beacon locations in the real world. For instance, if coming from the junction's southern entrance, you must turn left to reach the western

goal. Table 6 below shows the format of a row from junction information in the map representation of the tunnel.

Table 6. Map Database Junction Sample Table Row [7]

	North	**East**	**South**	**West**
Junction ID	{ Beacon ID, DestJunction }	{ Beacon ID, DestJunction }	{ Beacon ID, DestJunction }	{ Beacon ID, DestJunction }

However, despite the outline of the desired mechanism for navigation, no implementation for the navigation node, which was intended to build the route, was provided. As a consequence, no verification of the design behaviour was possible, and the robot was not able to navigate intersections.

3.5 State Machine Design Analysis

By Jacob Charpentier

The junction state machine developed previously can be seen below in figure 9. This state machine relies on five states, each of which contains a complex sub-state system. These states are as follows: WallFollowing, RightTurnJunction, PassThrough, LeftTurnJunction, and Collision Sub-states.

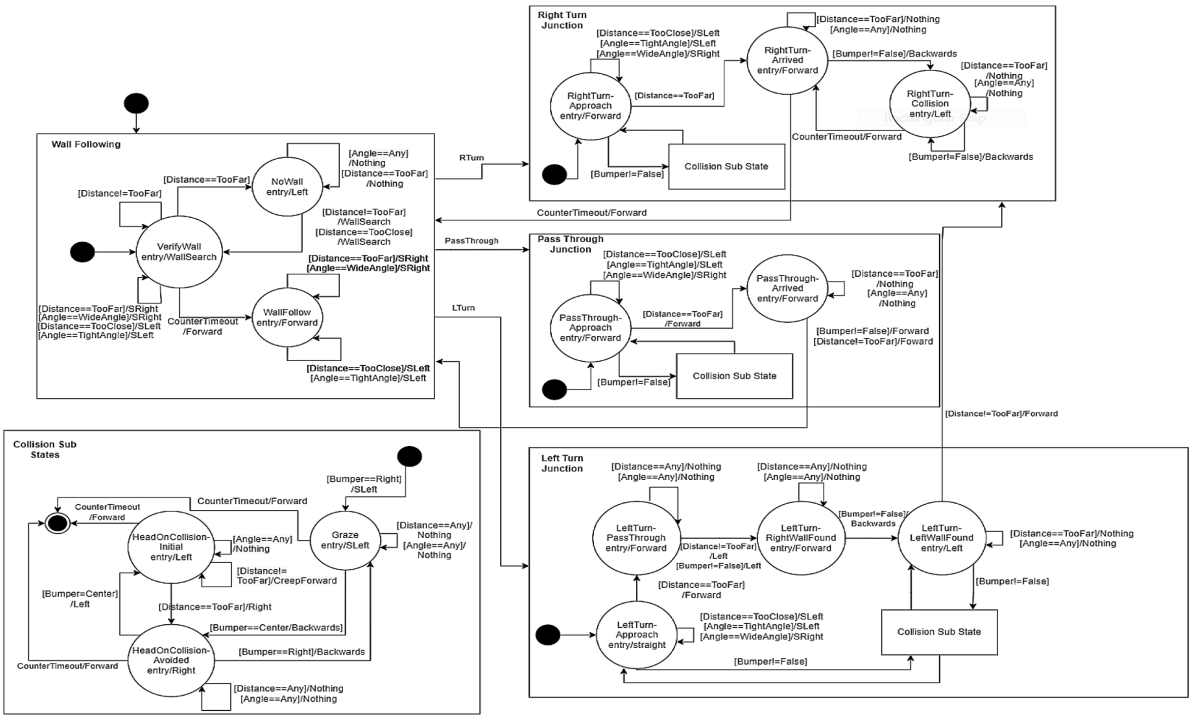

Figure 9: Junction State machine Implemented During Previous Years.

3.5.1 Wall Following States Analysis

By: Jacob Charpentier

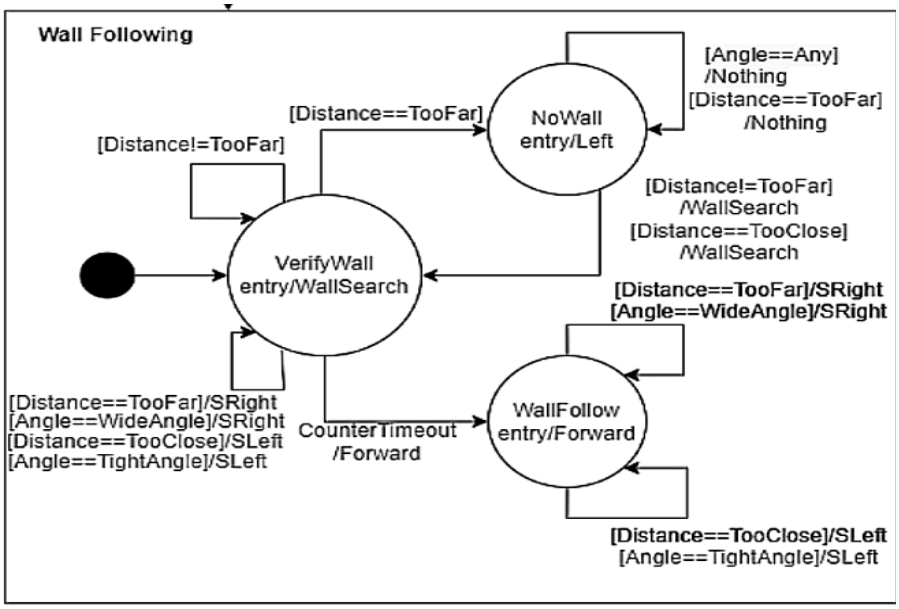

Figure 10: Wall Following State Machine Implemented During Previous Year

The WallFollowing state will verify if the wall is detected until the CounterTimeout is triggered, once it is triggered it will begin travelling forward and adjust its path to hold an ideal angle with the wall. Through analysis of this state system, there are a few flaws which may cause unintended results, these are as follows. First, the same condition from the same state causes two different outcomes (A distance of TooFar from the VerifyWall state can lead to both the NoWall and re-enter the VerifyWall state with no differentiating features). Secondly, the WallFollow state triggered by the CounterTimeout has no handling for when the Wall is lost, the robot will simply begin turning right until a wall is found, this state removes access to the VerifyWall and NoWall states unless a junction is intersected. Lastly, there is no collision handling while WallFollowing which can cause major issues as the WallFollowing state is the most active state in the system.

3.5.2 Right Turn States Analysis

By: Jacob Charpentier

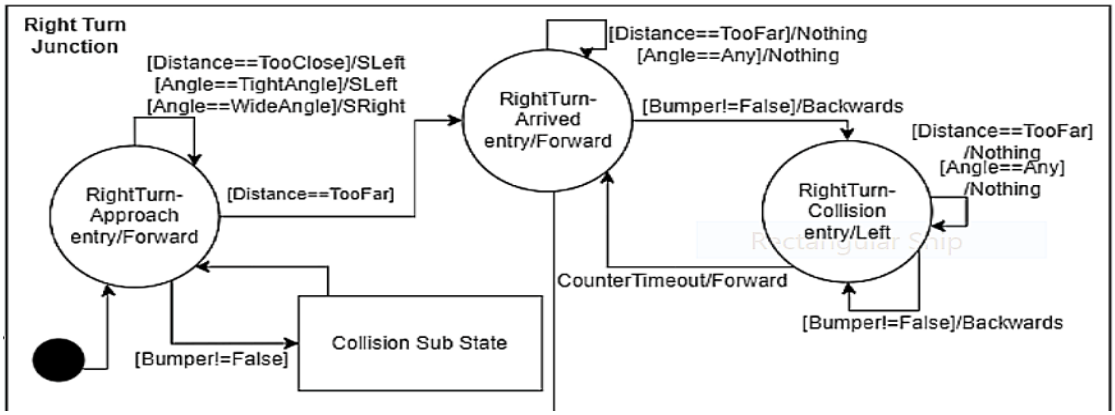

Figure 11: Right Turn State Machine Implemented in Previous Years

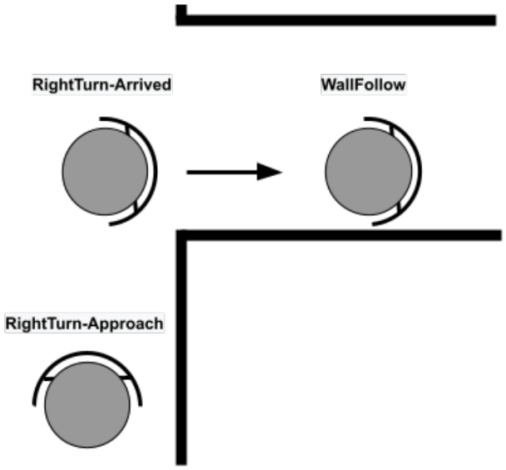

Figure 12: Right Turn Handling

When an intersection is reached and a right turn is required the state will change to the RightTurnJunction state. This state will cause the robot to travel forward and follow the wall until the wall is lost, indicating that the right turn has been reached. Once the wall is lost, the robot will begin turning right until it collides with the wall, once this collision occurs it will cause the robot to turn left and begin the wall following the new wall as shown in figure 12 above. Through analysis a couple of risks can be identified, these include, the lack of collision handling until the CounterTimeout is triggered the second time means that an accidental collision during the right turn could cause the robot to begin an infinite left turn loop, also the continuous left turning that occurs until the CounterTimeout is triggered for the first time can cause the robot to incorrectly realign with the wall as it relies on a timer based approach.

3.5.3 Pass Through States Analysis

By: Jacob Charpentier

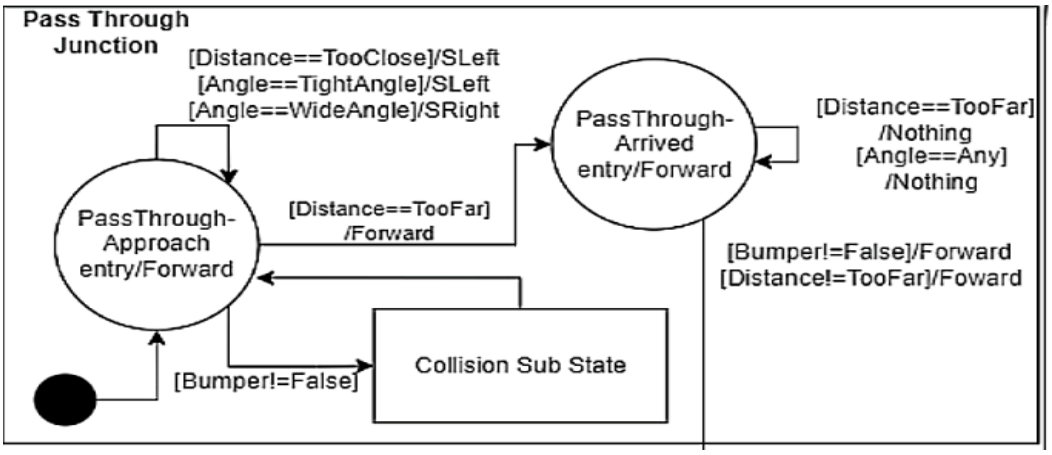

Figure 13: Pass Through State Machine Implemented in Previous Years

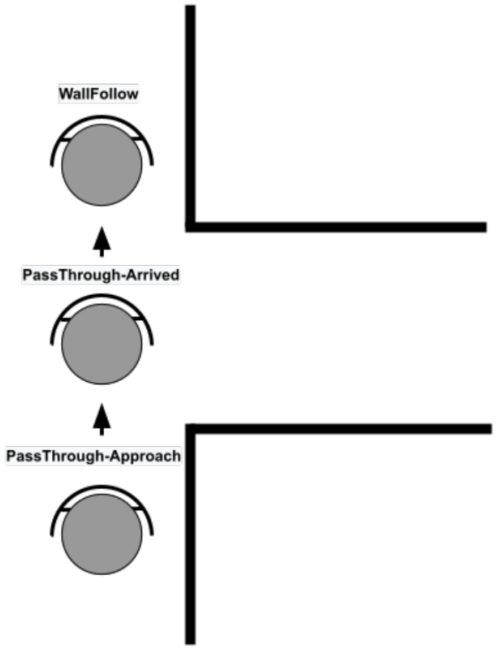

Figure 14: Straight Through Intersection Handling

When a pass-through intersection is reached the PassThroughJunction state is entered. This state will cause the robot to follow the wall forward until the wall is lost, once the wall is

lost it will continue driving forward until the wall is once again detected, when this occurs the robot will re-enter the wall following state as shown in figure 14 above. Through analysis, it can be seen that the lack of collision handling during the traversal of the intersection can cause the robot to get stuck during an unexpected collision.

3.5.4 Left Turn States Analysis

By: Jacob Charpentier, Emmitt Luhning

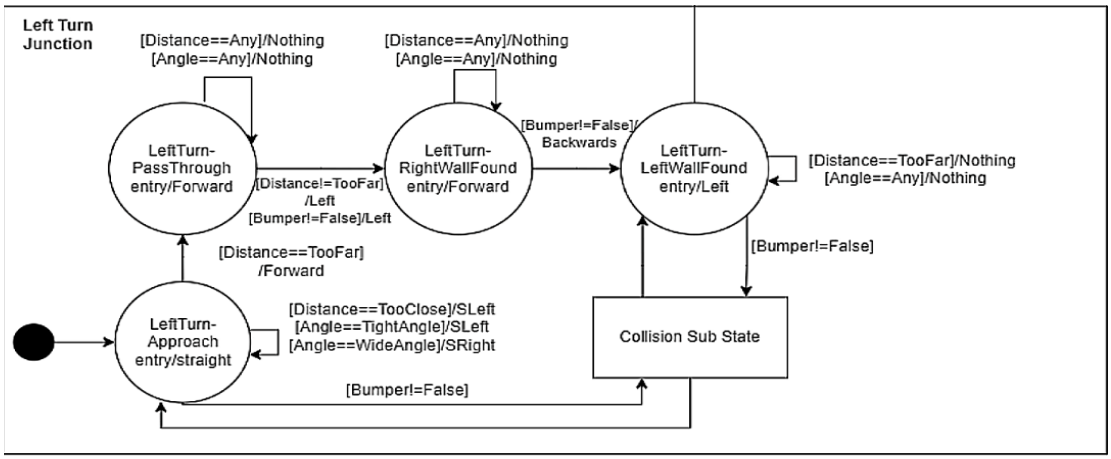

Figure 15: Left Turn State Machine Implemented in Previous Years

When a left turn intersection is reached the LeftTurnJunction state is entered. This state will have the robot approach the corner and pass through the intersection, once the pass-through is complete the robot will turn left and travel forward until a collision is detected. This collision will cause the robot to turn left and then wall follows until the edge of the intersection is reached.

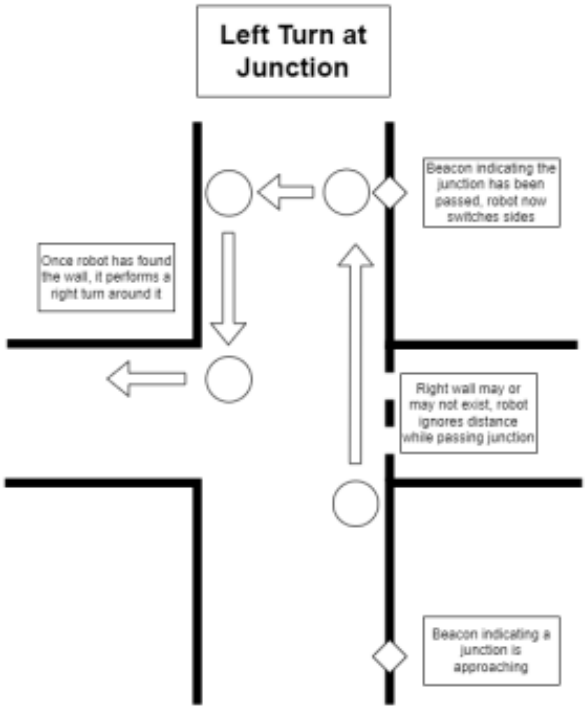

Figure 16: Existing Left Turn Logic [7]

This u-turn behaviour was found to be unreliable. As there is no environmental information to denote when the robot has turned ninety degrees, the robot was required to turn a predetermined amount by using a hardcoded angular speed. When running the robot, the assumption that the robot would always turn ninety degrees proved untrue due to deviations in the initial robot position. This is sufficient if the turn is too short, as the robot will still eventually reach the opposing wall, despite the delay caused by heading down the wrong hallway first, but in cases when the robot overshoots the turn, it often drove back into the intersection upon which point the turn was not salvageable.

3.5.5 Collision States Analysis

By: Jacob Charpentier

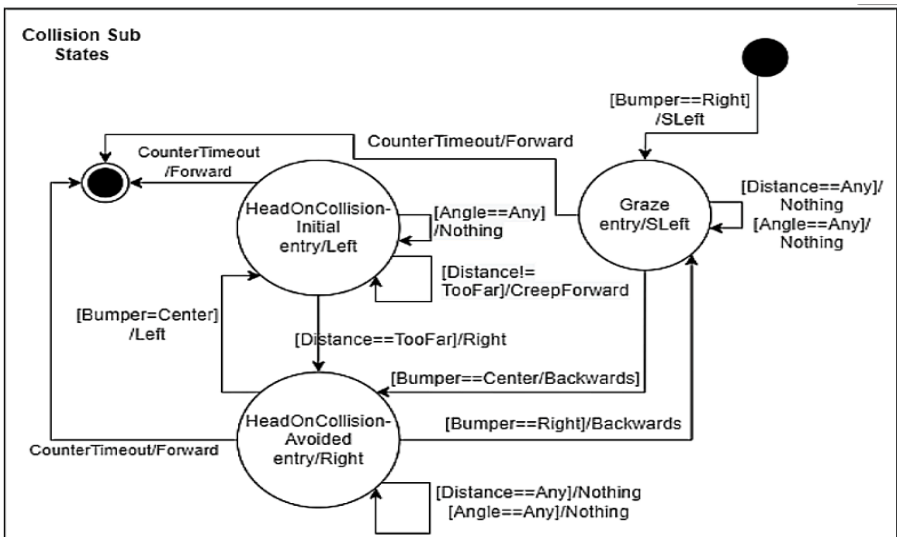

Figure 17: Collision State Machine Implemented in Previous Years

The final state is the Collision Sub-state. This state will be triggered whenever an unintended collision is detected. During this, the robot will perform a slight left turn and travel forward if no more collisions are detected. Otherwise, it will back up and turn right, if the bumper is triggered again it will perform two turns left and continue forward if no obstacle is detected. If an obstacle is detected it will continue the loop of two left turns and one right turn until the CounterTimeout is triggered, at which point it will leave the Collision Sub-state. This causes a threat to reliability, since the robot has no system to confirm if the obstacle has truly been avoided and will assume a successful avoidance after the CounterTimeout is reached even if the robot continues to detect obstacles.

3.5.6 State Machine Analysis Conclusion

By: Jacob Charpentier

Through the use of this state machine, the proposed robot would be capable of handling most scenarios that it may encounter, however, this level of complexity poses a threat. When a highly complex state machine is developed each state must be able to handle all interruptions otherwise a fault may occur and cause the robot to enter an unexpected state, by having a large number of states the chance of encountering this issue increases exponentially. Furthermore, having a complex state machine can have unintended negative side effects. Primarily, the code can become difficult to maintain and expand upon as the addition of more sensors causes the complexity to increase exponentially. Through analysis of the risks and benefits provided by this state machine, it can be surmised that this state machine requires a re-work and should be a part of the overall state machine which is not documented.

3.6 Hardware Design Analysis

By Jacob Charpentier

The work from previous years produced a circuit design capable of powering two IR sensors from the Raspberry Pi, with the data lines returning to the Raspberry Pi through an I2C converter.

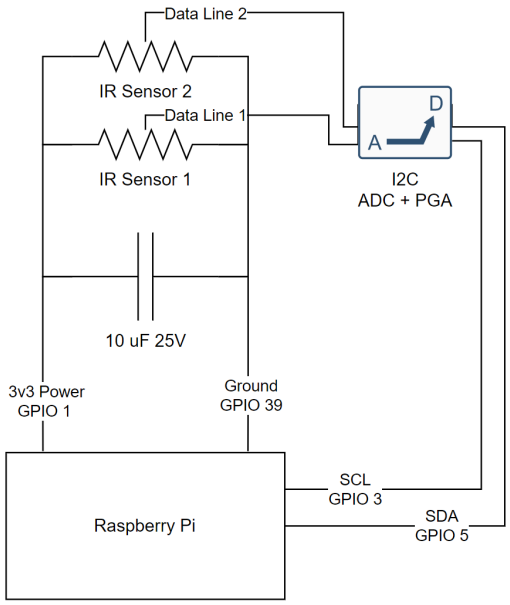

Figure 18: Existing Circuit Design [C]

This circuit was found to be successful but its implementation was limiting modification as it was soldered and could not be changed. This also meant that any issues or damaged cables could not simply be replaced and the entire circuit would have to be rebuilt. The chassis for the robot was found to be insufficient as it required affixing a highly sensitive system to a location on the robot by tape. This poor affixing system means that if the sensors are not attached perfectly, which is highly probable, then the robot would behave abnormally.

3.7. Collision Behaviour Analysis

By Emmitt Luhning

The work previously done included some attempted implementation for handling collisions. There were two generic collision handling functions specified: one for head on collisions, and one for a collision to the side bumper. The collision behaviours did not differ depending on the state they were triggered from. When run on the robot, this implementation did

not successfully avoid or handle collisions with an obstacle. Rather, a collision simply resulted in the robot throwing an error and ceasing operation.

3.8 Problem Analysis Implications

By Favour Olotu and Chase Scott

After conducting a thorough analysis of the existing system, it has become apparent that much of the functionality is inadequate and in need of refinement. Specifically, the current wall following functionality is unsuitable for navigating in a straight line due to close proximity of obstacles in the Carleton tunnel system, and is prone to oscillatory behaviour due to implementation shortcomings. Our team is dedicated to addressing these issues in the upcoming requirements section by developing a robust and consistent solution. Additionally, we discovered several unreliable elements within the state machine, with particular emphasis on the navigation behaviour as it pertains to the beacons. Lastly, in terms of testing infrastructure, docking behaviours, and collision behaviours, there ranges from stubs to no implementation at all. However, our analysis also revealed some system components that can be successfully carried over to the new system, such as the communication between the Raspberry Pi and iRobot, as well as the use of Bluetooth beacons for positional triangulation and navigation. These findings will guide our efforts in satisfying the requirements of the project as we strive for a superior and optimized system.

4.0 Requirements

By Chase Scott

Based on our analysis of the system in its current state, we have identified several functional and non-functional requirements that need to be addressed in order to improve the system's overall performance and functionality. Not all of these requirements will be explicitly accomplished in our iteration of the project, but they are kept in mind for every design decision we make, and can serve as guidelines for future groups when inheriting our system.

4.1 Non-Functional Requirements

By Chase Scott, Emmitt Luhning

The current app implementation is limited in its function. In order to engineer a good user experience, a new interface will be created to display the approximate location of the robot and allow users to communicate with the robots remotely. To accomplish this, a map of the tunnels will be created that is updated with new information whenever the robots hit a location with wifi and displayed to the user so they can know where their mail is. These design goals are reflected in the following requirements:

- When connected to the internet, the hallway the robot is currently travelling in shall be accurately reflected by the application's map.
- The robot control software shall be interoperable with the app display system

It has been identified that the robot must deliver mail promptly. In pursuing this, the team has identified that the robot's speed should be increased by smoothing the robot’s movement. To do this, a more consistent perception component will be required to reduce stuttering caused by the inconsistent IR sensors. Taking a small test section of the tunnel system at Carleton

University, the distance between the Minto Case building and the Mackenzie building through the tunnels is estimated at 210 meters, and the current average speed of the robot was measured as 0.196 meters per second as found in section 3.2. The amount of time it would take the robot to travel over 210 meters in the tunnel can be calculated as:

$$Travel\ time\ =\ Distance\ \div\ average\ speed$$

$$Travel\ Time\ =\ 1072\ Seconds\ \simeq\ 18\ minutes$$

This serves as important qualitative goal when designing our robot, so it is reflected in the following requirements:

- The robot shall be able to travel over a distance of 210 meters in approximately 18 minutes.

The safety of the system is of utmost importance, as the tunnels of Carleton are very busy, and the robot presents a distinct tripping hazard to those walking. Therefore, designing the robot in a way that is safe and respectful of the pedestrians in the tunnel is a core requirement of the system to-be:

- The robot shall navigate through the tunnels in a way that does not harm or obstruct any tunnel user's path.

The reliability of the beacons used is an important metric to consider when designing the system. If the robot is passing an intersection and cannot maintain a reliable connection, then unforeseen behaviour may be taken by the robot. With the beacon's current consistency factor of 87.6%, there is a definite chance that this can occur. Therefore, it is important to implement systems that make it so we can be sure that a return signal will be received before it is too late. This reduces our standard deviation and allows us to be more confident in our robot's ability to

navigate the tunnels and intersections. The range of the current beacon setup was found to be acceptable, being above acceptable values within 14.2m. Therefore it is essential this metric must not be sacrificed for consistency. These goals are reflected in the following requirements:

- The beacons shall be able to be interfaced with by the robot within 14.2 meters.
- The beacons shall send a return signal to the robot in enough time for the robot to take corrective action based on that signal

The robot should be charged between deliveries, or when necessary at delivery locations. Given the wide range in potential distances for any given delivery, it was determined that the robot should not simply locate a charging dock when the battery is below some threshold but that it should determine whether to charge based on a combination of factors, including both the current battery percentage and the distance to its next destination. This is reflected in the following requirements:

- The robot shall retain the operating battery power necessary to complete its current delivery.
- The robot shall be able to maintain operational status for a full 8-hour work day.

The following are some security concerns related to the operation of the robot and handling of the mail. Since the robot is designed to handle sensitive documents and requests, it is essential that this data is kept securely and only viewable by authorised personnel.

- The system shall ensure that all mail being transported can only be accessed by the sender and the recipient of the mail.
- The system shall ensure that only authorised users can view mail requests.

- The system shall only be accessible to users on the Carleton campus wifi network.

The following are operational and architectural requirements relating to how the system is to be represented, including expected platforms, setup and maintenance times, and our budget and time-frame for project completion.

- The system shall have support for PC and mobile.
- The setup for the robot shall not exceed an hour if following instructions and maintenance shall not take longer than 30 minutes for anticipated failures.
- The system shall remain under a budget of $500.
- The expected features for this semester of the project shall be completed by April 2023.

4.2 Functional Requirements

By Chase Scott

The following are the functional requirements for the robot in the form of use-case descriptors of the use-case diagram shown in figure 19 below:

Table 7. Send Mail use case

Use Case Name	Send Mail
Brief Description	A user initiates a request to send some mail via the robot
Primary Actor	User
Secondary Actor	Request Manager
Precondition	The system is running and the beacon and robots are functioning
Dependency	INCLUDE USE CASE Handle Request
Basic Flow	Steps: 1. Sender submits current location and delivery location 2. The system validates that a nearby robot is available 3. The system replies with the nearest available robot 4. The sender places mail in the robot receptacle 5. The system notifies the recipient of upcoming delivery **Postcondition:** Mail is in the robot and the user is notified
Specific Alternative Flows	RFS Basic Flow 2. Steps: 1. IF no available robot is found then ABORT ENDIF **Postcondition:** Mail cannot be sent

Table 8. Retrieve Mail use case

Use Case Name	Retrieve Mail
Brief Description	A user retrieves their mail from the robot.
Primary Actor	User
Secondary Actor	N/A
Precondition	The system is docked at a station and contains mail for the specified user.
Dependency	INCLUDE USE CASE Handle Request, Notify User
Basic Flow	Steps: 1. System notifies the sender and recipient Users that the mail has arrived. 2. The recipient retrieves the mail from the robot's mail receptacle. **Postcondition:** Mail has been successfully delivered to the recipient and the mail is out of the hands of the robot.
Specific Alternative Flows	RFS Basic Flow 2. Steps: 1. The recipient never comes to retrieve the mail. **Postcondition:** Robot resumes normal operation.

Table 9. Check Status use case

Use Case Name	Check Status
Brief Description	A User checks on the status of their current delivery.
Primary Actor	User
Secondary Actor	N/A
Precondition	The system is running and the User has a delivery in progress.
Dependency	N/A
Basic Flow	Steps: 1. The User requests to view the status of the delivery. 2. The system replies with the most recent information regarding the robot's position and delivery status. **Postcondition:** The user has been informed of the status of their delivery.
Specific Alternative Flows	N/A

Table 10. Monitor System use case

Use Case Name	Monitor System
Brief Description	An admin monitors the current state of the system.
Primary Actor	Admin
Secondary Actor	N/A
Precondition	System is running
Dependency	N/A
Basic Flow	Steps: 1. Admin requests to view the current state of the system. 2. System replies with information regarding the state of the current deliveries and the robot. **Postcondition:** Admin is aware of the system state.
Specific Alternative Flows	RFS Basic Flow 2. Steps: 1. IF System is not running, THEN send an error ENDIF **Postcondition:** Admin is aware that the system is not running.

Table 11. Handle Request use case

Use Case Name	Handle Request
Brief Description	The Request Manager receives a request from a user and handles it.
Primary Actor	Request Manager
Secondary Actor	User
Precondition	System is running
Dependency	INCLUDE USE CASE Notify Robot
Basic Flow	Steps: 1. USE CASE Send Mail or Receive Mail is initiated. 2. The type of use case is interpreted by the Request Manager. 3. The action to take is calculated and sent to the robot. **Postcondition:** The robot is notified of the new request.
Specific Alternative Flows	RFS Basic Flow 2. Steps: 1. IF robot is unable to be communicated with, THEN wait until the robot has a connection and attempt to send again ENDIF. **Postcondition:** The robot is notified of the new request.

Table 12. Notify Robot use case

Use Case Name	Notify Robot
Brief Description	The robot is notified of a new request that needs to be handled.
Primary Actor	Request Manager
Secondary Actor	N/A
Precondition	System is running
Dependency	N/A
Basic Flow	Steps: 1. USE CASE Handle Request has finished. 2. The robot has received a new request to be handled. **Postcondition:** The robot is handling the new request.
Specific Alternative Flows	N/A

Table 13. Navigate Hallway use case

Use Case Name	Navigate Hallway
Brief Description	The robot is traversing through a hallway towards its next destination.
Primary Actor	N/A
Secondary Actor	N/A
Precondition	System is running and robot has a request to handle
Dependency	EXTENDS USE CASE Notify Robot
Basic Flow	Steps: 1. The robot retrieves sensor information from its surroundings. 2. The robot calculates its next action to take based on this information. 3. The robot executes the next action. 4. Loop steps 1-3. **Postcondition:** The robot is successfully navigating the hallway.
Specific Alternative Flows	RFS Basic Flow 2. Steps: 1. IF the robot is unable to determine the best action to take, THEN enter an error state ENDIF. **Postcondition:** System admins are notified that there has been an error.

Table 14. Handle Collision use case

Use Case Name	Handle Collision
Brief Description	The robot collides with an object and attempts to navigate around it.
Primary Actor	N/A
Secondary Actor	N/A
Precondition	System is running and a collision has occurred
Dependency	EXTENDS USE CASE Notify Robot
Basic Flow	Steps: 1. The robot bumpers detect a collision has occurred. 2. The robot accelerates backwards. 3. The robot turns away from the collision source and attempts to move forward. **Postcondition:** The robot has successfully navigated the collision source.
Specific Alternative Flows	RFS Basic Flow3. Steps: 1. IF another collision occurs, THEN repeat steps 2-3 until clear ENDIF. **Postcondition:** The robot has successfully navigated the collision source.

Table 15. Navigate Intersection use case

Use Case Name	Navigate Intersection
Brief Description	The robot is traversing through a hallway towards its next destination and encounters an intersection.
Primary Actor	N/A
Secondary Actor	N/A
Precondition	System is running and an intersection is encountered.
Dependency	EXTENDS USE CASE Notify Robot
Basic Flow	Steps: 1. The robot determines what intersection it is currently at. 2. The robot Determines what direction it needs to go in order to reach its destination. 3. The robot makes the necessary turn and continues USE CASE Navigate Intersection. **Postcondition:** The robot has successfully navigated the intersection.
Specific Alternative Flows	N/A

Table 16. Read Beacon use case

Use Case Name	Read Beacon
Brief Description	The robot is traversing through a hallway towards its next destination and senses data from a beacon.
Primary Actor	N/A
Secondary Actor	N/A
Precondition	System is running and a beacon signal is encountered.
Dependency	EXTENDS USE CASE Notify Robot
Basic Flow	Steps: 1. The robot determines what intersection the beacon is transmitting from. 2. The robot determines if it is travelling towards or away from the beacon. 3. The robot logs this information for use in other systems. **Postcondition:** The robot has successfully read the beacon data.
Specific Alternative Flows	N/A

Table 17. Station Dock use case

Use Case Name	Station Dock
Brief Description	The robot has arrived at a station it intends to dock at.
Primary Actor	N/A
Secondary Actor	N/A
Precondition	System is running and a docking station is encountered.
Dependency	EXTENDS USE CASE Notify Robot INCLUDES USE CASE Notify User
Basic Flow	Steps: 1. The robot drives into the docking station and begins to charge. 2. The robot initiates the USE CASE Notify User that it has arrived. **Postcondition:** The robot is charging and the relevant users have been notified.
Specific Alternative Flows	N/A

Table 18. Notify User use case

Use Case Name	Notify User
Brief Description	The system retrieves new information and notifies the users who are privy to that information.
Primary Actor	N/A
Secondary Actor	N/A
Precondition	System is running and new information to notify users is received.
Dependency	EXTENDS USE CASE Notify Robot INCLUDES USE CASE Notify User
Basic Flow	Steps: 1. The system determines what users of the system the message is intended for. 2. The system notifies those users of the new information regarding the system. **Postcondition:** The relevant users have been notified.
Specific Alternative Flows	N/A

Figure 19: Mail Delivery Robot Use Case Diagram

4.3 Requirements Implications

By Chase Scott

Now that we have identified these requirements, we have gained a comprehensive understanding of what the system needs to accomplish and how it should behave in order to meet the expectations of the system goal. This knowledge can then be used to guide the design process, as we develop a solution that addresses all of the requirements identified. Ultimately, having a clear understanding of both functional and non-functional requirements enables us to design a system that meets the needs of the project description and performs optimally in its intended context.

5.0 Design & Implementation

By Chase Scott

Moving from requirements to design is a crucial phase in the development life cycle, where the focus shifts from identifying what the software should do to how it should do it. Designing a software system involves making crucial decisions about technologies, frameworks, and hardware platforms, as well as considering factors such as scalability, maintainability, and reliability. A well-designed system is essential to ensure that it can meet the intended purpose of our system effectively and efficiently. In the following paragraphs, we will delve into the various aspects of our design and explore its implementation.

5.1 State Machine Design

By Jacob Charpentier

As determined through analysis of the state machine there are unnecessary states, overly complex sections, and several reliability issues which could cause unexpected responses. To solve this issue several iterations of new state machines were developed, with the final version being discussed. Given that the previous system was overly complex, the goal of this state machine was to simplify the system while improving reliability by handling all events at each state. The new state machine can be seen below, split into three sections, the main module, the intersection handling substates, and the collision handling substates.

5.1.1 Main Module

By Jacob Charpentier

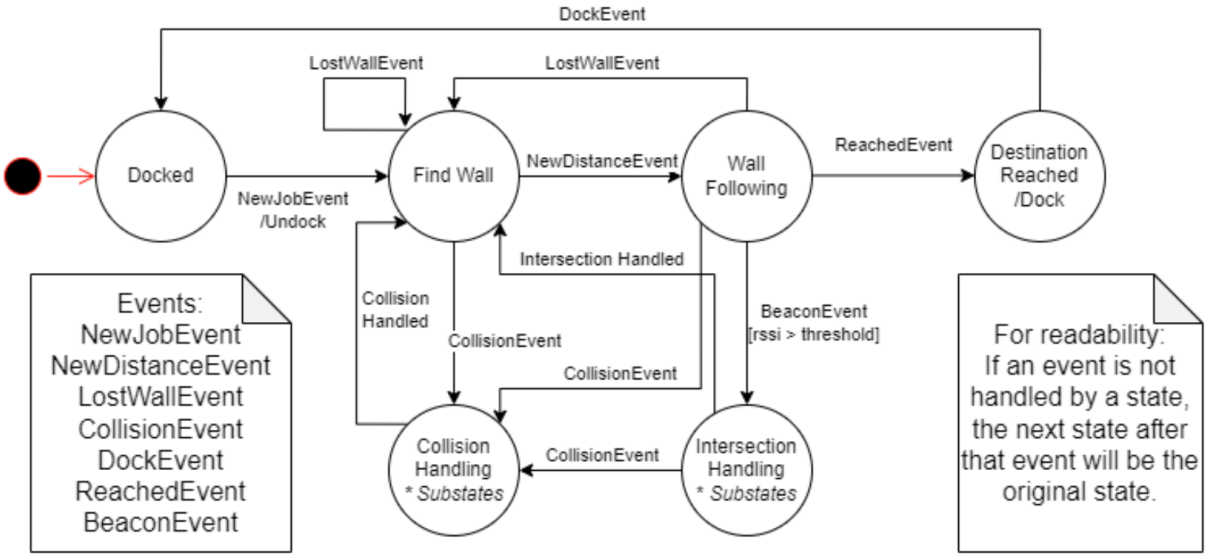

Figure 20: Main State Machine Module

The state machine designed above is the main module of the new design. The new version simplifies the system down to two supplementary states, the Docked and DestinationReached states, and four main movement states. These four main states are the FindWall state, the WallFollowing state, the IntersectionHandling state, and the CollisionHandling state.

The Docked state is active when the robot is docked in a charging station, this state serves to allow the robot to charge until a NewJobEvent is detected at which point it will move to the FindWall state.

The FindWall state will cause the robot to slowly search for a wall. This state can either experience a CollisionEvent and move into the Collision state or it can experience a

NewDistanceEvent and move into the WallFollowing state. Once in the WallFollowing state the robot will implement the PID wall following functionality. This state can experience either a LostWallEvent where it will return to the FindWall state, a CollisionEvent causing it to enter the CollisionHandling state, a BeaconEvent causing it to enter the IntersectionHandling state, or a ReachedEvent causing it to enter the DestinationReached state. The CollisionHandling and IntersectionHandling states will be further described in the next sections in 5.1. The DestinationReached state indicates that the robot has reached its destination and is ready to receive or deliver mail, these destinations are indicated by charging stations and the robot will then proceed to dock and begin charging as it changes to the Docked state.

5.2 Navigation Design

5.2.1 Beacon Placement Revision

By Emmitt Luhning and Favour Olotu

As determined through the analysis of the beacons, and their given ranges, it can be seen that the range of a given bluetooth beacon is greater than the dimensions of any of the tunnel intersections. This fact, alongside a recommendation from the previous project team to consider alternative strategies informed the decision to adjust the design to utilise one beacon at each intersection, rather than four.

Given that the previous implementation only used two beacons for a given turn, one to determine when the robot entered the intersection, and one to determine when it left, the decision was made to instead calculate the direction that the robot was moving relative to the beacon in order determine whether it was moving toward or away from an intersection. Rather than looking

for two different beacon IDs, and 2 slopes, the new solution simply looks for one beacon ID to determine which intersection it is at, and the slope of that beacon's RSSI values to determine whether it is approaching or leaving the intersection. As such, the new design required beacons to be fixed at the centre of each junction, rather than at the entrance and exits.

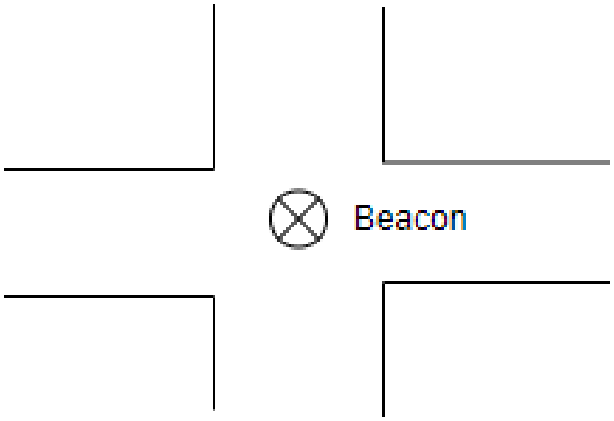

Figure 21: Updated Junction Beacon Placement

5.2.2 Route Determination and Following

By Emmitt Luhning and Favour Olotu

While there was no implementation provided for this behaviour, the check-in system outlined was deemed to be a viable strategy for navigating with the state machine in mind, as it allows for events to be published when necessary. In order to accomplish this a physical map of the tunnels needed to be converted to a data file, which could be read by a node to provide the total route that would be referenced by the Captain. The resultant csv file of the map is demonstrated below.

	A	B	C	D	E	F	G	H	I	J
1	Junction ID	North Beacon ID	North Junction ID	East Beacon ID	East Junction ID	South Beacon ID	South Junction ID	West Beacon ID	West Junction ID	
2	1	2	2	none	none	none	none	none	none	
3	2	3	3	none	none	1	1	none	none	
4	3	7	7	4	4	2	2	none	none	
5	4	5	5	8	8	none	none	3	3	
6	5	6	6	none	none	4	4	none	none	
7	6	none	none	none	none	5	5	7	7	
8	7	none	none	6	6	3	3	none	none	
9	8	none	none	none	none	none	none	4	4	

Figure 22: Graph Representation of the Developmental Map

This then needs to be converted by a node into a graphical representation so that a search can be done on it. In total, this results in a system where the navigation node can provide the following output, which is saved into a queue for use within the system.

```
Path going from junction 1 to 5: ['1', '2', '3', '4', '5']
Initial direction: straight
--------------------------------------------------------------------------------
At beacon 2
            ------> straight
--------------------------------------------------------------------------------
At beacon 3
            ------> right
--------------------------------------------------------------------------------
At beacon 4
            ------> left
--------------------------------------------------------------------------------
At beacon 5
Arrived at destination!
```

Figure 23: Sample Path Plotting Output

5.3 Navigation Implementation

5.3.1 Detection of Intersection and Associated States

By Emmitt Luhning

In order to eliminate one of the two beacons being used at intersections, it was necessary to derive a new way for the robot to know that it had passed the intersection beyond just the slope of the RSSI value. This is because, as there is a wide range of intersection sizes that may be encountered, the threshold RSSI that would be adequate for one intersection, would not translate to others. In order to accomplish this, the new design relies on a combination of inputs from the beacons, and the IR sensors. If a beacon has been sensed, but the IR sensor still sees a wall, then it should follow as normal, and if the signal is decreasing, but a wall is not seen, the robot should not yet enter normal wall following behaviour. To accomplish this for the varieties of turns that may occur, the following states were introduced.

Intersection Reached Right Wall Following

This state is triggered when the captain node sees a beacon and sends a message with the direction for the turn to the state machine telling it that it will turn right, but a wall is still seen, so regular wall following behaviour should still be utilised. When the wall is lost, the state will then transition.

Intersection Reached Right Wall Lost

This state is triggered when the system is in the state Intersection Reached Right Wall Following, and the IR sensors no longer detect a wall. This state contains the logic that will cause the robot

to begin turning right in small increments, while applying a small amount of forward movement, until the wall has been seen again, upon which time it will revert to the generic Wall Following state.

Intersection Reached Left Wall Following, and Intersection Reached Left Wall Lost

These two states behave in the same manner as for the right turn, given that the signal from the captain instead dictates the robot should turn left. First the state will transition to Intersection Reached Left Wall Following, when a beacon is detected, then to Intersection Reached Left Wall Lost, when the wall is no longer seen by the IR sensors, triggering the left turn logic, and subsequent transition to generic Wall Following.

Intersection Reached Straight Wall Following and Intersection Reached Straight Wall Lost

These sets of states behave the same as the previous ones, with the only difference being they are triggered by a “straight” signal from the captain, and the logic within Intersection Reached Wall Lost simply requests the robot move forward through the intersection until a wall is seen once again.

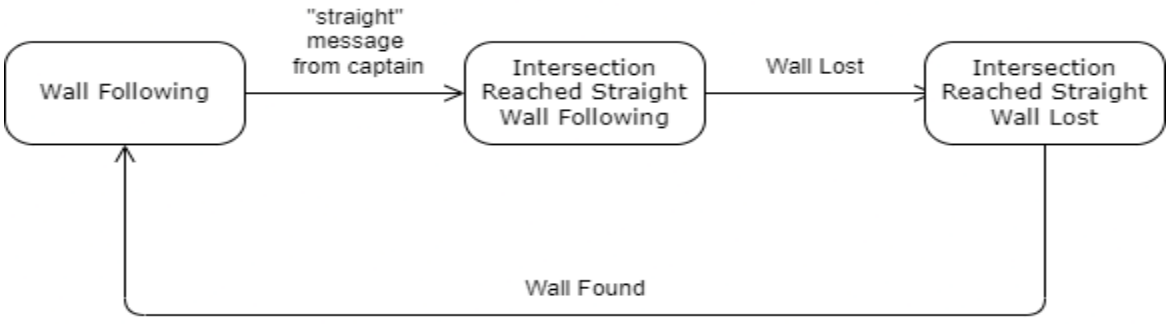

Figure 24: Illustration of Moving Straight Through Intersection State Behaviour

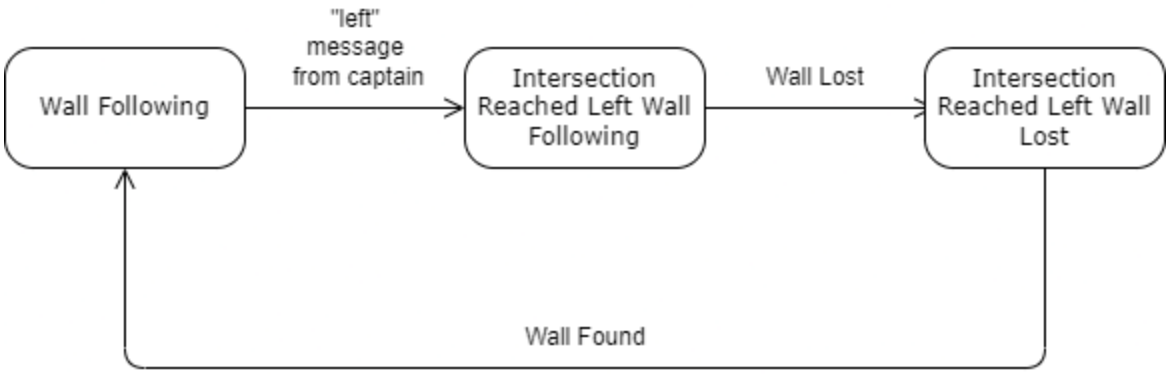

Figure 25: Illustration of Left Turn State Behaviour

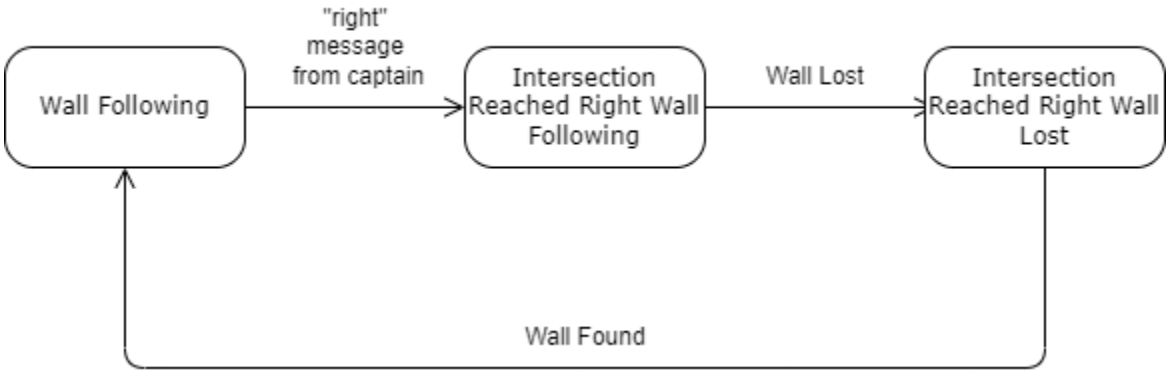

Figure 26: Illustration of Right Turn State Behaviour

The above explanations and diagrams of the states and their transitions are aimed to illustrate the intended turn behaviour without interruption and assume that there are no collisions or other events occurring within the system during the execution of these states. In actuality, there are more than one state transitions from each of the described states that can occur. Discussion of collision behaviour is described in the collision handling section.

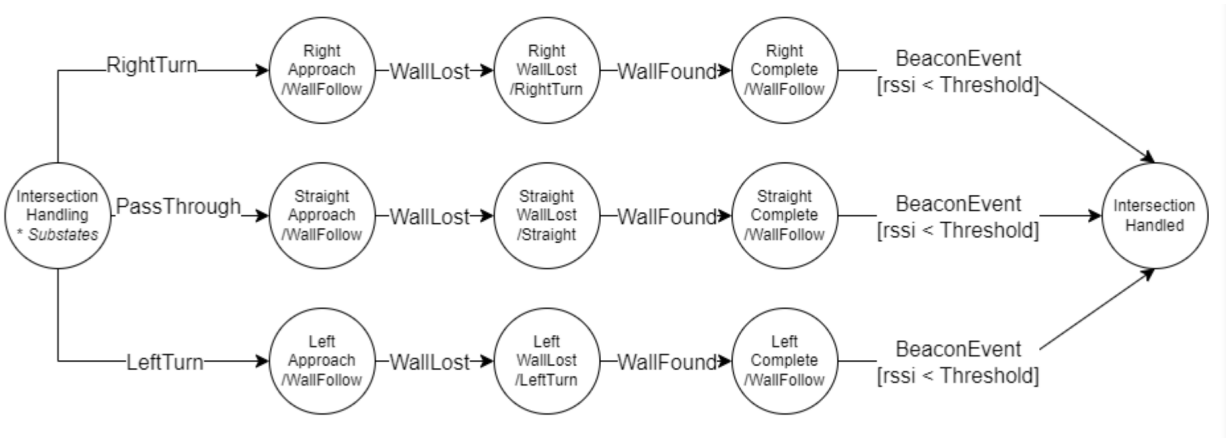

Figure 27: Overview of All Three Turn Type State Transitions

5.3.2 Turn Behaviour

By Emmitt Luhning

The right and straight behaviours at intersections remained mostly unchanged throughout the development of this project, however the shortcomings of the left turn were addressed. Instead of attempting to make a u-turn, the new left turn behaviour attempts to turn left directly, and adjust once it hits a wall, or sees one with the IR sensors, depending on which of those two events occurs first. Upon hitting a wall, the robot simply backs up, turns left slightly and tries again. This solution was found to be insufficient for certain conditions, however, as described in the testing and recommendation sections.

5.3.3 Navigation Node

By: Emmitt Luhning, Favour Olotu

In order to implement the proposed design for the navigation system, the missing navigation node had to be implemented to construct the graph from the map data, and provide information to the Captain upon request. This node includes functionality to read the map file,

determine which beacons are expected to be encountered along the path for a given intersection, and execute a breadth first search of the map data to determine the best route, which it saves in a queue which it uses to provide the captain with the next instruction.

```
"""
This function performs a breadth first search on the map graph
to find the shortest path from the source to the destination junction.
Returns a list of junctionIDs of shortest path
"""
root = self.junction_id_to_vertex_number(map_graph, source_junction)

visited = [False] * len(map_graph)
queue = []
traceback = []
traversal = []
found = False

queue.append(root)
visited[root] = True

while queue:
    root = queue.pop(0)
    traversal.append(root)

    for i in map_graph[root][1]:
        if i[1] == destination_junction:
            traceback.append(i[1])
            found = True
            break
        num = self.junction_id_to_vertex_number(map_graph, i[1])
        if num != -1 and visited[num] == False:
            queue.append(num)
            visited[num] = True

    if found:
        traversal.reverse()
        for vertex in traversal:
            for i in map_graph[vertex][1]:
                if i[1] == traceback[-1]:
                    traceback.append(map_graph[vertex][0])
                    break
        traceback.reverse()
```

Figure 28: Implementation of a Breadth First Search in Navigation

5.3.4 Beacon Sensor Node

By: Favour Olotu

Another aim for this project was to correctly implement the beacon sensor node, which didn't function out of the box, or communicate with the rest of the system. The "beacon_sensor" node works by frequently scanning for Bluetooth signals, it must then determine if this signal belongs to a beacon and which beacon. This is done by inspecting the MAC address of the Bluetooth signal and referring to a cached lookup table stored on the robot. If the robot detects a Bluetooth signal for which the MAC address does not exist on the lookup table, it will ignore it. The lookup table contains the MAC address and identifying information about the junction that the robot is approaching for each Bluetooth beacon. The lookup table is "Beaconlist.csv" which contains a list of all known and used Bluetooth beacons.

Table 19. Bluetooth MAC Address Lookup Sample Table Row

Beacon ID (as represented in the map)	Beacon MAC Address

Table 19 above is a format of the sample row from the Bluetooth signal MAC address lookup table. After the beacon has been successfully identified, the beacon data is organised in a way the captain is expecting and then sent to the captain through the "beacons" topic.

5.4 Wall Following Design

By Chase Scott

One of the most important aspects of this project is the ability of the robot to be able to traverse the tunnels in a way that is robust and consistent. The tunnels of Carleton are hectic and prone to many different environmental variables, so the success of this project as a product ultimately depends on its ability to navigate these tunnels without failure. The two qualitative metrics that capture these requirements are robustness and consistency, and were kept in mind for every upcoming design decision made. The main two methods we used to improve these metrics are the implementation of the PID Controller which correlates to the robustness metric, and the use of machine learning which correlates to the consistency metric.

5.4.1 PID Design

By Chase Scott

In order to fix the current implementation of the IR sensor calculations, where the calculations assume an isosceles triangle, the equations must consider the scenario of the robot not being perpendicular to the wall. Our ideal scenario for the wall following behaviour is an angular heading of 80 degrees, with a distance to the wall between 10cm and 20cm. In order to find these numbers, we must model the system and calculate them based on our inputs. The following diagrams show the IR Sensor configuration in its ideal scenario:

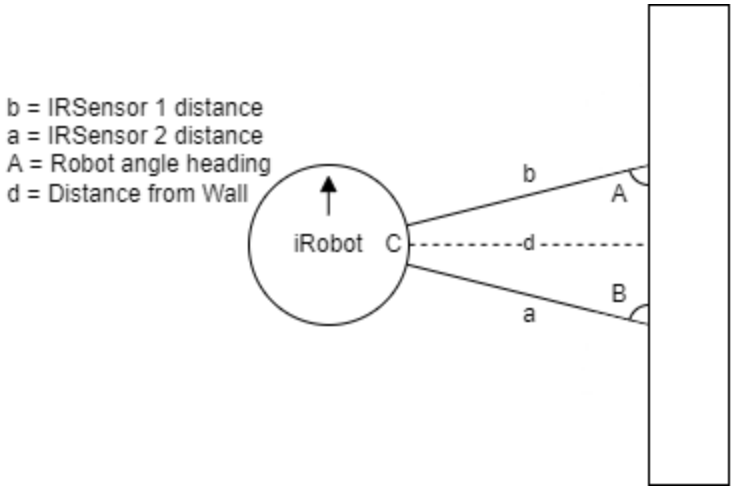

Figure 29: Ideal Wall Following Scenario

The important measurements to gather from this are d, which is the distance between the wall and the robot, and A, which is the angular heading of the robot. Our inputs are b and a, which are the readings received from the IR sensors, and C, which is the angle at which the IR Sensors are affixed to the robot. First, we need to calculate c, which is the distance between where the infrared beams intersect the wall:

$$c = \sqrt{a^2 + b^2 - 2ab * cos(\frac{\pi * C}{180})}$$

Next, we can calculate the angle A in degrees which is the angular heading of the robot with respect to the wall:

$$A = arccos\left(\frac{b^2 + c^2 - a^2}{2bc}\right) * \frac{180}{\pi}$$

Then finally to get the distance from the wall we do one last calculation:

$$d = b * sin(\frac{\pi * A}{180})$$

Now that we have calculated the distance from the wall, and the angular heading, these numbers can be fed into our state behaviours in order to tell the robot how to act. In order to do this, it is necessary to employ the use of control theory which is a field of mathematics which aims to control dynamic systems in engineered processes and machines. The objective in relation to this system is to develop an algorithm governing the movement of the robot, which drives the system to our desired state or SetPoint while minimising delay, overshoot, and ensuring high stability. In order to accomplish this, we will employ the use of a PID controller, which stands for Proportional, Integral, and Derivative.

A PID controller takes into account the magnitude of error and attempts to add terms that ramp to the desired SetPoint while offering a damping response to ensure there is no overshoot. The output *u(t)* of this analysis is calculated as follows:

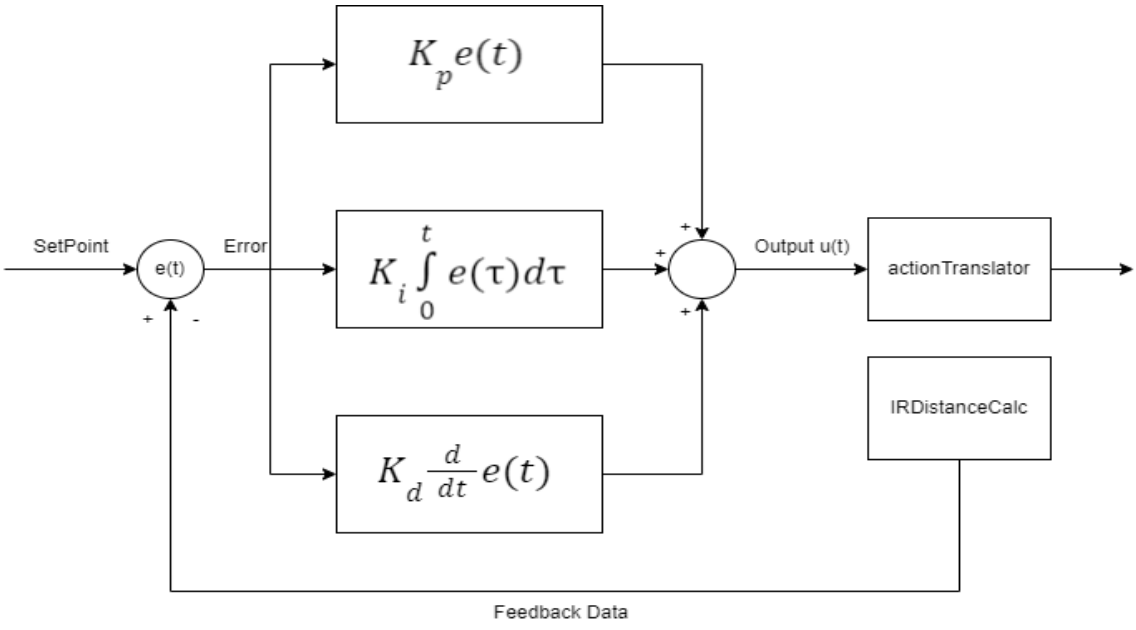

Figure 30: PID Controller Diagram

This output is then fed back into the robot to motivate its future movements in reaching the SetPoint. Based on the analysis of the current system, a follow distance between 10cm and 20cm to the wall seems optimal for this system, therefore a SetPoint of 15cm will be used as our desired state.

Choosing specific constants Kp, Ki, and Kd for the controller is the most important part of designing this type of behaviour, therefore having a way of rapidly testing different values and measuring their effectiveness against each other is important for determining which gives us the best performance. The graph for a control run for the robot starting at 23 cm and normalising to 15 cm is shown below, with the constants Kp = 1; Ki = 0; Kd = 5.

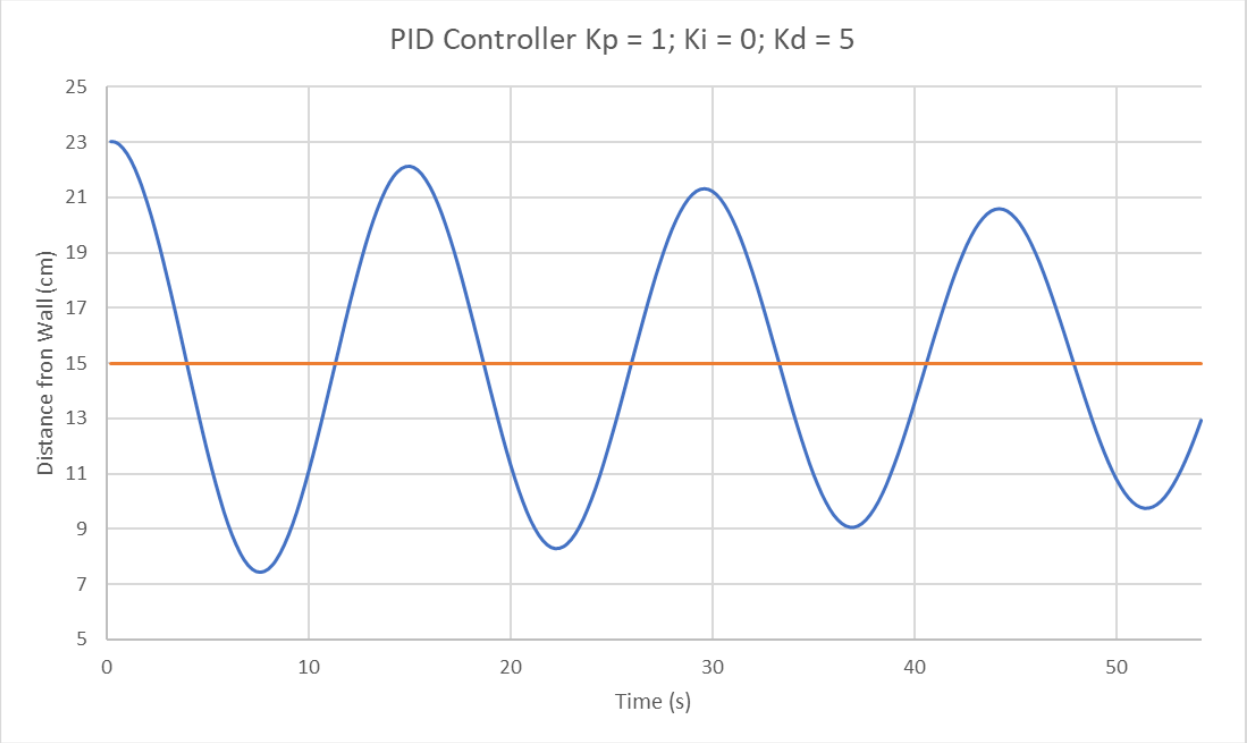

Figure 31: Initial PID Controller Test Run

As we can see, there is a lot of room for improvement in the choice of constants to reduce overshoot and increase the damping response of the output. Because these constants are chosen on a system-by-system basis, they require manual tuning to achieve optimal results, and will also probably differ across different iRobot Create deployments. Therefore it is imperative that a way to automate this tuning process is created in order to have confidence that the best values are being used

5.4.2 Obtaining PID Constants using a Genetic Algorithm

By Chase Scott

The crux of the problem regarding this design is finding values Kp, Ki, and Kd which cause the robot to follow the wall in the most efficient way possible. There are many ways to define efficiency, but for the purpose of this system, it will be defined as Integral Time Absolute Error (ITAE). This is a performance criterion used to evaluate the effectiveness of a PID controller in regulating a process variable to a desired set point. It is a measure of the time-weighted absolute deviation of the controlled variable from the set point, integrated over time.

The ITAE criterion takes into account both the magnitude and duration of the error between the set point and the process variable, making it a popular choice for optimising PID controller constants. The equation for ITAE is given by:

$$ITAE = \int_0^n t * |SP - PV(t)|dt$$

where SP is the set point, PV(t) is the process variable at time t, and the integral is taken over time of our sample time. The ITAE criterion is often used in conjunction with a genetic algorithm to optimize the PID controller constants. The genetic algorithm generates a population of candidate solutions, each represented by a set of PID constants, and evaluates their performance using the ITAE criterion. The algorithm then uses selection, crossover, and mutation operations to evolve the population towards an optimal solution.

The goal of the genetic algorithm is to find the set of PID constants that minimize the ITAE criterion. This means that the controller will regulate the process variable to the set point with the least amount of time-weighted absolute deviation. By minimizing the ITAE criterion, the controller will respond quickly to changes in the set point while maintaining stable control of the process variable.

To achieve the goal of maintaining a distance of 15cm from the wall, the robot can be equipped with a PID controller that uses the ITAE criterion to optimize its PID controller constants. This simulation was run using various random starting distances from the setpoint, which resulted in finding optimized PID constants. The result of this optimization can be seen in a graph that shows the performance of the PID controller with these constants:

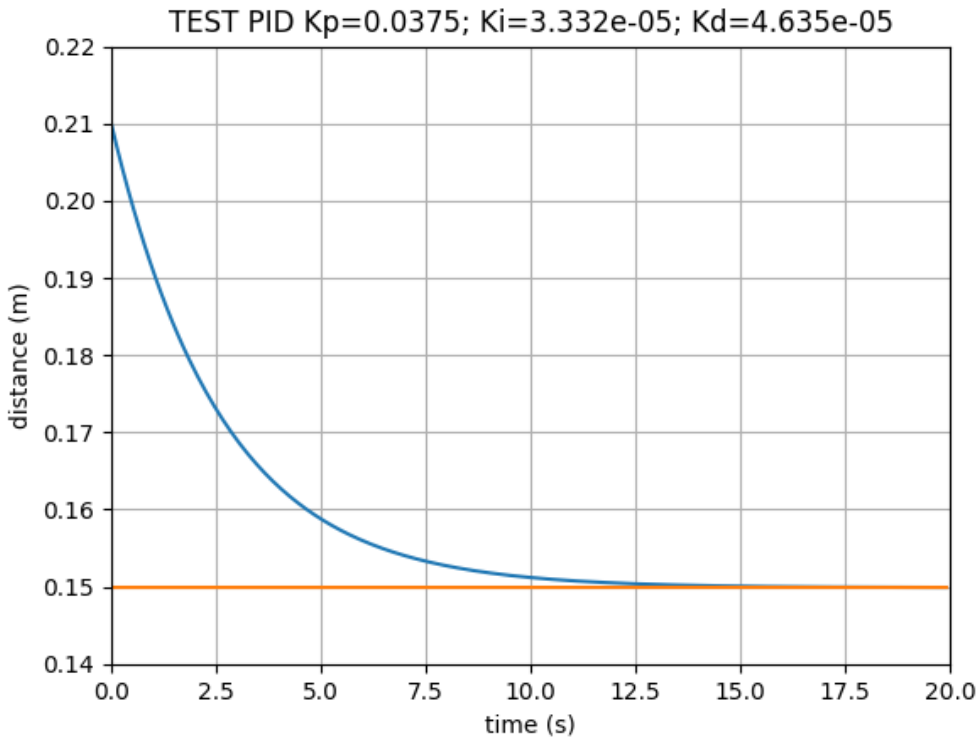

Figure 32: Simulated Run with Genetic Algorithm Constants

With a population size of 100, and with 10000 generations, the genetic algorithm produced this solution in under 25 seconds, showing that this is a quick and easy way to determine these constants when they are needed. It can be plainly seen that we have a marked improvement from before, converging within 1 second of run-time. Now instead of manually configured magic numbers for each robot, we can use this algorithm to determine the constants on a case by case basis. All this combined, we expect to make tangible improvements in both our consistency and robustness metrics. We have reduced the oscillation present through the use of a PID controller, fixed our calculations by assuming a scalene triangle, and reduced the reliance on magic numbers via machine learning.

5.5 Collision Handling Design

By Emmitt Luhning

The current collision handling design was deemed infeasible. Due to the wide array of different behaviours that the robot might need to handle depending on the state it is in when a collision occurs, a generic collision handling state is not adequate for handling all cases. The new proposed design is to introduce a new state for every existing state which requires unique collision behaviour. For example, all four states that are wall following may be able to handle a collision in the same way, but this will differ from the collision handling necessary for the robot when it is mid intersection.

5.6 Collision Handling Implementation

By Emmitt Luhning

In order to implement collision handling, each state was given a function to call should one of the robot's bumpers be activated, which results in the robot changing to the appropriate collision handling state. Due to time restriction, these behaviours were implemented on an as-needed basis, as such, not all collision behaviours for all states have been introduced.

The Intersection Reached Left Wall Lost state behaviour is an example of a state that required a unique collision implementation. As the current design expects the left turn to collide with the wall on certain occasions, logic was introduced for the Intersection Reached Left Wall Lost Collision, which requests the robot back up, and attempt to turn slightly left until it can see and follow the wall.

5.7 Hardware Design

By Jacob Charpentier

As seen in the analysis the hardware circuit design itself remained functional and met requirements however the way the system was built and attached to the robot was limiting modification and was unreliable.

5.7.1 Circuit Modifications

By Jacob Charpentier

The previous iteration had the circuitry soldered directly into a board. This worked and was reliable; however, due to the transportation of the robot the wires that attach to the solder board were damaged and due to the hardwired nature of the circuit fixing this was difficult. It was decided that until a final version of the robot is decided upon the flexibility of breadboards was preferred as it enables changes and fixes to be applied to the circuit quickly and easily. A small breadboard was chosen to reduce size and with the few components attached it functions identically to the previous iteration, but the goal of flexibility has been met through this change.

5.7.2 Chassis Design

By Jacob Charpentier

The previous iteration relied on a small 3D printed block to be attached to the robot at a perfect distance and angle relative to the center of the robot. The analysis in previous sections highlights how this introduces human error into the system as a misplacement or dislodging of the block can cause unexpected results. To solve this a much larger chassis was designed using a 3D modelling software.

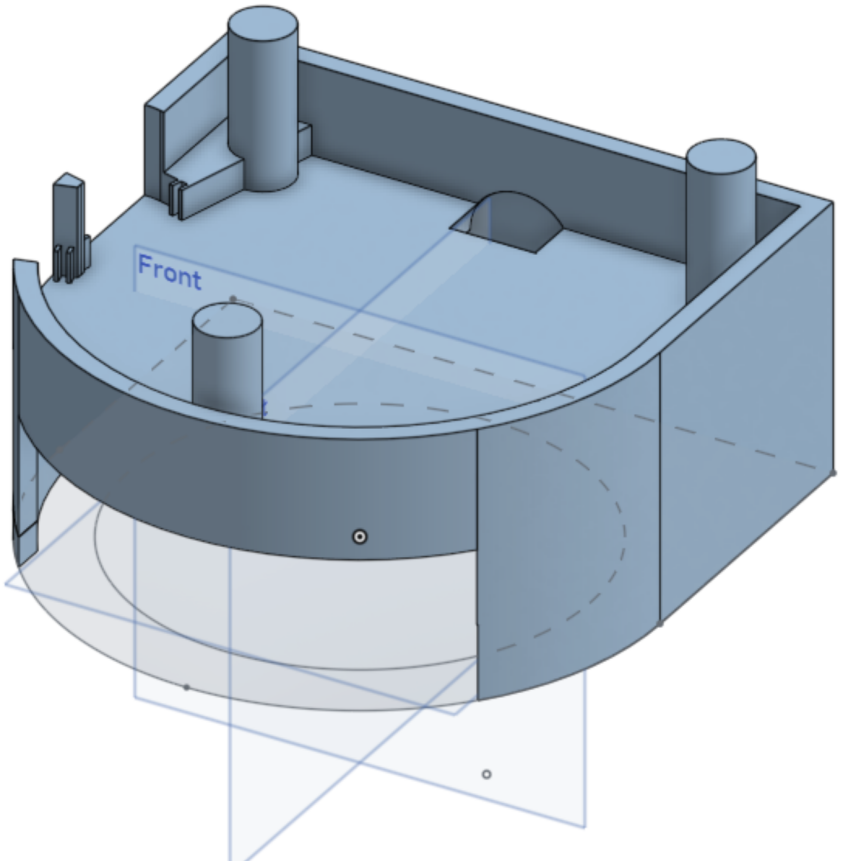

Figure 33: Chassis Design Version Two

This chassis design removes the risk of human error from the implementation by having very precise measurements which ensures that the location and angle of the IR sensors relative to the center of the system remains constant. Furthermore it adds to the functionality of the robot by providing a location to attach future additions such as mail slots or package containers by use of the three columns on the top of the model. The exact model can be further seen in Appendix C with measurements provided. Note that the model displayed is not the one printed, this is because the model printed had a couple issues which were solved in this updated model. This new model is present in the GitHub Repository under 3D Models. This updated design changes the location of the hole for the wire as well as opens up the underside making it easier to attach the wire to the robot and fit a hand to reach the buttons on top of the robot.

6.0 Testing & Evaluation

By Chase Scott

Now that the design phase is completed, the next course of action is the testing and evaluation phase. This phase involves executing test cases to ensure that the software is functional, meets the requirements, and performs as expected. The testing and evaluation phase is crucial for identifying and fixing any defects or issues that might have been introduced during the development phase. In the following paragraphs, we will explore the various testing techniques and evaluation methodologies used during the project.

6.1 Wall Following

By Chase Scott

In order to test the IR sensor's perception, it is necessary to listen in on the actions node, which is the perception node that publishes data received from the IR Sensors. This data is then used by the actions node to determine what action the robot should take based on its distance from the wall. In order to accomplish this, a PerceptionsTest node is created, which publishes to the perceptions node and subscribes to the actions node. This node is then used to publish various sensor measurements that we would expect our system to produce and verify that the actions node receives the expected data. Using this we can have simple unit tests that verify that the data being sent between nodes is consistent with what we expect.

Next, we consider integration testing of the PID development into the existing system. This involves implementation of the new wall following algorithms developed with the state machine code to allow for the robot to react intelligently to data being received from its surroundings. Using the results from the genetic algorithm as a baseline, several sample runs of

the robot were run using the constants generated at random starting points from the wall. The results of these runs were then graphed and used to compare to our requirements and testing metrics. Our main non-functional requirement we're evaluating against is:

- The robot shall be able to travel over a distance of 210 meters in approximately 18 minutes.

If we have successfully reduced the erratic oscillation present in last year's wall following behaviour, we should be much closer to this target. The results of a test run overlaid with the simulator run can be seen in the following figure:

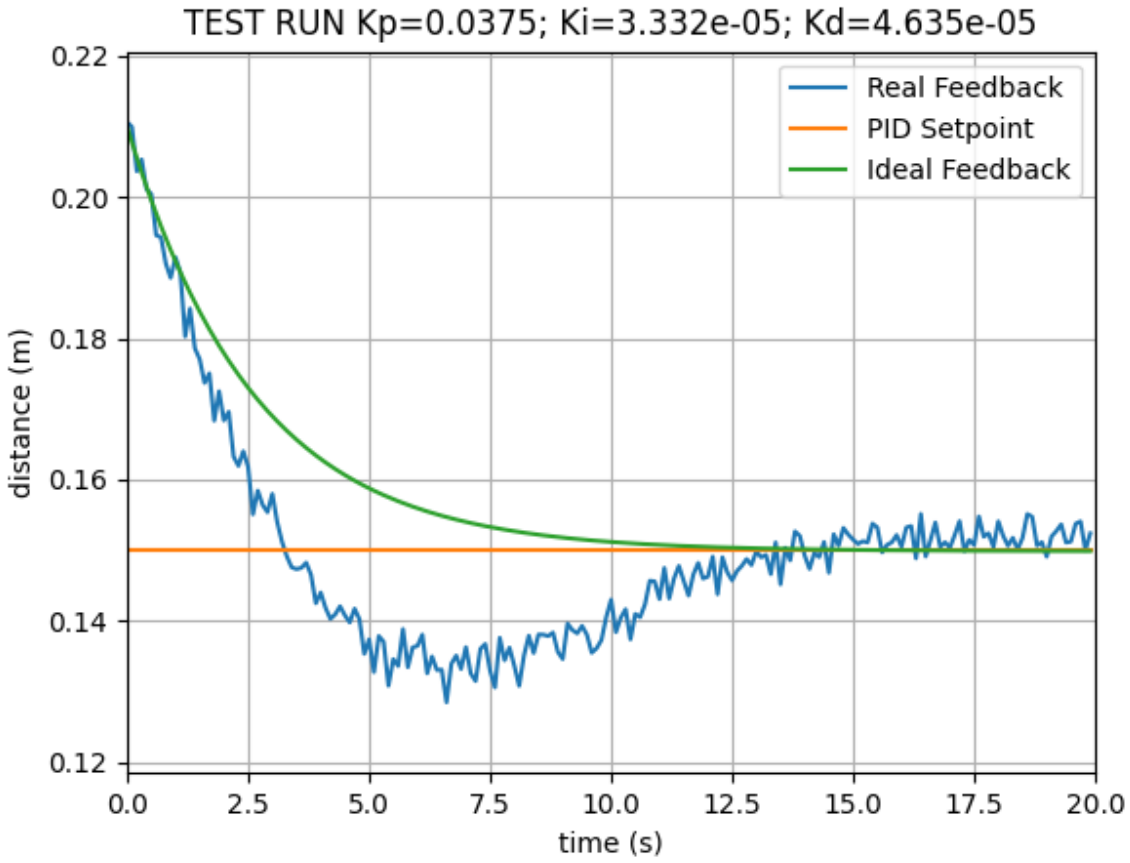

Figure 34: Real Run vs. Simulation Run

As we can see using the constants generated from the genetic algorithm, the actual behaviour of the robot differs in the approach to the setpoint. There appears to be slight overshoot on approach, but they reach steady state in approximately the same time frame. The reason for this overshoot could be due to many factors, including hardware limitations of the Pi, inconsistent terrain, and inaccurate turning from the Roomba. We can also see from this plot that there is a significant amount of noise in the IR Sensor readings, which can lead to inaccurate corrections from the PID controller. This was mediated by noise dampening code, but it still has a negative side effect optimization of the controller.

Lastly, concerning our travel time requirement, velocity traversing a section of wall over several runs was collected at random starting positions and graphed in a histogram. This was used to discern the average velocity and standard deviation across trail runs.

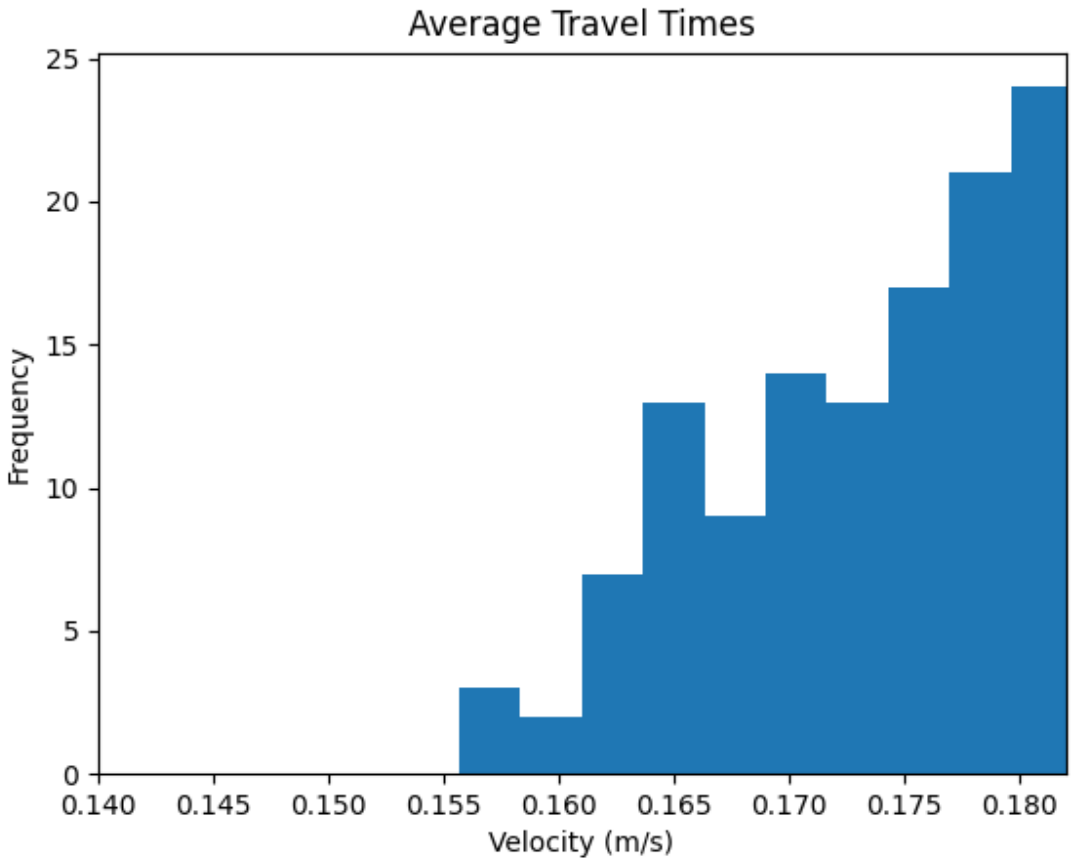

Figure 35: Average Wall Velocity Histogram

From this experiment, we get an average velocity of 0.1794 m/s with a standard deviation of 0.0103m. This means our travel time is around 19.5 minutes, which is only an 8.39% error from our requirement goal, which shows we have made great strides in increasing the performance and reliability of the wall following behaviour through the use of the PID controller. However, there is still room to improve as will be discussed further in section 7.x, along with solutions to the issues raised earlier.

6.2 Intersection Handling

6.2.1 Beacon Placement

By Emmitt Luhning and Favour Olotu

As the number of beacons chosen at an intersection differed from the previously proposed design, it was necessary to test that the single beacon was adequate for triggering a state change, both to an intersection handling state, and then out of an intersection handling state when the intersection is passed. To test this ten runs were made, with the robot placed such that it would approach and then pass a beacon, and the states of the robot were recorded throughout the traversal. For all ten runs, the robot successfully detected the beacon and transitioned to the appropriate state. This confirmed that the use of one beacon is adequate for garnering information about the robot's location in an intersection.

6.2 Turns

By Emmitt Luhning

To test turns, an environment was mocked in order to test the robot at an intersection, and modify the direction it would turn. For each turn, ten runs were executed.

Right Turn

Table 20: Intersection Tests for Right Turn Results

Run	Result	Reason for failure (if any)
1	Success	
2	Success	
3	Success	
4	Failure	Collided with wall

5	Success	
6	Success	
7	Success	
8	Success	
9	Success	
10	Success	

Straight

The straight through movement was tested for 5 runs with a two metre gap between the two walls, and 5 runs with a 3 metre gap.

Table 21: Intersection Tests for Pass Through Results - 2 Metre Gap

Run	**Result**	**Reason for failure (if any)**
1	Success	
2	Success	
3	Success	
4	Success	
5	Success	

Table 22: Intersection Tests for Pass Through Results - 3 Metre Gap

Run	**Result**	**Reason for failure (if any)**
1	Success	
2	Success	
3	Failure	Wall not found after intersection
4	Success	

5	Failure	Wall not found after intersection

Left Turn

The left turn was also run with two different hallway sizes. This time, the target hallway varied in width. The first 5 runs were tested with a 1 metre hallway width, and the following 5 were run with a 2 metre hallway width. With the hallway width set to 2 metres, all of the turns were successful, while none of the turns were successful for the 1 metre hallway width. This was caused by the robot not turning fast enough. It was trivial to change the turn speed so that a 1 metre hallway width was viable, but this resulted in the robot turning around in larger hallways, as it would not get close enough to the target wall to transition out of.

6.3 Navigation

By Favour Olotu and Emmitt Luhning

For testing, unit tests were written for the ROS nodes responsible for the navigation. The unit tests stubbed the expected incoming messages to the nodes and verified their responses are as expected, and were successful in demonstrating that the navigation node was able to build a suitable map for navigating from one point in the map to another. To understand the communication between the components figure 36 below shows the communication channel as a ROS node diagram:

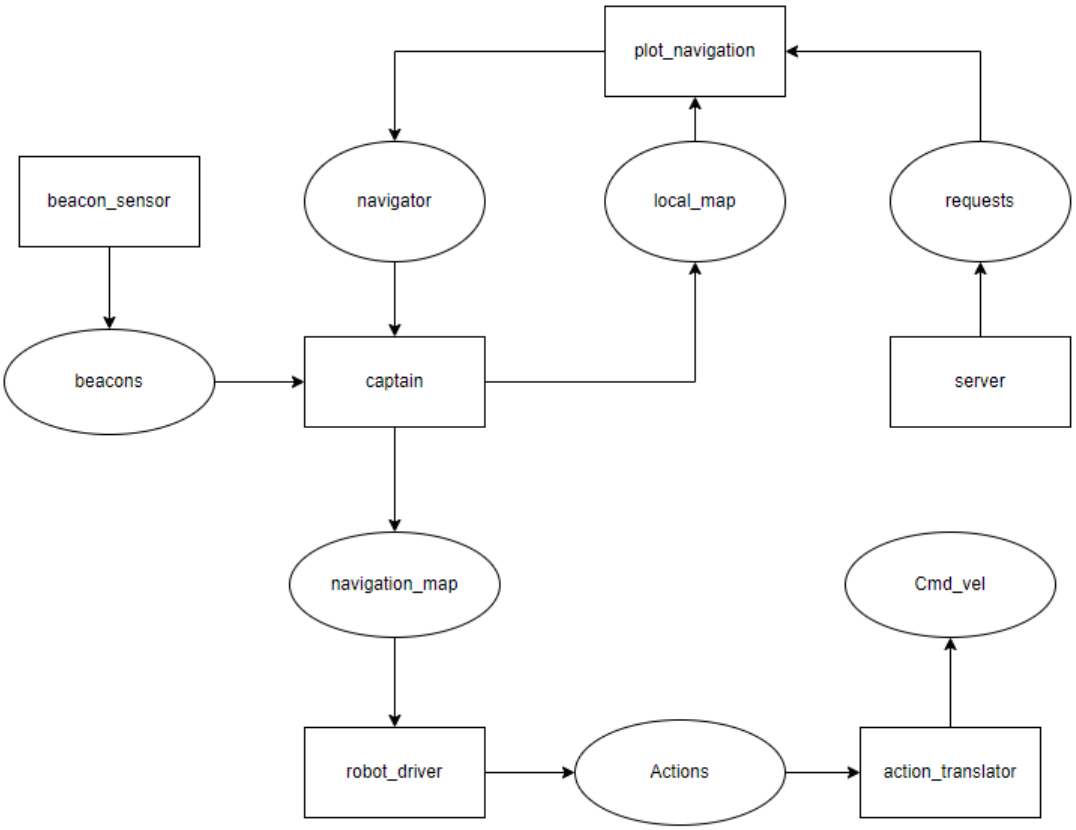

Figure 36: Communication ROS Nodes Responsible for the Tunnel Navigation

Figure 36 above shows the communication between the nodes required for the tunnel navigation functionality. The rectangle boxes represent the nodes and the circular boxes represent the topics (channel of communication). For a closer look at the type of unit tests performed, refer to figure 37 below. The figure shows the output logs of the captain's node test. It shows the type of messages sent to the captain node and checks if the response published by the captain node is as expected. A similar test was performed on the plot_navigation, beacon_sensor and robot_driver nodes.

```
ubuntu@DESKTOP-N79PPPO:~/create_ws$ ros2 launch mail_delivery_robot test_captain_node.launch.py
[INFO] [launch]: All log files can be found below /home/ubuntu/.ros/log/2023-04-12-14-53-56-789282-DESKTOP-N79PPPO-3770
[INFO] [launch]: Default logging verbosity is set to INFO
[INFO] [captain_test-1]: process started with pid [3772]
[INFO] [captain-2]: process started with pid [3774]
[captain_test-1] [INFO] [1681325637.303068771] [tests.captain_test]: Executing tests...
[captain_test-1] [INFO] [1681325642.305239427] [tests.captain_test]: Sending -2 straight- navigation message
[captain_test-1] [INFO] [1681325642.307992227] [tests.captain_test]: TEST 1 PASSED: Captain Navigation command as expect
ed
[captain_test-1] [INFO] [1681325647.304363550] [tests.captain_test]: Sending -2 right- navigation message
[captain_test-1] [INFO] [1681325647.306244550] [tests.captain_test]: TEST 2 PASSED: Captain Navigation command as expect
ed
[captain_test-1] [INFO] [1681325652.305588372] [tests.captain_test]: Sending -2 u-turn- navigation message
[captain_test-1] [INFO] [1681325652.309107472] [tests.captain_test]: TEST 3 PASSED: Captain Navigation command as expect
ed
[captain_test-1] [INFO] [1681325657.306420288] [tests.captain_test]: Sending -2 destination- navigation message
[captain_test-1] [INFO] [1681325657.312098588] [tests.captain_test]: TEST 4 PASSED: Captain Navigation command as expect
ed
[captain_test-1] [INFO] [1681325662.306583443] [tests.captain_test]: TESTS PASSED SUCCESSFULLY
```

Figure 37: Unit Test for the Captain Node

The next step was confirming the presence of a single beacon could trigger a state change in the robot driver. It was later tested on the robot that this resulted in the correct message and consequently the correct state change. With ten runs for three different directions, each requiring a different type of turn at the intersection, the robot successfully transitioned to the appropriate state for handling the intersection each time.

7.0 Reflection & Conclusions

7.1 IR Sensors and Wall Following

By Chase Scott

The problem we aimed to solve in regards to wall-following was the inconsistent and oscillatory behaviour of the current implementation. To address this issue, our proposed solution involved using a PID controller paired with a genetic algorithm to optimise the controller's constants. By using a genetic algorithm, the system could find the optimal constants for the PID controller that would ensure consistency, robustness, and scalability in wall following.

Throughout the analysis and implementation of the IR Sensor wall-following solution, several issues were encountered. The most prevalent errors occurred from the reading received from the IR Sensors. They were found to be extremely noisy and just as inconsistent. Even with extensive care put into denoising the readings, there is still very little reliability and consistency in their readings. In addition to this, the analog to digital converter used seems to clamp the values in unpredictable ways, which directly affects the effective range of the wall following. It is highly recommended that future groups investigate a more powerful alternative to the IR Sensors, as the scope of the design problem is not adequately covered by their capabilities. The PID controller module was designed with extensibility and modularity in mind, so it should be able to interface easily with any new sensors that are used. Overall, this approach offers a significant improvement to the current implementation of wall following and provides a solid foundation for future research and development.

7.2 Navigation

By Favour Olotu

The problem addressed for navigation was the lack of an implementation for the described design. We successfully implemented the navigation design proposed, resulting in an effective module that is able to create routes from one building to another in the tunnel, and communicate those readings to the state machine. Within the scope of this project, the current navigation implementation meets all of the requirements it aimed at fulfilling, and it is highly recommended that future work is built with the current navigation system in mind. A suggestion for further improvement, would be the use of the google cartographer software to create an accurate to scale map of the tunnel from the point-of-view from the robot. This will help provide a better estimation of the travel times and the location of the robot during travel. The additional hardware needed to accomplish this is a LIDAR.

7.3 Turns and Intersection Handling

By Emmitt Luhning

The intersection handling solution derived solved some of the problems that needed to be addressed. The main issue faced was the absence of interoperability between components, and a consequential lack of actual intersection handling capability. Throughout the project, we integrated the state machine with the navigation system such that the robot can determine how to behave in an intersection. We also successfully introduced a large array of new states, which implemented solutions for the logic controlling the robot's movement for an intersection pass through and right turn, as evidenced by the tests.

The occasional failures observed while the robot performed these operations were the result of the same root issue that lead to impairments in the wall following capability, mainly the assumption that the IR sensors utilised would provide a larger range, and more accurate information than was observed. It is heavily recommended that future work introduce more robust sensing, and look at alternatives to IR for wall following, rather than attempt to tweak the logic of these two turn types to fit the current configuration, as the limitations caused by the current sensing setup are severe.

The left turn logic proved to be inadequate through analysis of our design, as it was only sufficient for certain intersection sizes, and hallway widths. This is due to the same issue faced by the previous implementation, being that the robot cannot adjust the speed of its turn in order to adjust to its current position, and therefore relies on a predetermined value. Being able to see the wall from a greater distance, or the ability to sense multiple walls would assist in solving these problems. As such, It is recommended that additional sensors be used such that the robot has a clearer idea of its position relative to other physical entities while performing the turn. Additionally, it is not recommended that additional beacons are looked at as a solution to this problem, and should only be used for navigation, as they do not provide accurate enough information for a precise understanding of the robot's position.

7.4 Collision Behaviour

By Emmitt Luhning

The problem addressed with regards to collision behaviour was that the monolithic design was not adequate for all cases, and did not function in practice. We began solving this problem by implementing some individual collision behaviours related to specific states, such as the left

turn. It is recommended that future work aims to increase the number of collision handling states, such that all unique cases are addressed.

During the design of this project, there was an attempt to use the built in collision handling behaviour of the robot, but this is not recommended. It was found that in order to activate the onboard collision handling behaviour, the mode of the robot had to be changed, which gives up control of the robot to the onboard software, and as the robot was no longer under the control of our software, it was not possible to regain control, or change state once the collision was avoided.

7.5 Docking

By Jacob Charpentier

The problem of automated docking was addressed early in the development process. Through testing it was determined that the docking functionality can work independently when within a short range. We began solving this problem by adding a docking event and triggering that when the docking station was detected, however integrations involving the automated docking feature were made but always resulted in the robot either missing the station or hitting it instead of docking.

For future iterations it is recommended that a beacon be placed above each docking station to increase the range of the destination signal, when the signal strength reaches a maximum or begins to decrease the automated docking function could then be activated in hopes that the robot would be close enough and positioned correctly to begin docking.

7.6 Conclusion

By Jacob Charpentier and Chase Scott

The goal of the project was to develop an efficient autonomous mail delivery system that could deliver mail through the Carleton University tunnel system. Several challenges were encountered, including navigating tunnels and intersections, avoiding collisions, and user interaction with the system. To overcome these challenges, a complex state machine and navigational system was used to enable live responses to sensor information, and a map of the tunnel system was hard coded using Bluetooth beacons. While some challenges, such as navigating complex sections of the tunnels and full-function web application, remain unsolved, the group has accomplished the successful implementation of an autonomous system that can navigate, wall follow, and handle intersections while remaining safe and reliable. The robot design was also designed with scalability in mind to prepare for future additions. Overall, the project was a valuable learning experience for us, and we were able to make significant strides towards the development of an efficient autonomous mail delivery system. We hope that the work we provided and recommendations for future work can help guide this product to the one we imagine in the future.

References

[1] “Create 2 Robot.” *IRobot Education*, https://edu.irobot.com/what-we-offer/create-robot (accessed January 10, 2023).

[2] “Office of Quality Initiatives.” *Healthy Workplace*, https://carleton.ca/healthy-workplace/lunchtime-activities/outdoor-walking-group/tunnels/.

[3] Onyedinma, Chidiebere, Gavigan, Patrick & Esfandiari, Babak. (2020). Toward Campus Mail Delivery Using BDI. Journal of Sensor and Actuator Networks. 9. 56. 10.3390/jsan9040056.

[4] Perron, Jacob. “create_autonomy.” *Wiki.ros.org*, http://wiki.ros.org/create_autonomy.

[5] Raspberry Pi. “Buy A Raspberry Pi 4 Model B.” *Raspberry Pi*, https://www.raspberrypi.com/products/raspberry-pi-4-model-b/.

[6] “Ros 2 Documentation.” *ROS 2 Documentation - ROS 2 Documentation: Rolling Documentation*, https://docs.ros.org/en/rolling/index.html.

[7] Wicklund, Stephen, and Emily Clarke. Ottawa, Ontario, 2022, pp. 1–98, *Carleton University Mail Delivery Robot*.

[8] Richard Van Loon et al., "Laboratory Health and Safety Manual" Environmental Health and Safety Services, Ottawa, Ontario, 2000.

Appendices

A Environment Setup

Setup Operating System

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.

Establish the Host Name and User

1. sudo su - root
2. hostnamectl set-hostname <new hostname>
3. adduser robot
4. su - robot

Configure Wi-fi

https://itsfoss.com/connect-wifi-terminal-ubuntu/

Configure ROS

Install from instructions here:
https://docs.ros.org/en/foxy/Installation/Ubuntu-Install-Debians.html

Followed by these commands

1. sudo rosdep init
2. source /opt/ros/foxy/setup.bash (must be run with every new terminal. Recommend adding this to .bashrc)

Configure Create Autonomy

https://github.com/AutonomyLab/create_robot/tree/foxy

Enable i2c

https://askubuntu.com/questions/1273700/enable-spi-and-i2c-on-ubuntu-20-04-raspberry-pi

Setup Local CodeBase

1. cd ~/create_ws/src
2. Git clone https://github.com/Em-kale/carleton-mail-delivery-robot.git
3. sudo apt-get install python3-pip
4. pip3 install Adafruit_ADS1x15
5. cd ~/create_ws
6. colcon build

Run Code

1. sudo -s (beacons require root privileges)
1. source /opt/ros/foxy/setup.bash
2. source /home/ubuntu/create_ws/install/setup.bash
3. cd create_ws
4. ros2 launch mail_delivery_robot robot.launch.py 'robot_model:=CREATE_2'

B Running Tests

Each test-suite has a provided launch file, which launches the test node along with the nodes that tests are being run on. To run the tests, ensure that your environment is configured as shown in the setup instructions and run the following command, replacing the test name with the test you are trying to run:

```
source /opt/ros/foxy/setup.bash
source ~/create_ws/install/setup.bash
colcon build
ros2 launch mail_delivery_robot <testname>.launch.py
```

Creating New Tests

The process for adding new tests to the project involve three major steps:

1. Create test node
2. Configure setup.py
3. Create launch file

To create a new test, add a new node to the tests file with the test code you wish to execute. After which, modify the setup.py file in the root of the mail_delivery_robot folder to include your tests. In particular, you need to add to the py_modules and entry_points sections. It should look like this when you are finished, replacing your_test_name with the actual name of your test file:

```
py_modules=["tests.your_test_name", "tests.robot_driver_test","preceptions.IRDistanceCalc","control.act
entry_points={
    'console_scripts': [
        'your_test_name = tests.your_test_name:main'
        'robot_driver_test = tests.robot_driver_test:main',
        'action_translator_test = tests.action_translator_test:main',
        'IRSensor = preceptions.IRDistanceCalc:main',
```

Figure 38: Sample Entry Point Addition

Next, the launch file needs to be made. You need to include both your test nodes, and the nodes that it depends on. The file that the name of the file must end in .launch.py. It is also imperative that remappings are done correctly. Any action that is published or subscribed to from the tests folder, or any folder other than its origin will need to be remapped to the folder of the nodes it is trying to communicate with. An example for the action translator test is provided below:

```python
from launch import LaunchDescription
from launch_ros.actions import Node

def generate_launch_description():

    return LaunchDescription([
            Node(package='mail_delivery_robot',
            namespace='tests',
            executable='action_translator_test',
            name='action_translator_test',
            output='log',
            remappings=[('/tests/actions', '/control/actions')]),
            Node(package='mail_delivery_robot',
            namespace='control',
            executable='action_translator',
            name='action_translator',
            remappings=[('/control/cmd_vel', '/tests/cmd_vel')]
            )
        ])
```

Figure 39: Sample Launch File

C Hardware Configuration

With the roomba facing away from the user, perform the following setup steps.

1. Attach the IR distance sensors and connected circuit to the right side of the roomba so that the distance sensors are equidistant from the center of the robot as seen below.

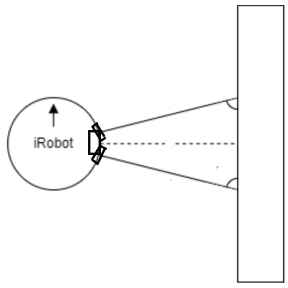

Figure 40: IR Sensor Set-up on the Roomba

2. Connect the red wire to the raspberry pi pin 1, connect the purple wire to pin 3, connect the blue wire to pin 5, and finally connect the brown wire to pin 39 as seen below in the diagram, to form the circuit shown in figure 35.

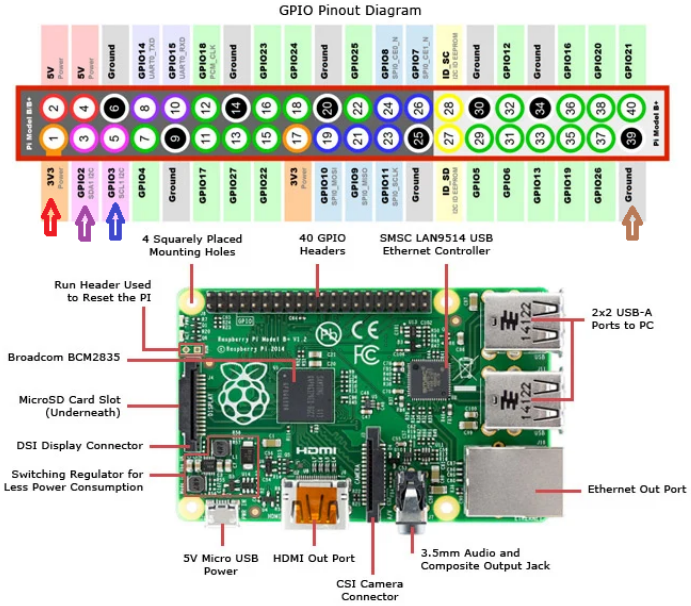

Figure 41: Raspberry Pi Connection Map

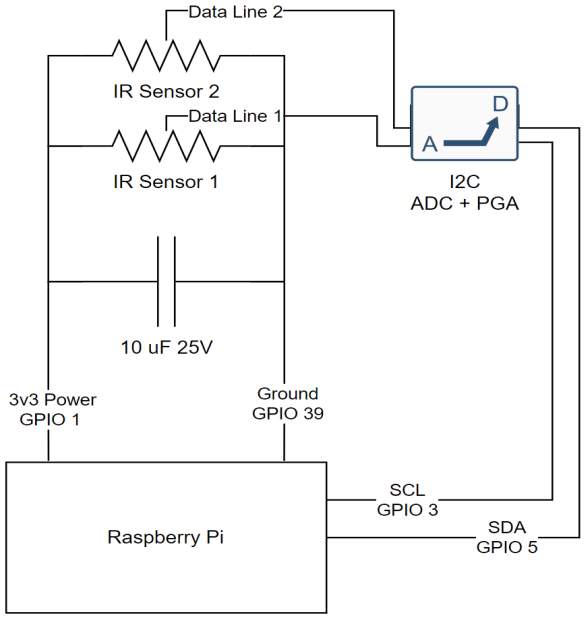

Figure 42: Circuit Diagram

3. Connect the red and black wires from the DC-DC converter to the USB-A pins as seen below.

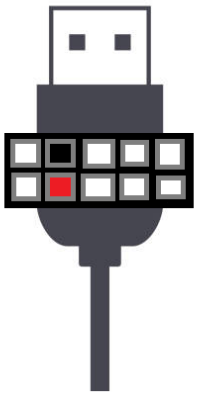

Figure 43: USB-A Pin Connection

4. If the USB-C cable is not connected to the DC-DC converter, unscrew the pins for the 5V 3A max output side and take the stripped wires from the USB-C cable and attach them. Note, grey wire to negative screw, and pink wire to positive screw. To attach the wires, securely take the interior copper filaments from one of the stripped wires, untangle them, split them down the middle, place the screw between the two strands, wrap the filaments together on the other side of the screw and fold the twisted section back under the screw.
5. Connect the USB-C cable to the Raspberry Pi.
6. Connect the USB-A to the Raspberry Pi, and the opposite side to the iRobot Create 2.

This concludes the macro setup for the hardware of the Mail Delivery Robot.

D Chassis Design

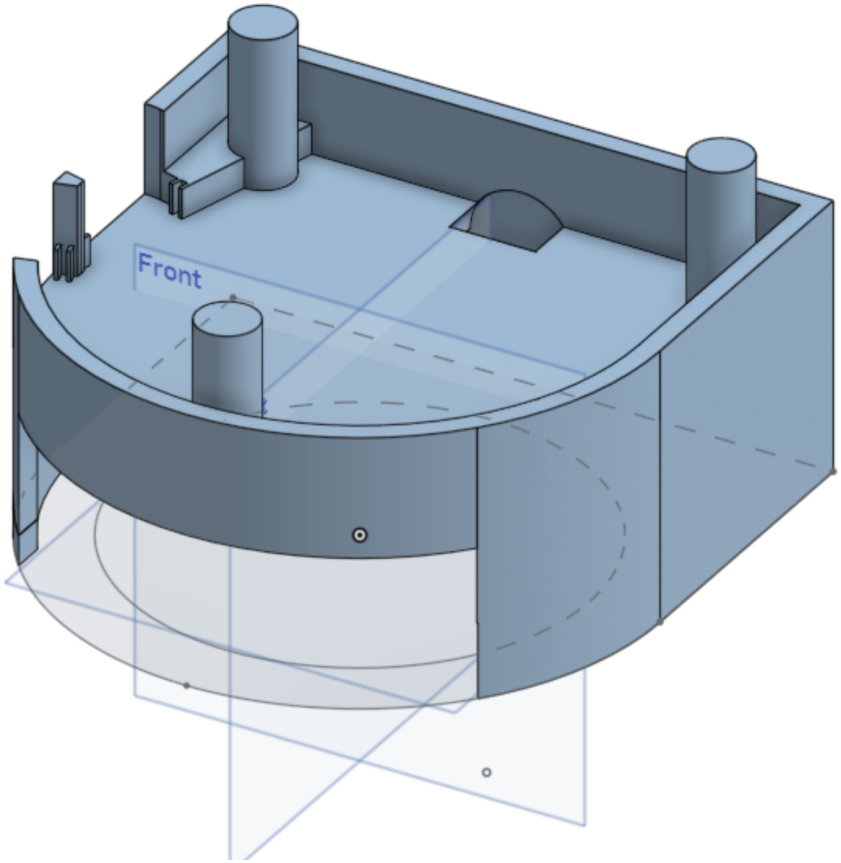

Figure 44: Chassis Design Isometric View

Note the 3 pillars protruding from the top, these have the shape and measurements seen below, alongside the chassis dimensions, and can be used to attach features to the top of the chassis. The website used to model is below, the model can be found in github under the documentation folder. https://cad.onshape.com/

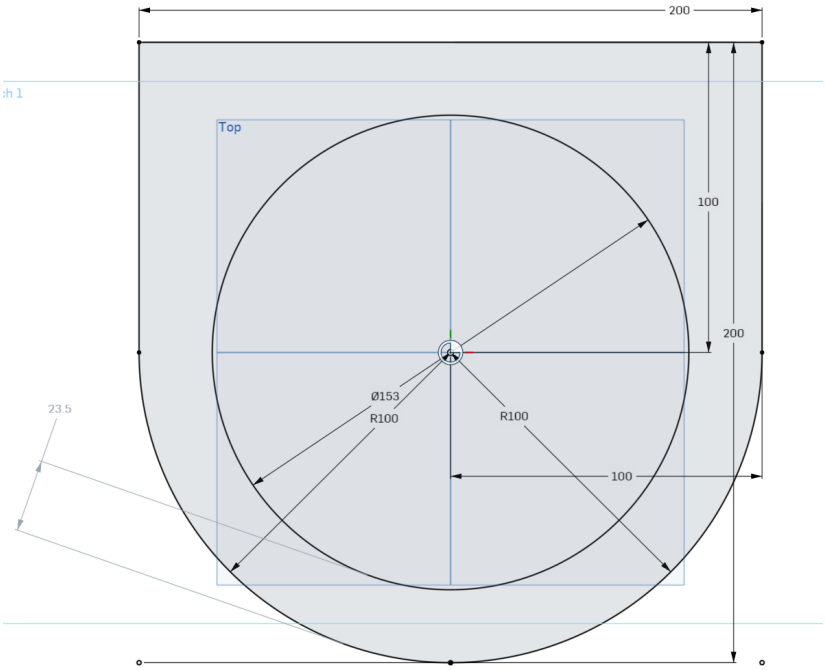

Figure 45: Chassis Design Bottom View

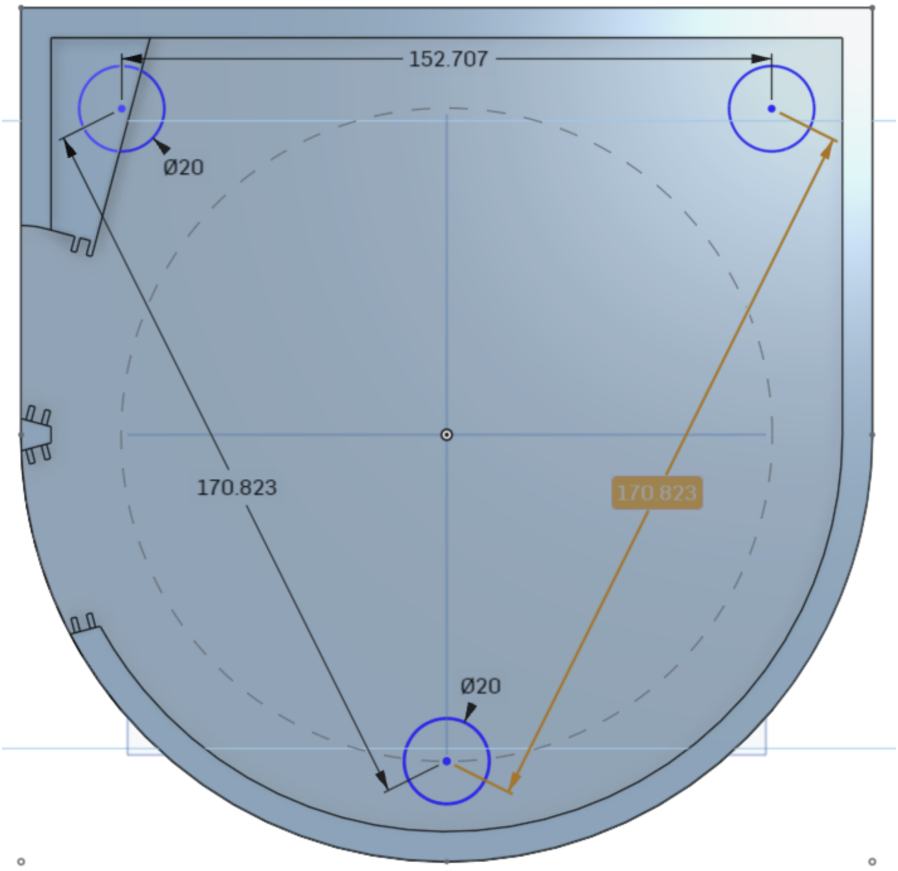

Figure 46: Chassis Design Top View