

# **Autonomous Mail Delivery Robot**

## **Project Proposal**

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

Supervisor: Dr. Babak Esfandiari

Department of Systems and Computer Engineering  
Faculty of Engineering  
Carleton University

October 30<sup>th</sup>, 2023

# Table of Contents

<b>List of Tables</b>	<b>2</b>
<b>List of Figures</b>	<b>3</b>
<b>1 Project Objectives</b>	<b>4</b>
1.1 Requirements	4
1.1.1 Functional Requirements	4
1.1.2 Non-Functional Requirements	5
1.2 Use Cases	5
1.2.1 Use Case Diagram	5
1.2.2 Individual Use Cases	6
1.3 Metrics	17
1.4 State Machine	18
1.4.1 List of States, Events, and Actions	19
1.4.2 Justifications	20
1.5 Node Structure	21
1.5.1 List of Nodes, Messages and Entities	21
1.5.2 Justifications	22
<b>2 Background</b>	<b>23</b>
2.1 Robot Operating System	23
2.2 iRobot Create	23
2.3 LiDAR Sensor	24
<b>3 Description</b>	<b>25</b>
<b>4 Justification of Suitability for Degree Program</b>	<b>26</b>
<b>5 Group Skills</b>	<b>27</b>
<b>6 Methods</b>	<b>28</b>
<b>7 Analysis</b>	<b>29</b>
7.1 Analyzing the Equipment	29
7.2 Beacon and Dock Station Placement	33
<b>8 Project Timeline</b>	<b>35</b>
8.1 Proposed Timeline	35
8.2 Gantt Chart	36
<b>9 Risks and Mitigation Strategies</b>	<b>37</b>
<b>10 List of Required Components/Facilities</b>	<b>38</b>
10.1 Justification of Purchases	38
<b>11 Documented Issues</b>	<b>41</b>
11.1 Automatic Undocking	41
11.2 Powering the entire system using the robot	42
<b>12 Minimum Viable Product</b>	<b>44</b>
<b>13 References</b>	<b>45</b>
<b>Appendix A: Chassis Design</b>	<b>47</b>

## List of Tables

Table Title	Page #
Table 1: Send Mail use case	6
Table 2: Retrieve Mail use case	7
Table 3: Navigate tunnels use case	8
Table 4: Update Status use case	9
Table 5: Handle Collisions use case	10
Table 6: Read Beacons use case	11
Table 7: Monitor System use case	12
Table 8: Dock Robot use case	13
Table 9: Create Request use case	14
Table 10: Notify Robot use case	15
Table 11: Notify User use case	16
Table 12: Measuring the speed of the robot	29
Table 13: The ability of the robot to find the wall and maintain it	29
Table 14: The ability of the robot to make a successful right turn consistently	30
Table 15: The ability of the robot to make a successful left turn consistently	30
Table 16: Measuring the beacons at known distances	31
Table 17: Measuring the strengths of the internet signal in Carleton University tunnels.	32
Table 18: Project milestones and proposed target completion dates	35
Table 19: Possible risks and their mitigation strategies	37
Table 20: List of required components/facilities, their objectives of the project, and estimated cost	38
Table 21: Comparison of sensor candidates based on power usage, range of effectiveness for both distance and direction, and price	38
Table 22: Comparison of battery pack candidates based on supplied power, size, and price	39

## List of Figures

Figure Title	Page Number
Figure 1: Use Case Diagram for the Mail Delivery Robot System	5
Figure 2: A map of the Carleton University tunnels	17
Figure 3: The proposed state machine for the system	18
Figure 4: The proposed ROS node structure for the system.	21
Figure 5: Proposed locations for the beacons and docking stations in the tunnels.	33
Figure 6: Gantt chart of proposed timeline and due dates	36
Figure 7: The chassis design for the robot holding the circuits for the robot	47
Figure 8: The design of the mailbox for the robot	47
Figure 9: The technical drawing for the robot chassis detailing its measurements	48
Figure 10: The technical drawing for the mailbox detailing its measurements	48

# 1 Project Objectives

By Max Curkovic

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

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

## 1.1 Requirements

By Max Curkovic and Bardia Parmoun

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

### 1.1.1 Functional Requirements

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

### 1.1.2 Non-Functional Requirements

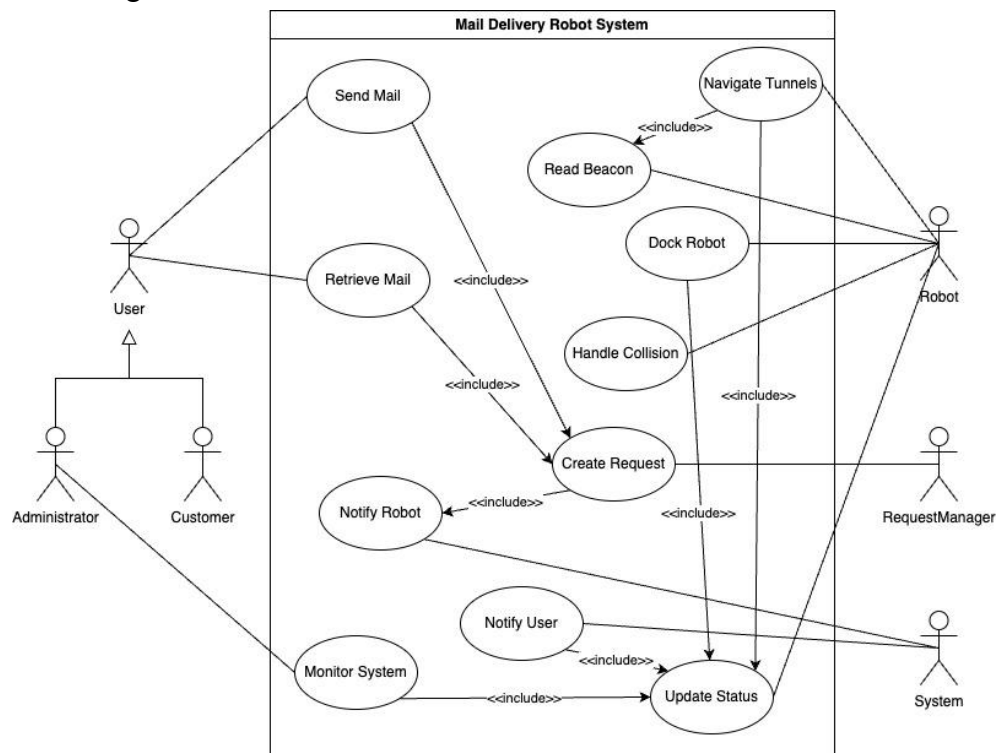
- **R12:** The robot control software shall be interoperable with the web display application.
- **R13:** The robot shall be able to travel at a speed of at least 0.15m/s through Carleton University's tunnels.
- **R14:** The beacons shall be able to be interfaced with by the robot within 14.2 meters.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

## 1.2 Use Cases

By Max Curkovic and Bardia Parmoun

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

### 1.2.1 Use Case Diagram



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

### 1.2.2 Individual Use Cases

**Table 1:** Send Mail use case

<b>Use Case Name</b>	Send Mail
<b>Brief Description</b>	The user sends a request to send mail to a destination.
<b>Primary Actor</b>	User
<b>Secondary Actor</b>	RequestManager
<b>Precondition</b>	The system is running, the robot is turned on, and the beacons are installed.
<b>Dependency</b>	INCLUDE USE CASE Create Request
<b>Basic Flow</b>	<p>Steps:</p> <ol style="list-style-type: none"><li>1. The sender opens a request to send mail.</li><li>2. The RequestManager receives and validates the request.</li><li>3. RequestManager asks for an available robot.</li><li>4. The sender places the package in the robot.</li><li>5. The system notifies the recipient of the mail.</li></ol> <p><b>PostCondition:</b> Mail is in the robot being delivered.</p>
<b>Specific Alternative Flows</b>	<p>RFS Basic Flow 2.</p> <ol style="list-style-type: none"><li>1. IF There are no available robots THEN the request will be placed in a pending queue ENDIF</li></ol> <p><b>PostCondition:</b> Request will be placed in a pending queue.</p>

**Table 2:** Retrieve Mail use case

<b>Use Case Name</b>	Retrieve Mail
<b>Brief Description</b>	The user sends a request to retrieve mail from the robot.
<b>Primary Actor</b>	User
<b>Secondary Actor</b>	RequestManager
<b>Precondition</b>	The system is running, the robot is turned on, and the beacons are installed.
<b>Dependency</b>	INCLUDE USE CASE Create Request
<b>Basic Flow</b>	<p>Steps:</p> <ol style="list-style-type: none"><li>1. Users (sender and recipient) are notified by the System that mail has arrived at the destination.</li><li>2. The recipient retrieves the mail from the robot.</li></ol> <p><b>Postcondition:</b> The recipient User has successfully received mail from the Robot, sent by the sender User.</p>
<b>Specific Alternative Flows</b>	<p>RFS Basic Flow 2.</p> <p>Steps:</p> <ol style="list-style-type: none"><li>1. IF The Recipient User does not retrieve the mail from the robot THEN record the error and provide the robot status ENDIF</li></ol> <p><b>Postcondition:</b> The robot returns to its home dock and proceeds with other requests.</p>



**Table 3:** Navigate tunnels use case

<b>Use Case Name</b>	Navigate Tunnels
<b>Brief Description</b>	The robot is able to navigate the tunnels based on its given path. This includes navigating various intersections and turns.
<b>Primary Actor</b>	Robot
<b>Secondary Actor</b>	System
<b>Precondition</b>	The system is running, the robot is turned on and the beacons are installed.
<b>Dependency</b>	INCLUDE USE CASE Read Beacon INCLUDE USE CASE Update Status
<b>Basic Flow</b>	<p>Steps:</p> <ol style="list-style-type: none"> <li>1. The system provides the robot with a destination and a navigation path.</li> <li>2. The robot uses its sensors to receive information from its surroundings.</li> <li>3. The robot uses the beacon sensors to determine its required next action.</li> <li>4. The robot calculates its next move</li> <li>5. The robot reaches its destination.</li> </ol> <p><b>Postcondition:</b> The robot successfully navigated the hallway.</p>
<b>Specific Alternative Flows</b>	<p>RFS Basic Flow 2.</p> <p>Steps:</p> <ol style="list-style-type: none"> <li>1. IF The robot is unable to detect the sensor THEN update the system ENDIF</li> </ol> <p><b>Postcondition:</b> The robot notifies the system.</p>
<b>Specific Alternative Flows</b>	<p>RFS Basic Flow 3.</p> <p>Steps:</p> <ol style="list-style-type: none"> <li>1. IF The robot is unable to detect the beacons THEN update the system ENDIF</li> </ol> <p><b>Postcondition:</b> The robot notifies the system.</p>

**Table 4:** Update Status use case

<b>Use Case Name</b>	Update Status
<b>Brief Description</b>	The robot is able to send a status update to the System.
<b>Primary Actor</b>	Robot
<b>Secondary Actor</b>	System
<b>Precondition</b>	The system is running, the robot is turned on, the beacons are installed and a delivery must be in progress.
<b>Dependency</b>	N/A
<b>Basic Flow</b>	<p>Steps:</p> <ol style="list-style-type: none"><li>1. The sender user requests the System for a status update.</li><li>2. The robot sends a positional update, as well as a delivery status, to the System.</li><li>3. The system notifies the sender user.</li></ol> <p><b>Postcondition:</b> The sender user and the System are now both aware of the delivery status.</p>
<b>Specific Alternative Flows</b>	<p>RFS Basic Flow 3.</p> <p>Steps:</p> <ol style="list-style-type: none"><li>1. IF there is no Internet in the Carleton tunnels, THEN save it as a cache and try again ENDIF.</li></ol> <p><b>Postcondition:</b> The sender user will still be updated on the delivery status.</p>

**Table 5:** Handle Collisions use case

<b>Use Case Name</b>	Handle Collisions
<b>Brief Description</b>	The Robot collides with an object (wall, cart, etc.) and re-oriens itself.
<b>Primary Actor</b>	Robot
<b>Secondary Actor</b>	N/A
<b>Precondition</b>	The system is running, the robot is moving through the tunnels.
<b>Dependency</b>	N/A
<b>Basic Flow</b>	<p>Steps:</p> <ol style="list-style-type: none"><li>1. The Robot collides with an object.</li><li>2. The Robot bumpers detect that a collision has occurred.</li><li>3. The Robot reverses.</li><li>4. The Robot turns away from the object and re-oriens in the right direction.</li></ol> <p><b>Postcondition:</b> The robot has successfully navigated through the collision.</p>
<b>Specific Alternative Flows</b>	N/A

**Table 6:** Read Beacons use case

<b>Use Case Name</b>	Read Beacons
<b>Brief Description</b>	The robot detects a beacon signal traveling through the tunnels.
<b>Primary Actor</b>	Robot
<b>Secondary Actor</b>	N/A
<b>Precondition</b>	The system is running and the robot is moving.
<b>Dependency</b>	N/A
<b>Basic Flow</b>	<p>Steps:</p> <ol style="list-style-type: none"><li>1. The robot detects the beacon and determines what intersection it is coming from.</li><li>2. The robot logs the information on the system.</li></ol> <p><b>Postcondition:</b> The robot has successfully provided the system with its beacon data.</p>
<b>Specific Alternative Flows</b>	N/A

**Table 7:** Monitor System use case

<b>Use Case Name</b>	Monitor System
<b>Brief Description</b>	An administrator monitors the current state of the system.
<b>Primary Actor</b>	Admin
<b>Secondary Actor</b>	Robot
<b>Precondition</b>	The system is running, and the robot is operational.
<b>Dependency</b>	INCLUDE USE CASE Update Status
<b>Basic Flow</b>	<p>Steps:</p> <ol style="list-style-type: none"><li>1. USE CASE Update Status is initiated.</li><li>2. The administrator requests to view the current state of the system.</li><li>3. The system replies with the current state.</li></ol> <p><b>Postcondition:</b> The administrator is informed of the current system state.</p>
<b>Specific Alternative Flows</b>	<p>RFS Basic Flow 2.</p> <p>Steps:</p> <ol style="list-style-type: none"><li>1. IF the system is not running OR the robot is not operational, THEN send an error ENDIF</li></ol> <p><b>Postcondition:</b> The administrator is informed of the current system state.</p>

**Table 8:** Dock Robot use case

<b>Use Case Name</b>	Dock Robot
<b>Brief Description</b>	The robot is near a docking station and determines that it needs to dock.
<b>Primary Actor</b>	Robot
<b>Secondary Actor</b>	N/A
<b>Precondition</b>	The system is running and a docking station has been encountered.
<b>Dependency</b>	INCLUDE USE CASE Update Status
<b>Basic Flow</b>	<p>Steps:</p> <ol style="list-style-type: none"><li>1. The robot gets near a docking station.</li><li>2. The robot determines that it needs to dock at the specific docking station.</li><li>3. The robot mounts the docking station.</li><li>4. The robot updates the system about its behavior.</li></ol> <p><b>Postcondition:</b> The robot is charging and the system is notified.</p>
<b>Specific Alternative Flows</b>	<p>RFS Basic Flow 3.</p> <p>Steps:</p> <ol style="list-style-type: none"><li>1. IF The robot is unable to mount the docking station THEN send an error ENDIF</li></ol> <p><b>Postcondition:</b> The system is notified of the error.</p>

**Table 9:** Create Request use case

<b>Use Case Name</b>	Create Request
<b>Brief Description</b>	The request manager creates and validates a request for the sender user.
<b>Primary Actor</b>	RequestManager
<b>Secondary Actor</b>	Robot
<b>Precondition</b>	The system is running, and the robot is operational.
<b>Dependency</b>	INCLUDE USE CASE NotifyRobot
<b>Basic Flow</b>	<p>Steps:</p> <ol style="list-style-type: none"><li>1. The sender user creates a request to the request manager to send or receive mail.</li><li>2. The request manager validates the user's request.</li><li>3. USE CASE Notify Robot is initiated to inform the robot of the request.</li></ol> <p><b>Postcondition:</b> The robot is notified of the request and will proceed to fulfill it.</p>
<b>Specific Alternative Flows</b>	<p>RFS Basic Flow 3.</p> <p>Steps:</p> <ol style="list-style-type: none"><li>1. IF the robot is not operational, THEN inform the user to try again at a later time. ENDIF</li></ol> <p><b>Postcondition:</b> The sender user is notified that the robot is not currently operational and will have to wait in order to put in a request.</p>

**Table 10:** Notify Robot use case

<b>Use Case Name</b>	Notify Robot
<b>Brief Description</b>	The system notifies the robot of new updates. This includes new requests, navigation data, etc.
<b>Primary Actor</b>	System
<b>Secondary Actor</b>	Robot
<b>Precondition</b>	The system is running and the robot is operational.
<b>Dependency</b>	N/A
<b>Basic Flow</b>	<p>Steps:</p> <ol style="list-style-type: none"><li>1. The system prepares a new update for the robot.</li><li>2. The system sends the update to the robot.</li></ol> <p><b>Postcondition:</b> The robot receives the update and acts accordingly.</p>
<b>Specific Alternative Flows</b>	<p>RFS Basic Flow 2.</p> <p>Steps:</p> <ol style="list-style-type: none"><li>1. IF The system is unable to contact the robot THEN inform the user and terminate the request ENDIF</li></ol> <p><b>Postcondition:</b> The system notifies the user of the error.</p>



**Table 11:** Notify User use case

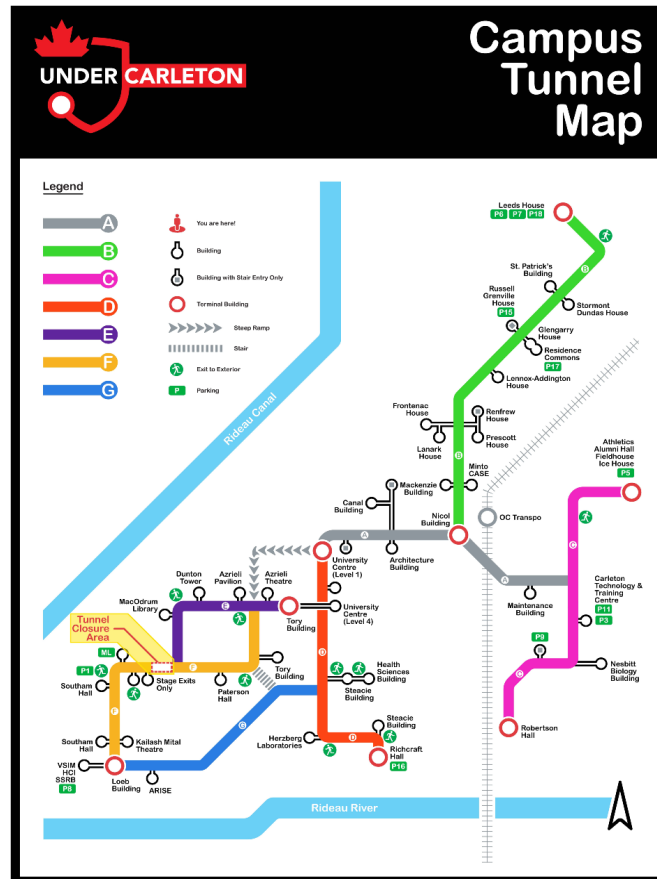
<b>Use Case Name</b>	Notify User
<b>Brief Description</b>	The user is notified of any updates to the robot, their delivery, or the system itself.
<b>Primary Actor</b>	System
<b>Secondary Actor</b>	Robot
<b>Precondition</b>	The system is running and the robot is operational.
<b>Dependency</b>	INCLUDE USE CASE Update Status
<b>Basic Flow</b>	<p>Steps:</p> <ol style="list-style-type: none"><li>1. USE CASE Update Status is initiated.</li><li>2. The system posts the updated results for the user.</li></ol> <p><b>Postcondition:</b> The user is now informed of any updates to the robot.</p>
<b>Specific Alternative Flows</b>	<p>RFS Basic Flow 2.</p> <p>Steps:</p> <ol style="list-style-type: none"><li>1. IF The system is unable to contact the robot THEN inform the user ENDIF</li></ol> <p><b>Postcondition:</b> The system notifies the user of the error.</p>

## 1.3 Metrics

By Bardia Parmoun and Cassidy Pacada

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

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



**Figure 2:** A map of the Carleton University tunnels.

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

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



### 1.4.1 List of States, Events, and Actions

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

#### **States:**

- Operational: The overall state containing all the valid states that the robot can have.
  - Find Wall: Indicates the state where the robot has still not found a wall.
  - Docking: The robot has reached a docking station and will dock.
  - Turn Left: The robot has reached an intersection turning left.
  - Turn Right: The robot has reached an intersection turning right.
  - Wall Following: The robot is following a wall on its right side.
  - Collision Handling: The robot has encountered a collision.
- Not Operational: The robot has encountered a fatal error and has stopped.

#### **Events:**

- entry: executed upon entering the state.
- navLeft: indicating that the robot should turn left.
- navRight: indicating that the robot should turn right.
- navPass: indicating that the robot should pass the intersection.
- navDock: indicating that the robot should dock.
- gotWall: indicating that the robot has encountered a wall.
- lostWall: indicating that the robot has lost the wall.
- gotBump: indicating the robot has encountered a collision (from thbumper sensor).
- gotError: indicating the robot has encountered a fatal error.
- gotTrip: indicating the robot has received a new trip.
- exit: executed upon leaving the state.

#### **Actions:**

- nothing: The robot will not do anything.
- wallFollow: The robot will follow the target distance and angle sent by lidar\_sensor.
- turnLeft: The robot will make a left turn.
- moveForward: The robot will move forward.
- dock: The robot will go to the docking station.
- undock: The robot leave the docking station.
- setDir: The value of the nextDir parameter of the state is updated.
- kill: Stops the system.

#### **Parameters:**

- nextDir: Used to store the next direction that the robot should follow. This parameter will be used to save the next navigation sent using the navigation\_plot node.

### 1.4.2 Justifications

By Bardia Parmoun

#### **The typical flow of the system:**

The robot starts at the FindWall state waiting for a wall signal. Once it has received a gotWall event from the LiDAR, it moves to the WallFollowing state where it constantly receives gotWall events and wall follows. During wall following the robot may receive navigation events as to its next moves. These events could be navRight, navLeft, navDock, or navPass. During this stage, these events will be saved in the nextDir parameter of the state. Once the robot gets a lostWall event, it will use the nextDir variable to determine where to go. This cycle will continue until the robot receives navDock as its next destination and move onto the Docking state.

Note that there can be cases where the robot get the nav events after losing the wall (ie reaches the intersection first then detects the beacon). In this case, the robot can directly move to the appropriate state for each nav event.

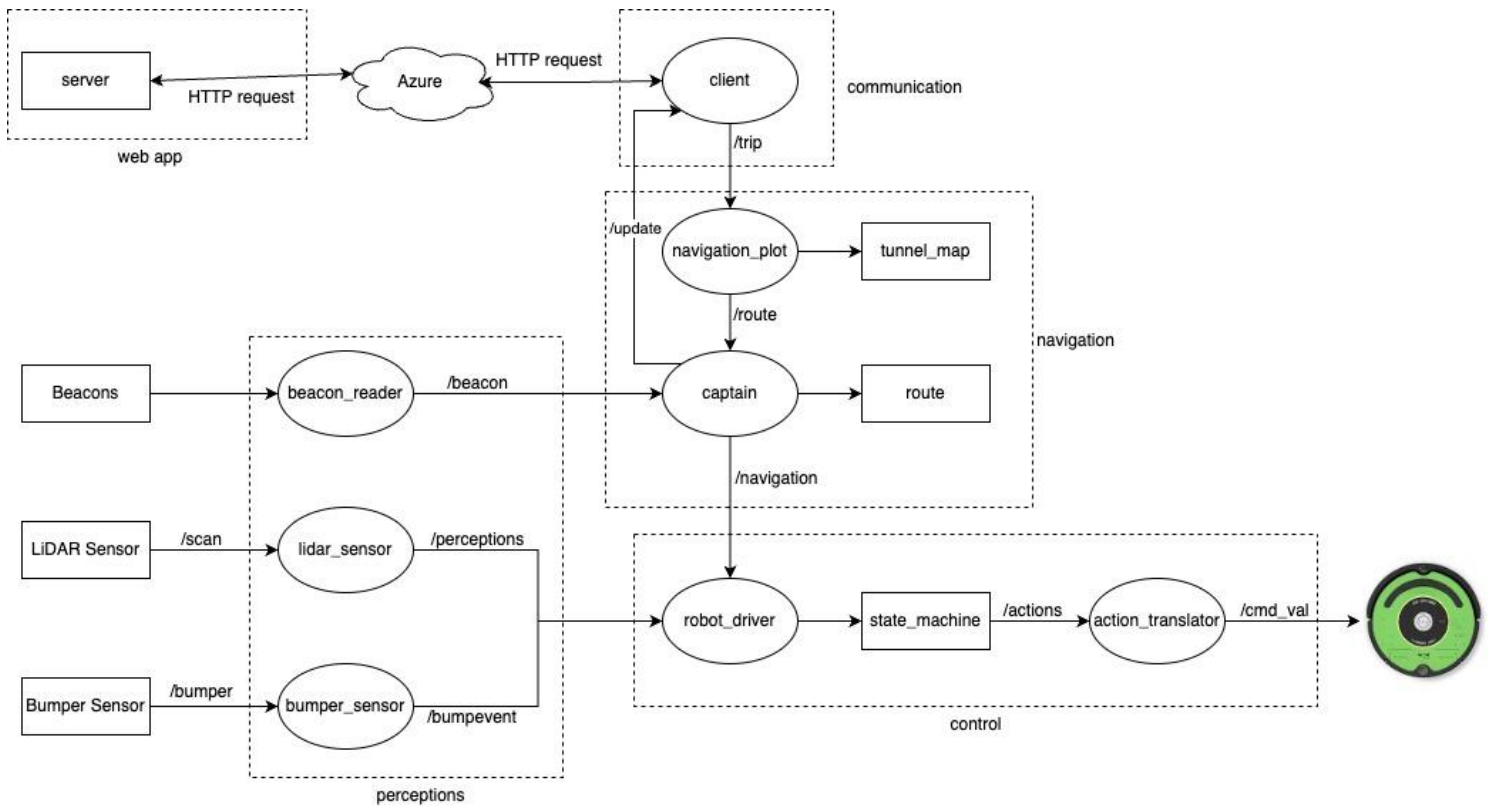
Note that there is a dedicated state for the 4 navigation events: For navRight, the robot will simply wall follow since that should allow it to easily turn right; for navLeft, the robot will turn left upon entry then when getting a wall it goes back to wall following; for navPass, the robot will constantly move forward until it reaches to a wall and does wall following; finally, for docking the robot will just run the automatic dock command.

#### **Collision Handling:**

The state machine is designed to account for the various collision handling events. If a collision occurs during the wall following state, the robot initiates the collision handling event with the nextDir variable. This allows the robot to remember its previous state. During collision detection all the robot needs to do is turn left. Once the robot has turned left it can then follow along the obstacle until it is over. This is achieved through the gotWall event that the robot will get from the obstacle. This allows it to go back to the wallFollowing state while remembering any nextDir directions that might have occurred beforehand. It is also possible for the robot to fully avoid the obstacle after a left turn and not have a wall (ie if the collision occurred at an intersection). This means that the robot will instead receive a lostWall event since there is no wall beside it anymore. Upon receiving lostWall, the robot will take advantage of the nextDir event that was passed onto it to determine which state it should go to. For example, in the case that the robot simply skipped the obstacle and is not at an intersection, the robot will just go to Find Wall and then the Wall Following states and resume its flow.

## 1.5 Node Structure

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



**Figure 4:** The proposed ROS node structure for the system.

### 1.5.1 List of Nodes, Messages and Entities

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

#### Nodes:

- Server: Defines a node which describes the server (from the web application).
- Client: Defines a node which describes the client.
- Navigation\_plot: Defines a node which describes the navigation plotting.
- Captain: Defines a node which describes the captain - controlling the robot en route.
- Beacon\_reader: Describes a node which allows for reading the detected beacons.
- Lidar\_sensor: Describes a node for the LiDAR sensor.
- Bumper\_sensor: Describes a node for the bumper sensor.
- Robot\_driver: Describes a node for the robot driver, which takes in state machine actions.
- Action\_translator: Describes a node for the action translator, which translates state machine actions into valid serial commands.

**Messages:**

- /trip: Sends a trip message from the client node to the navigation\_plot node.
- /route: Sends a route message from the navigation\_plot node to the captain node.
- /update: Sends an update message from the captain node to the client node.
- /navigation: Sends a navigation message from the captain node to robot\_driver node.
- /actions: Sends an action message from the robot\_driver node to the action\_translator node.
- /cmd\_val: Valid command that represents a state machine action sent to the robot.
- /bump\_event: Bumper sensor event message sent to the robot\_driver node.
- /perceptions: LiDAR sensor event message sent to the robot\_driver node.
- /beacon: Beacon reader message sent to the captain node.

**Entities:**

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

### 1.5.2 Justifications

By Max Curkovic

**The typical flow:**

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

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

## 2 Background

By Max Curkovic

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

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

### 2.1 Robot Operating System

By Matt Reid

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

### 2.2 iRobot Create

By Matt Reid

The current robot is built on the iRobot Create 2 which is an educational version of an autonomous iRobot vacuum that connects over a serial cable to a microcontroller to receive commands [2]. As discussed in Section 2.1, ROS is used to send commands from a Raspberry Pi to the serial port on the iRobot Create 2. The Raspberry Pi receives and processes all of the



sensor data, and then sends the robot commands to drive. It also supports the use of the “Virtual Wall” which works like a Bluetooth beacon and creates a wall that the robot will not pass.

A new version of the platform, called the iRobot Create 3 has ROS2 built-in and does not require a microcontroller to send instructions on what to do [3]. The robot has 6 IR obstacle sensors built in to detect objects and follow walls. It also has a custom faceplate which allows for easy mounting of boards through many screw holes, and cable passthrough to the storage compartment. Inside the storage component, it provides a 14.4V/2A battery connection as well as a 5V/3A usb-c port that can be used to power and connect to components. The built-in ROS2 includes a full interface of ROS commands to take sensor data, follow walls, and positional navigation. The use of ROS means that it would easily be able to run the same code that is currently being run on the Create 2 platform.

## 2.3 LiDAR Sensor

By Matt Reid

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

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

### 3 Description

By Bardia Parmoun and Matt Reid

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

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

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

## 4 Justification of Suitability for Degree Program

By Max Curkovic and Bardia Parmoun

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

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

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

Furthermore, the web application portion of the project is related to the material covered in SYSC 4504 (Fundamentals of Web Development) and SYSC 4806 (Software Engineering Lab). It is also worth mentioning that requirement analysis and software architecture will be covered in every step of the project which are the main focus of the following courses, SYSC 3020 (Introduction to Software Engineering), SYSC 3120 (Software Requirements Engineering), and SYSC 4120 (Software Architecture and Design). Finally, there will be some 3D design work done for the physical aspects of the robot such as its chassis which was covered in ECOR 1054 (Fundamentals of Engineering IV).

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

## 5 Group Skills

By Bardia Parmoun

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

- **Bardia Parmoun:** As a 4th-year software engineering student, Bardia has a lot of experience working with different languages such as Python, Java, C, and C++. Bardia also has some experience working with web development tools and languages such as HTML/CSS and JavaScript. Bardia also has some embedded software development experience and some electronics knowledge. Finally, Bardia also has some experience working with 3D designs and modeling.
- **Cassidy Pacada:** Cassidy is also a 4th year software engineering student with a lot of experience with Python and Java. Cassidy also has experience with front-end development using HTML/CSS and JavaScript. She also has some experience working in the field of cybersecurity which could be useful with regards to the security of the robot. Finally, Cassidy has a lot of experience working in Linux environments.
- **Max Curkovic:** Max is a software engineering student with knowledge of programming languages such as Python, Java, and C. Max also has a lot of experience with full-stack web development, which can help with the development of the web application associated with the robot. Max also has experience with Raspberry Pis and Linux.
- **Matt Reid:** Matt is a computer systems engineering student. As such he has a lot of experience working with embedded systems and hardware design. Matt has also done some projects with Raspberry Pis and Arduino microcontrollers. In addition, Matt has previously worked with ROS which is the operating system that is used to control the iRobot's movement. Finally, Matt also has some electrical engineering knowledge which could help with the circuit designs for the robot.

## 6 Methods

By Cassidy Pacada

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

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

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

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

## 7 Analysis

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

### 7.1 Analyzing the Equipment

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

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

**Table 12:** Measuring the speed of the robot

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

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

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

**Table 13:** The ability of the robot to find the wall and maintain it

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

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

**Table 14:** The ability of the robot to make a successful right turn consistently

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

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

**Table 15:** The ability of the robot to make a successful left turn consistently

<b>Trial #</b>	<b>Description (Missed, OK, GOOD, PERFECT)</b>
1	Missed
2	Missed
3	Missed
4	Missed

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

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

**Table 16:** Measuring the beacons at known distances

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

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



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

**Table 17:** Measuring the strengths of the internet signal in Carleton University tunnels

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

## 7.2 Beacon and Dock Station Placement

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

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

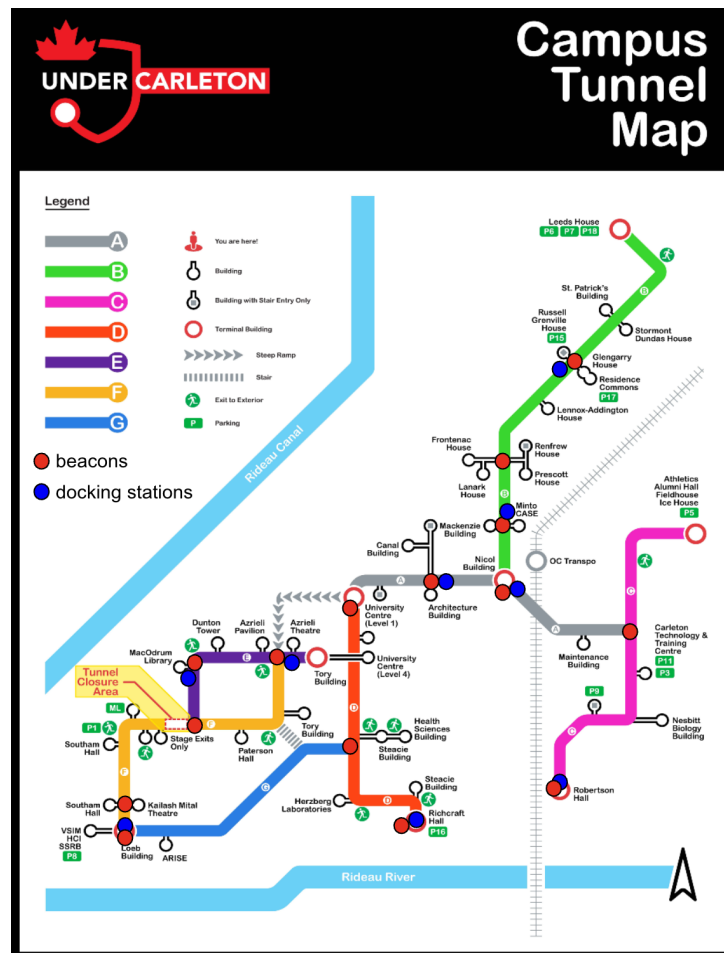


Figure 5: Proposed locations for the beacons and docking stations in the tunnels.

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

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

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

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

- Tunnel B:
  - Lanark/Renfrew House Entrance
- Tunnel C:
  - Tunnels A and C intersection
- Tunnel D:
  - Long ramp/Tunnel D turn
  - Tunnels D and G intersection
- Tunnel F:
  - Tunnels E and F intersection
  - Southam Hall/Kailash Mital Theatre

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

## 8 Project Timeline

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

The preliminary timeline for the project is below in Table 1, as well as in Gantt chart format in Figure 4.

### 8.1 Proposed Timeline

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

**Table 18:** Project milestones and proposed target completion dates in table form

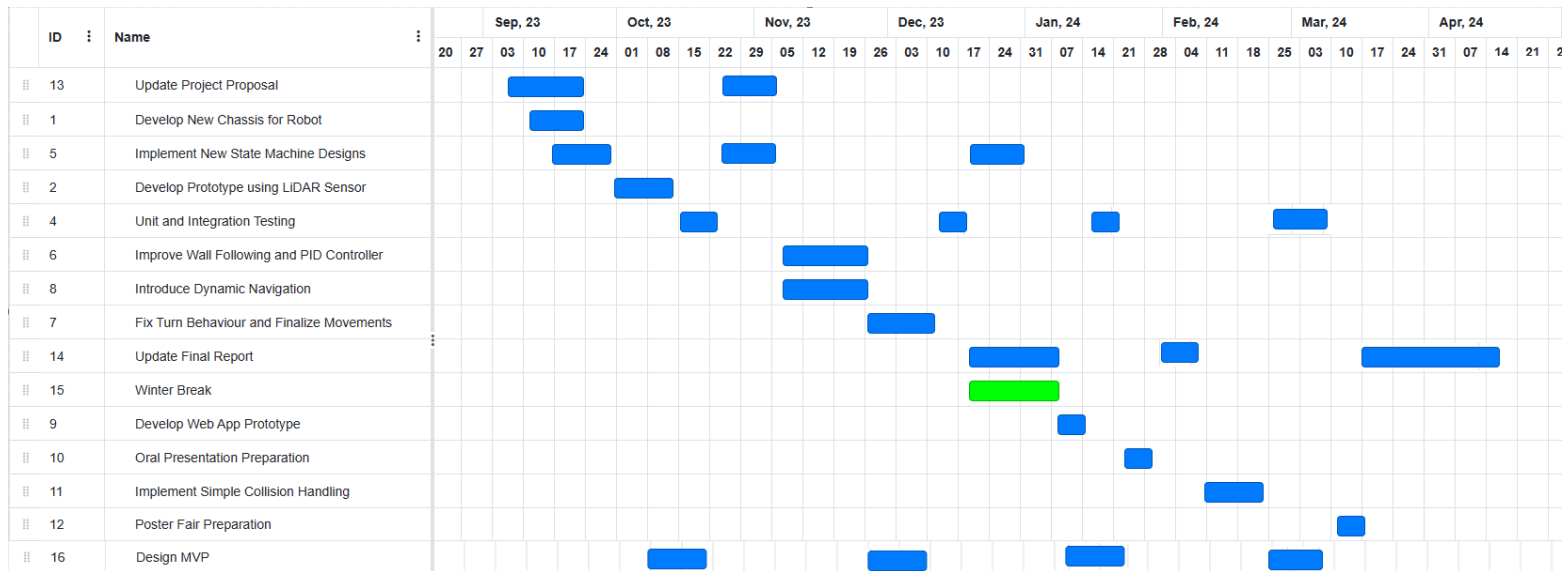
Milestone	Target Completion Date
1 - Design a new chassis that accommodates better stands for the components and print it.	September 30th, 2023
2 - Prepare a prototype using the lidar sensor and compare performance with existing sensors	October 13th, 2023
3 - Proposal	October 20th, 2023
4 - Introduce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

## 8.2 Gantt Chart

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

Each project milestone is divided as its own separate task, with Figure 4 (Gantt chart) being an agile, visual representation of when each milestone should be completed for.

Overall, each task is divided into its own “milestone”, subsumed by different cycles. The cycle typically starts with implementing any new states that were designed as part of the state machine, then considers all unit and integration testing for each task, and then allows for time to work on the project report. If this cycle is completed, then the team progresses to the next milestone.



**Figure 6:** Project milestones and proposed target completion dates in Gantt chart form

## 9 Risks and Mitigation Strategies

By Cassidy Pacada

**Table 19:** Possible risks and their mitigation strategies

Project Risk	Mitigation Strategy
Potential hardware failure can slow project progress as it takes time to re-solder circuitry. Should a larger, more important piece fail, such as the iRobot itself, it may take an extended period of time to replace.	The current faulty circuitry will be replaced with new equipment to minimize the risk of failure. Additionally, the team has extra pieces of critical hardware on hand in case a failure does occur.
Making incorrect changes to the code can potentially set the project back if there is no way to recover previously functional code.	The team will use a version control system, Github, to ensure that changes can be reverted if necessary.
There is a risk that the existing sensors will not be sufficient to navigate the tunnels effectively. Should this implementation strategy fail with no backup, it would leave the project directionless.	The team has come up with alternatives to fall back on if the initial navigational implementation does not work. These include using Lidar sensors and computer vision to navigate.
Any test of functionality where hitting an obstacle can cause damage to the robot or its hardware can potentially lead to costly replacement of said hardware.	The team will supervise and monitor all tests of functionality whenever they are performed, as someone can step in and stop the robot if absolutely necessary. All precautions to protect the robot (e.g. ensuring the chassis is in place) will also be taken prior to testing.

## 10 List of Required Components/Facilities

By Cassidy Pacada

**Table 20:** List of required components/facilities, their objectives of the project, and estimated cost

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

### 10.1 Justification of Purchases

By Cassidy Pacada

#### Sensors

**Table 21:** Comparison of sensor candidates based on power usage, range of effectiveness for both distance and direction, and price

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

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

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

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

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

## Battery Bank

**Table 22:** Comparison of battery pack candidates based on supplied power, size, and price

	<a href="#">Anker Portable Charger</a> [7]	<a href="#">Anker PowerCore 10000 Portable Charger</a> [8]
<b>Supplied Power</b>	15W	12W
<b>Size</b>	6.2in x 2.9in x 0.8in	3.6in x 2.3in x 0.9in
<b>Price</b>	\$56.50	\$38.42



The main candidates considered for the battery bank that would be used to power the Raspberry Pi and the LiDAR were the Anker Portable Charger and the Anker PowerCore 10000 Portable Charger. The biggest determining factor for our decision to choose the Anker PowerCore was that it was smaller than the Anker Portable Charger. The robot has a small surface area and a lot of the space is already being taken up by the Raspberry Pi, the LiDAR, and various wires. Although a chassis has been designed to hold everything in place, the team prefers to preserve as much space as possible so the robot will have space for the mail holder.

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

# 11 Documented Issues

By Matt Reid

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

## 11.1 Automatic Undocking

By Matt Reid

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

### **Temporary Solution:**

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

### **Permanent Solution:**

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

### **Other Solutions:**

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

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

## 11.2 Powering the entire system using the robot

By Matt Reid

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

### **Temporary Solution:**

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

### **Permanent Solution:**

Another option for powering the pi from the robot is to use the main vacuum brush's motor driver power. The main motor driver on the robot is capable of providing 1.45A at 12V which gives a capacity of 17W. Using a DC to DC converter, this means that the motor driver would be able to provide the Pi with 5V and 3A [11]. The problem with this solution is that there is no power to the motor driver until the robot is turned on, which requires a signal to be sent through the serial port (unless done manually). One solution for this would be to use a microcontroller that requires less power such as the Pi Zero W to send a start command over the serial so that the Pi 4 model B receives power and takes over from there with the LIDAR.

### **Other Solutions:**

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

As with the undocking issue, this issue would also be solved with the iRobot Create 3 platform. As was discussed in Section 2.2, the Create 3 provides a 5V/3A usb-c output to the

storage bay which would be enough to power the LiDAR module (the Pi would also no longer be necessary due to the built in ROS). If the LiDAR were connected to the usb-c port, it could communicate with the built in microcontroller and be sufficiently powered. As discussed previously, this solution will not be entertained this year due to the large added costs.

## 12 Minimum Viable Product

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

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

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

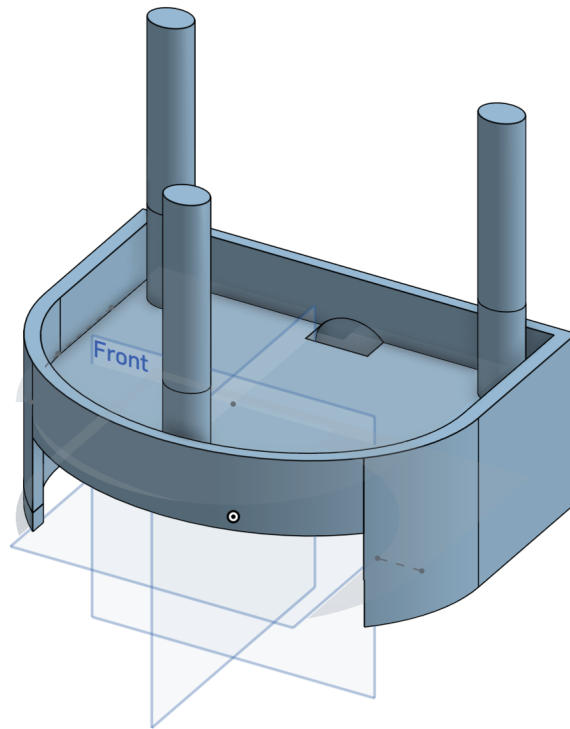
This MVP fully implements the state machine outlined in figure 3. It also includes the following ROS nodes: beacon\_sensor, lidar\_sensor, captain, robot\_driver, and action\_translator.

## 13 References

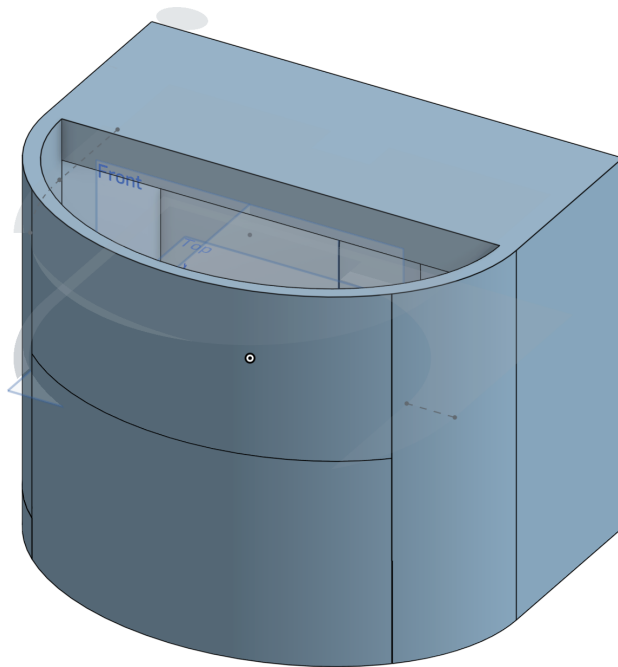
- [1] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, W. Woodall, “Robot Operating System 2: Design, architecture, and uses in the wild,” Science Robotics vol. 7, May 2022. (accessed Sep. 22, 2023).
- [2] “Create 2 Robot: What We Offer” edu.irobot.com.  
<https://edu.irobot.com/what-we-offer/create-robot> (accessed Sep. 22, 2023).
- [3] ”Overview - Create 3 Docs” edu.irobot.com.  
[https://iroboteducation.github.io/create3\\_docs/hw/overview/](https://iroboteducation.github.io/create3_docs/hw/overview/) (accessed Sep. 22, 2023).
- [4] “What is LiDAR?” Synopsys. <https://www.synopsys.com/glossary/what-is-lidar.html> (accessed Sep. 30, 2023).
- [5] “Slamtec RPLIDAR A1 Low Cost 360 Degree Laser Range Scanner. Introduction and Data Sheet” Slamtec.  
[https://bucket-download.slamtec.com/d1e428e7efbdcd65a8ea111061794fb8d4ccd3a0/LD108\\_SLAMTEC\\_rplidar\\_datasheet\\_A1M8\\_v3.0\\_en.pdf](https://bucket-download.slamtec.com/d1e428e7efbdcd65a8ea111061794fb8d4ccd3a0/LD108_SLAMTEC_rplidar_datasheet_A1M8_v3.0_en.pdf) (accessed Oct. 8, 2023).
- [6] “Meng Jie TF-Luna small size and Low Power, lidar range finder sensor module, single-point micro ranging module, ranging from 0.2-8m, Resolution 1cm, compatible with UART / I2C communication interface,” Amazon.ca: Electronics,  
[https://www.amazon.ca/Single-Point-Resolution-Compatible-Communication-Interface/dp/B09DCLPYV1/ref=sr\\_1\\_18?keywords=lidar&qid=1696789560&s=electronics&sr=1-18](https://www.amazon.ca/Single-Point-Resolution-Compatible-Communication-Interface/dp/B09DCLPYV1/ref=sr_1_18?keywords=lidar&qid=1696789560&s=electronics&sr=1-18) (accessed Oct. 8, 2023).
- [7] “Anker Portable Charger, Power Bank, 20K” Amazon.ca: Electronics,  
[https://www.amazon.ca/Anker-PowerCore-Technology-High-Capacity-Compatible/dp/B07S829LBX/ref=sr\\_1\\_6?crid=1Z7F36BX965BO&keywords=anker%2Bpower%2Bbank%2B5v%2B3a&qid=1697405645&sprefix=anker%2Bpower%2Bbank%2B5v%2B3a%2Caps%2C85&sr=8-6&th=1](https://www.amazon.ca/Anker-PowerCore-Technology-High-Capacity-Compatible/dp/B07S829LBX/ref=sr_1_6?crid=1Z7F36BX965BO&keywords=anker%2Bpower%2Bbank%2B5v%2B3a&qid=1697405645&sprefix=anker%2Bpower%2Bbank%2B5v%2B3a%2Caps%2C85&sr=8-6&th=1) (accessed Oct. 15, 2023).
- [8] “Anker powercore 10000 Portable Charger,” Amazon.ca: Electronics,  
[https://www.amazon.ca/Anker-PowerCore-Ultra-Compact-High-speed-Technology/dp/B0194WDVHI/ref=sr\\_1\\_29?crid=AKV3WJBMCKR8&keywords=anker%2Bpower%2Bbank%2B5V%2F2.4A&qid=1697245268&sprefix=anker%2Bpower%2Bbank%2B5v%2F2%2B4a%2Caps%2C73&sr=8-29&th=1](https://www.amazon.ca/Anker-PowerCore-Ultra-Compact-High-speed-Technology/dp/B0194WDVHI/ref=sr_1_29?crid=AKV3WJBMCKR8&keywords=anker%2Bpower%2Bbank%2B5V%2F2.4A&qid=1697245268&sprefix=anker%2Bpower%2Bbank%2B5v%2F2%2B4a%2Caps%2C73&sr=8-29&th=1) (accessed Oct. 15, 2023).

- [9] “Create 2 losing serial communication after toggling full to passive while charging” Robotics Stack Exchange.  
<https://robotics.stackexchange.com/questions/15433/create-2-losing-serial-communication-after-toggling-full-to-passive-while-charging> (accessed Oct. 27, 2023)
- [10] “Raspberry Pi 4 Tech Specs” Raspberry Pi.  
<https://www.raspberrypi.com/products/raspberry-pi-4-model-b/specifications/> (accessed Oct. 15, 2023)
- [11] “Battery Power from Create 2” iRobot Education.  
<https://edu.irobot.com/learning-library/battery-power-from-create-2>. (accessed Oct. 15, 2023)

## Appendix A: Chassis Design



**Figure 7:** The chassis design for the robot holding the circuits for the robot.



**Figure 8:** The design of the mailbox for the robot.



