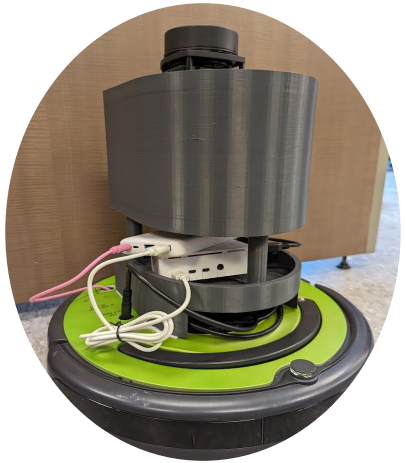
Carleton Mail Delivery Robot



Team 89

Supervisor: Dr. Babak Esfandiari

January 2024

Meet the team!

Bardia
4th Year, Software Engineering

Cassidy
4th Year, Software Engineering

Matt
4th Year, Computer Systems Engineering

Max
4th Year, Software Engineering

Agenda

- Project Description
- Background
- Planning
- Analysis
- Design
- Implementation
- Testing
- Conclusions
- Future work
- Q/A

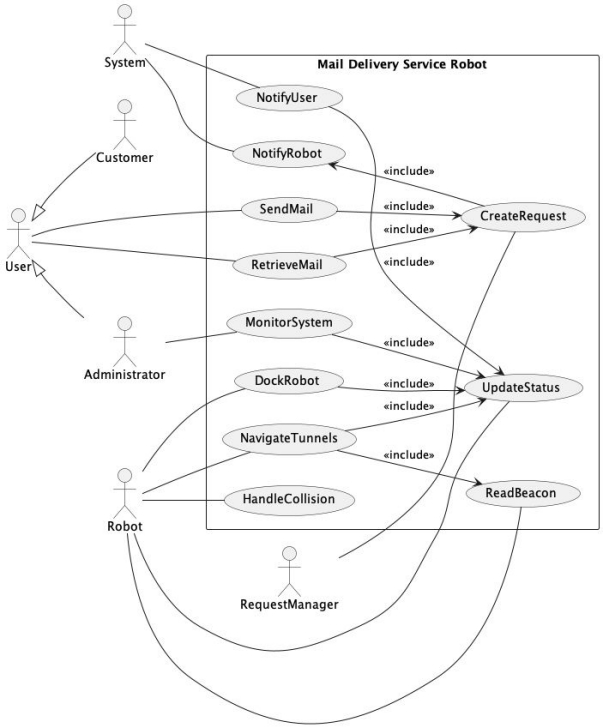
Project Description

Project Objectives

- Improve Carleton University's existing mail delivery system.
- Prepare a simple example of software Agent application.
- Apply different engineering principles such as design, documentation, analysis, and testing.

Project Description

- Developing a robot that autonomously delivers mail through the Carleton University tunnels.
- The system will have a web app component that will easily allow users to control the robot.

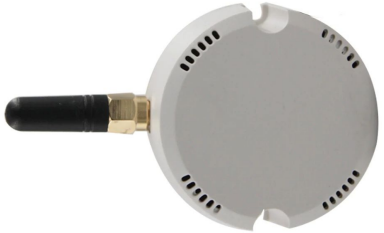


Background

Technologies Used

iRobot CREATE 2 & 3

Raspberry Pi 4B



AprilBeacon N04

RPLiDAR A1

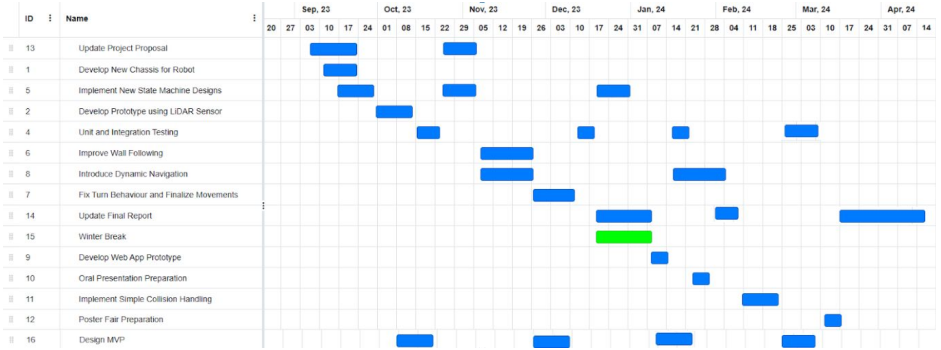
ROS

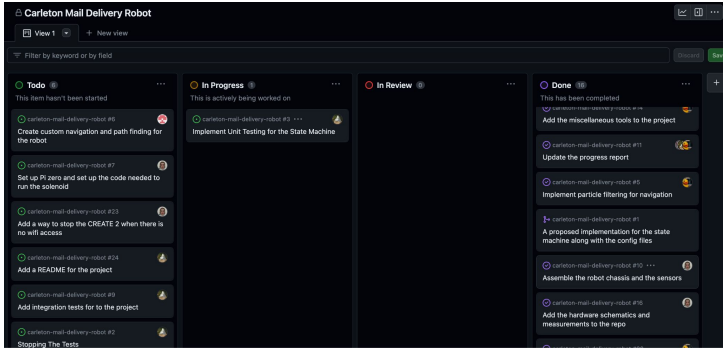
ROS2 (Foxy & Humble)

Planning

Agile Development

Milestone	Target Completion Date	Status
1 - Design a new chassis that accommodates better stands for the components and print it.	September 30th, 2023	Completed
2 - Prepare a prototype using the lidar sensor and compare performance with existing sensors	October 13th, 2023	Completed
3 - Proposal	October 20th, 2023	Completed
4 - Introduce unit tests and integration tests	October 27th, 2023	In progress
5 - Improve the state machine and implement new designs	November 4th, 2023	Completed
6 - Improve wall following	November 18th, 2023	Completed
7 - Fix the existing turn behavior and finalize movements	January 1st, 2023	In progress
8 - Replace the hard-coded navigation with a dynamic one	January 8th, 2023	In progress
9 - Create a simple prototype for the web app with simple commands: start, stop, status log, and delivery	January 22nd, 2023	Not started
10 - Oral Presentation	January 29th, 2023	Completed
11 - Add simple collision handling OR integrate iRobot's collision handling	March 10th, 2023	In progress
12 - Poster Fair	March 17th, 2023	Not started
13 - Final Report	April 12th, 2023	In progress





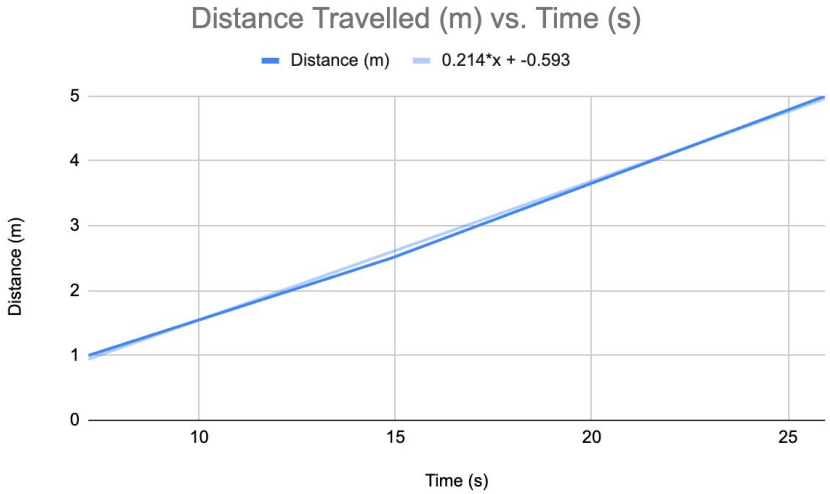
Carleton University

Project Risks

Risk	Mitigation Strategy
Hardware failure	Updated the circuitry
Insufficient sensor data	Moved to LiDAR from IR sensors
Unexpected events, including: • Robot is moved • Robot is blocked	Made the state machine more resilient

Analysis

Analyzing the Robot



Measuring the speed of the robot (Create 2)

Power Source	Rated Power (W)	Measured Power (W)
Serial Port	2 (10V, 0.2A)	3.2 (16V, 0.2A)
Main Brush Motor Driver	17 (~12V, 1.45A)	16.94 (12.1V, 1.4A)

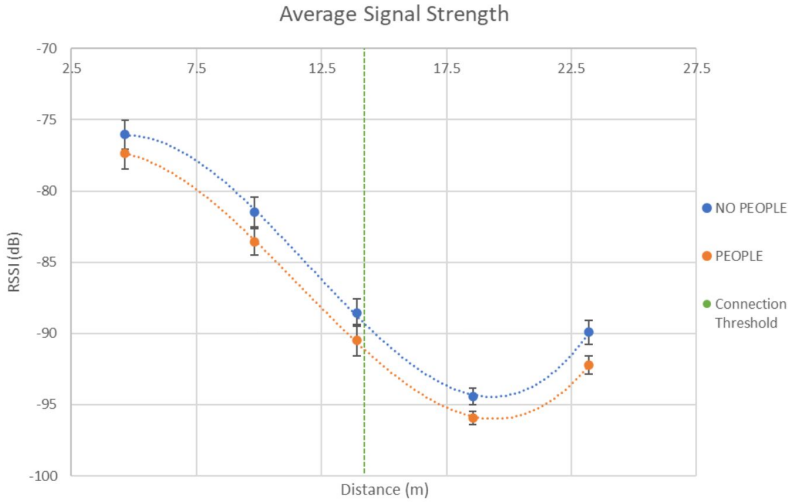
Measuring the Create 2 power outputs

Power Source	Rated Power (W)	Measured Power (W)
USB-C Port	15 (5V, 3A)	15 (5V, 3A)

Measuring the Create 3 power output

Analyzing the Beacons

- The team developed a simulator.
- Measured the beacon strength at different distances.

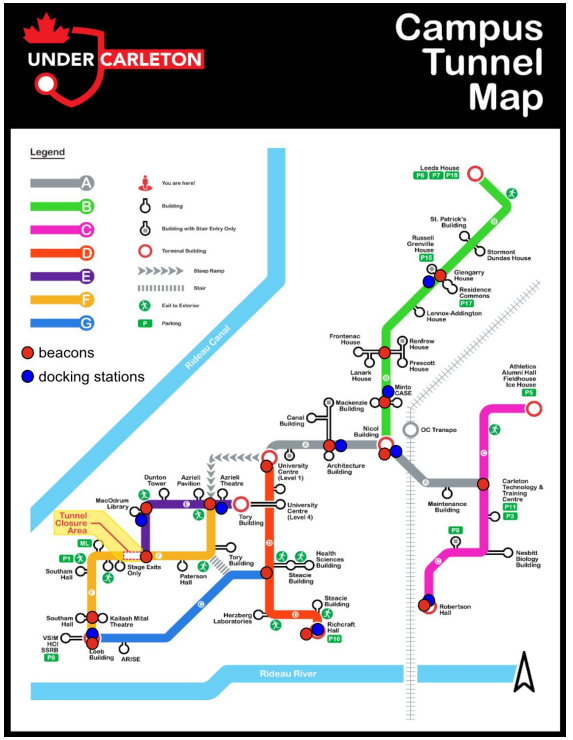


Beacon ID	Beacon Mac Address	Distance	RSSI
1	E2:77:FC:F9:04:93	1m	-73
1	E2:77:FC:F9:04:93	10m	-90
2	EA:2F:93:A6:98:20	1m	-69
2	EA:2F:93:A6:98:20	10m	-87
3	FC:E2:2E:62:9B:3D	1m	-71
3	FC:E2:2E:62:9B:3D	10m	-88.4
4	E4:87:91:3D:1E:D7	1m	-67
4	E4:87:91:3D:1E:D7	10m	-87
5	EE:16:86:9A:C2:A8	1m	-71
5	EE:16:86:9A:C2:A8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

Carleton University

Proposed Beacon Placement

- One beacon per intersection.
- One beacon per dock station.
- Grouped various destinations at each intersection to make implementation simple.

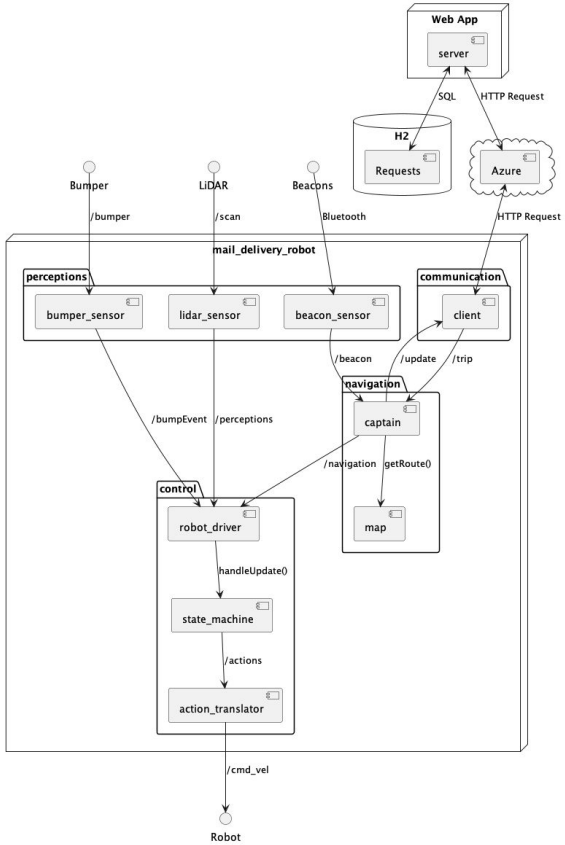


Carleton University

Design

Software Design - Deployment Diagram

- Two main subsystems:
  - Web App
  - mail_delivery_robot
- Web App will be deployed on Microsoft Azure.
- Major packages in mail_delivery_robot:
  - Perceptions
  - Navigation
  - Control
  - Communication

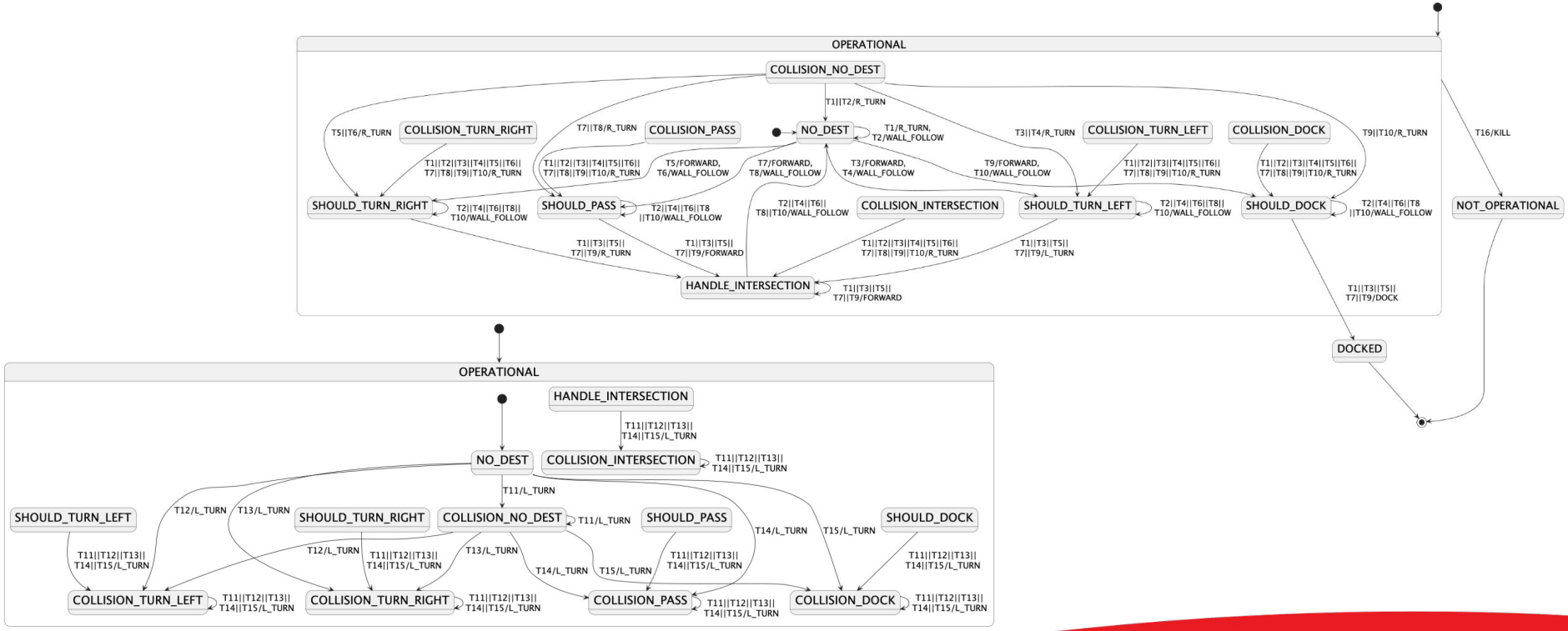


Software Design - State Machine (1)

- Four sources of events:
  - LiDAR
  - Beacon
  - Bumper Sensor
  - Error
- The state machine should account for all four.

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

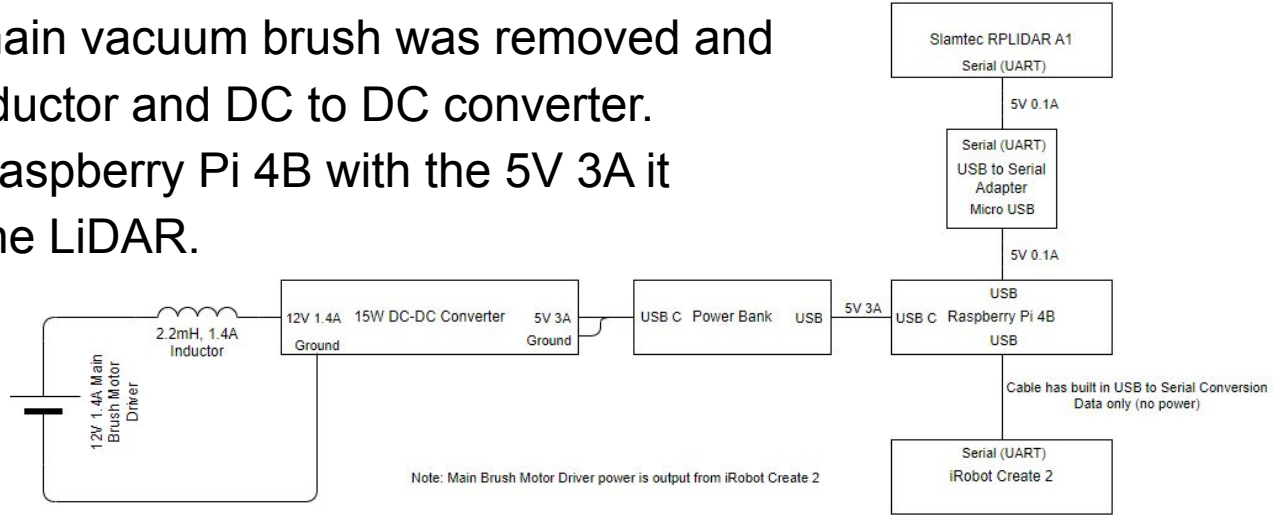
Software Design - State Machine (2)



Carleton University

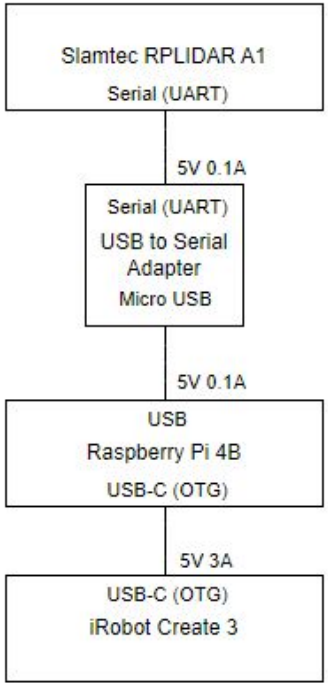
Hardware Design - iRobot Create 2

- The iRobot Create 2 Platform provides a UART Serial port for communication.
- To get power, the main vacuum brush was removed and replaced with an inductor and DC to DC converter.
- This provides the Raspberry Pi 4B with the 5V 3A it requires to power the LiDAR.



Hardware Design - iRobot Create 3

- The iRobot Create 3 Platform provides a 5V 3A USB-C OTG output removing the need for an external power system.
  - OTG allows for the robot to communicate with and power the Raspberry Pi 4B simultaneously.
- The LiDAR is connected to the Raspberry Pi the same way as the Create 2 design.



Implementation

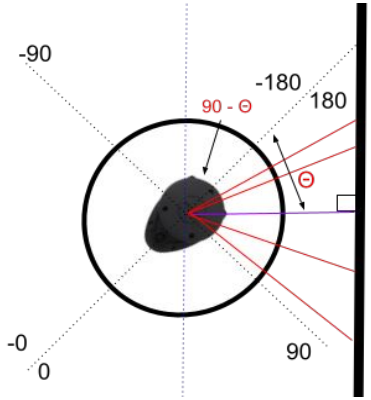
Software Implementation - State Machine

- The State Design pattern was used.
- The robot_driver sends updates to the current state, and a state is returned after the action is published.
- The robot_driver uses a clock (every 0.01 seconds) to pass perception data variables to the state machine and uses these values to trigger and call a transition.

```python
def updateStateMachine(self):
    '''
    The callback for the timer.
    Sends the current state of the robot to the state machine to update the state.
    '''
    # Waiting for everything to calibrate.
    if self.wall_data == "":
        return

    new_state = self.state.handleUpdate(self.bump_data, self.nav_data, self.wall_data)

    if new_state != self.state:
        self.state = new_state
        self.get_logger().info("Changed State " + new_state.printState())

    self.nav_data = "NAV_NONE" # Should reset the navigation data
```

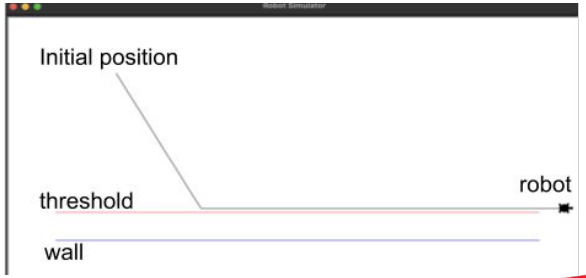
```python
def handleUpdate(self, bumper, nav, wall):
    '''
    Locates the appropriate state transition based on the update.

    @param bumper: The state of the bumper sensor.
    @param nav: The navigation event.
    @param wall: The wall status.
    '''
    self.wall = wall
    # Make sure to capture important navigation info.
    if nav != "NAV_NONE":
        self.nav = nav

    # Checks to see if the state machine has unfinished action.
    if self.longAction != None:
        if self.longActionCount < self.longActionLimit:
            self.actionPublisher.publish(self.longAction)
            self.longActionCount += 1
            return self
        else:
            self.longActionCount = 0
            self.longActionLimit = 0
            self.longAction = None
            self.isBusy = False
            return self.postLongActionState

    # Finds the proper transition based on the data.

    # Restore the saved nav data.
    if self.nav != "NAV_NONE":
        nav = self.nav
        self.nav = "NAV_NONE"

    if bumper:
        if nav == Nav_Event.NAV_LEFT.value:
            return self.bumper_left()
        elif nav == Nav_Event.NAV_RIGHT.value:
            return self.bumper_right()
        elif nav == Nav_Event.NAV_PASS.value:
            return self.bumper_pass()
        elif nav == Nav_Event.NAV_DOCK.value:
            return self.bumper_dock()
        else:
            return self.bumper_none()
    if nav == Nav_Event.NAV_LEFT.value:
        if wall == "-1-1":
            return self.no_bumper_left_no_wall()
        else:
            return self.no_bumper_left_wall()
    elif nav == Nav_Event.NAV_RIGHT.value:
        if wall == "-1-1":
            return self.no_bumper_right_no_wall()
        else:
            return self.no_bumper_right_wall()
    elif nav == Nav_Event.NAV_PASS.value:
        if wall == "-1-1":
            return self.no_bumper_pass_no_wall()
        else:
            return self.no_bumper_pass_wall()
    elif nav == Nav_Event.NAV_DOCK.value:
        if wall == "-1-1":
            return self.no_bumper_dock_no_wall()
        else:
            return self.no_bumper_dock_wall()
    else:
        if wall == "-1-1":
            return self.no_bumper_none_no_wall()
        else:
            return self.no_bumper_none_wall()
```

Software Implementation - Wall Following



- A very simple approach was implemented.
  - Find the angle with the wall.
  - If far, aim for a 60 degree angle with the wall.
  - When close to the wall, aim to be parallel.
- A small error region was defined for more accuracy.

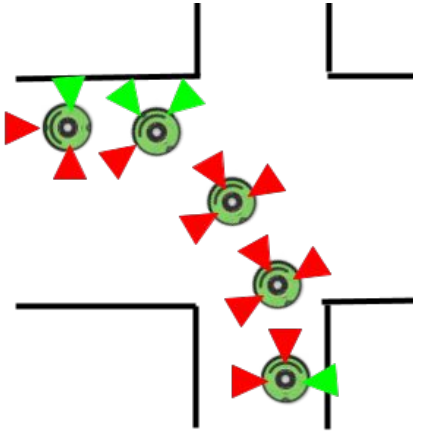
  error = (speed * sin(angle) * time) / 2
- A simulator was implemented to showcase this behaviour.



Software Implementation - Intersection Detection

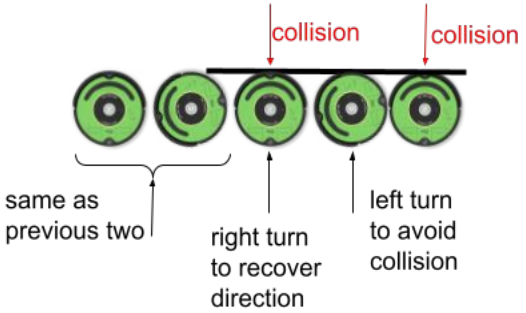
- The robot uses LiDAR to detect wall in its front, left, and right.
- Determines if it is at an intersection based on the expected intersection type and its direction (T vs four way intersections).
- A typical flow of states for an intersection:

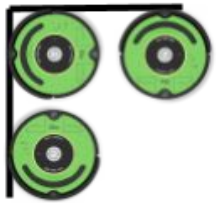
State	NO_DEST	SHOULD_TURN_LEFT	SHOULD_TURN_LEFT	HANDLE_INTERSECTION	HANDLE_INTERSECTION
Transition	T4	T2	T1	T1	T2
Action	WALL_FOLLOW	WALL_FOLLOW	L_TURN	FORWARD	WALL_FOLLOW
Next State	SHOULD_TURN_LEFT	SHOULD_TURN_LEFT	HANDLE_INTERSECTION	HANDLE_INTERSECTION	NO_DEST



Software Implementation - Collision Handling

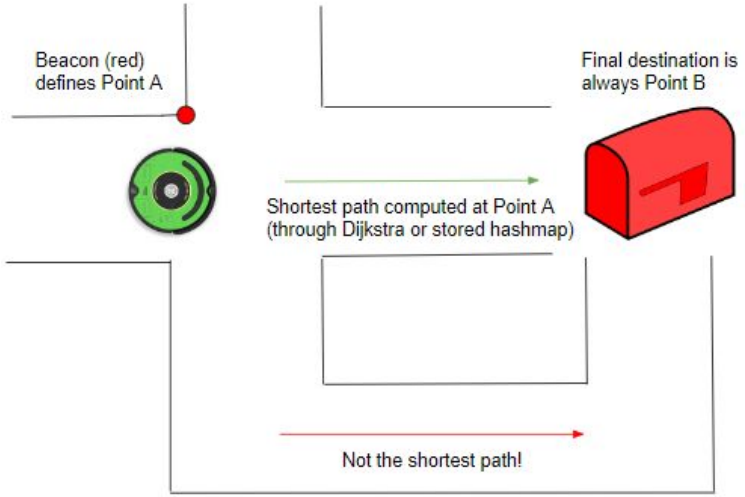
- A series of repeated left and right moves while slowly crawling forward.
- This way the robot maintains its original direction.
- It counts the number of left turns made so it can reorient itself properly.
- This is a very simple approach; it does not support complex obstacles.





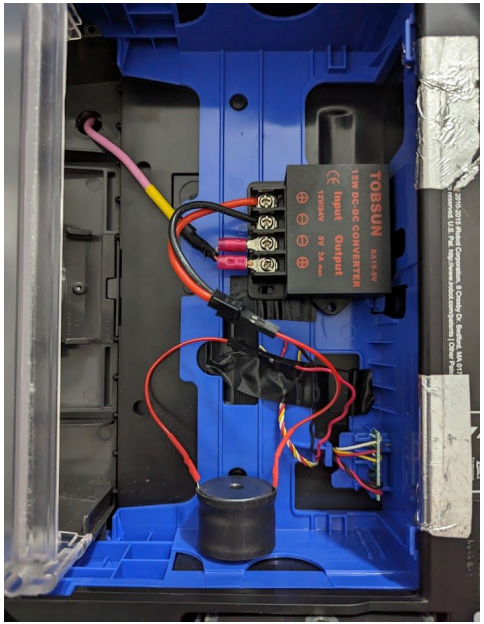
Software Implementation - Dynamic Navigation

- Shortest path for the robot is calculated using Dijkstra's algorithm (source-based) at every **intersection**.
- When a robot traverses a path, it will store the beacon id and final destination, for retrieval on a future route - increases the robot's resiliency.
- "Erroneous" events have to be accounted for, using **U-Turns**.



Hardware Implementation - Create 2

- The main brush motor driver is connected to the inductor and DC to DC converter underneath the robot.
- A 3D printed mailbox chassis holds the external components and mail.
- The LiDAR sits on top of the mailbox.



Hardware Implementation - Create 3

- The Create 3 hardware implementation is a work in progress.
- The Create 3 is connected to the Raspberry Pi 4B with a USB-C cable in the cargo bay.
- The LiDAR will be mounted on the top at the front of the robot.
- The chassis will only need to hold mail.

Testing

Unit Tests

- Test suite was based on the state machine and provides all transitions coverage.
- The team used pytest and colcon test.

```
cassidypacada@ubuntu: ~/cmds_ws
cassidypacada@ubuntu:~/cmds_ws$ colcon test --packages-select mail_delivery_robot --event-handlers console_cohesion+
Starting >>> mail_delivery_robot
--- output: mail_delivery_robot
============================= test session starts ==============================
platform linux -- Python 3.8.10, pytest-4.6.9, py-1.8.1, pluggy-0.13.0
cachedir: /home/cassidypacada/cmds_ws/build/mail_delivery_robot/.pytest_cache
rootdir: /home/cassidypacada/cmds_ws/src/carleton-mail-delivery-robot/mail_delivery_robot
plugins: ament-pep257-0.9.8, launch-testing-0.10.10, ament-xmllint-0.9.8, ament-lint-0.9.8, ament-flake8-0.9.8, ament-
collecting ...
collected 13 items

test/test_state_machine.py .............                                 [100%]

- generated xml file: /home/cassidypacada/cmds_ws/build/mail_delivery_robot/pytest.xml -
========================== 13 passed in 0.12 seconds ===========================
---
Finished <<< mail_delivery_robot [0.80s]

Terminal
1 package finished [1.87s]
cassidypacada@ubuntu:~/cmds_ws$
```

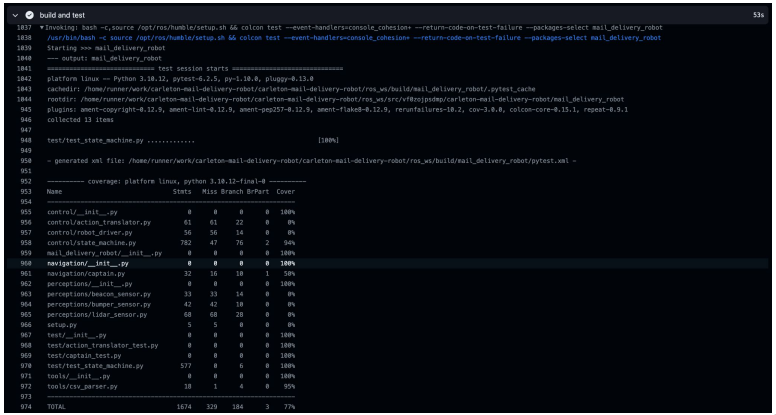
```python
def test_path_one():
    '''
    Test Path 1: Asserts that state transitions are good for right turns and that the completed actions
    are as expected
    '''

    actionPublisher = ActionPublisherStub()
    state = state_machine.No_Dest(actionPublisher)

    state = send_update(state, T2)
    assert state.stateType.value == "NO_DEST"
    state = send_update(state, T11)
    assert state.stateType.value == "COLLISION_NO_DEST"
    state = send_update(state, T13)
    assert state.stateType.value == "COLLISION_TURN_RIGHT"
    state = send_update(state, T6)
    assert state.stateType.value == "SHOULD_TURN_RIGHT"
    state = send_update(state, T1)
    assert state.stateType.value == "HANDLE_INTERSECTION"

    actionPublisher.extract_data()
    expected_actions = ["WALL_FOLLOW", "L_TURN", "L_TURN", "R_TURN", "R_TURN"]
    assert actionPublisher.action_data == expected_actions
```

Integration Tests

- Primary goal is to verify communication between the nodes.
- Simulate specific scenarios for the robot:
  - Intersection
  - Wall Following
  - Collision
  - And a combination of the these!
- Logs can be used to verify the robot is working as expected.
- Accelerating the development process by simplifying testing.

GitHub Workflow - Build and Test

- Continuous integration, a concept learned in SYSC 4806, in action!
- Pipeline is created within the project repository to ensure that all unit tests pass and the build compiles successfully.




```yaml
name: ROS Test

on:
  push:
    branches: [ "master" ]
  pull_request:
    branches: [ "master" ]

jobs:
  build_and_test:
    runs-on: ubuntu-latest

    steps:
    - name: Checkout repository
      uses: actions/checkout@v2

    - name: Set up ROS environment
      uses: ros-tooling/setup-ros@v0.7
      with:
        ros-distro: humble

    - name: build and test
      uses: ros-tooling/action-ros-ci@v0.2
      with:
        package-name: mail_delivery_robot
        target-ros2-distro: humble
```

Specified as ros_colcon_build.yaml

Demo!

Wall Following

Simple Wall Following

Complex Wall Following

Turns

Left Turn

Right Turn

Pass Intersection

Pass Intersection in the Tunnels

Collision Handling

Collision Handling in Canal Building

Collision Handling in the Tunnels

Resiliency!

Testing the robot in unexpected environments

Conclusions

Conclusions

- Relying solely on LiDAR for wall and intersection detection is unreliable.
- Investigating other approaches such as computer vision is recommended for future iterations.
- iRobot CREATE 3 is a much better option due to its many sensors and built-in ROS support.
- ROS Gazebo simulation is recommended to facilitate testing.
- The current implementation of the state machine can be improved to account for more complex actions (especially for collision handling).

Future Work

Roadmap Ahead!

- Fully implement dynamic navigation.
- Investigate a more reliable approach for detecting intersections (i.e using the signal strength of beacons).
- Implement a simple prototype of the web app.
- Transfer the code base to iRobot CREATE 3.
- Simulate different scenarios for the robot using integration tests.
- Make the robot's movements smoother by adding a simple PID controller for wall following.

Q/A

References

- 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.
- “Create 2 Robot: What We Offer” edu.irobot.com. https://edu.irobot.com/what-we-offer/create-robot.
- ”Overview - Create 3 Docs” edu.irobot.com. https://iroboteducation.github.io/create3_docs/hw/overview/.
- “What is LiDAR?” Synopsys. https://www.synopsys.com/glossary/what-is-lidar.html
- “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.
- “Raspberry Pi 4 Tech Specs” Raspberry Pi. https://www.raspberrypi.com/products/raspberry-pi-4-model-b/specifications/.
- “Battery Power from Create 2” iRobot Education. https://edu.irobot.com/learning-library/battery-power-from-create-2.
- “Create 2 losing serial communication after toggling full to passive while charging” Robotics Stack Exchange. https://robotics.stackexchange.com/questions/15433/create-2-losing-serial-communication-after-toggling-full-to-passive-while-chargi

Thank you!

Carleton University