April 10, 2024

Autocart

Ahmed Abdelaziz
101202944

Akuei Minyang
101133928

Bilal Chaudhry
101141634

Bren Gelyn Padlan
101148482

Cam Frohar
101148699

Kousha Motazedian
101140854

Mohamed Kaddour
101140829

Nadia Ahmed
101172713

Prianna Rahman
101140422

Sebastien Marleau
101155551

Abstract

The integration of smart systems into everyday life marks a new era of convenience and efficiency. At the forefront of innovation is the Autocart, revolutionary technology designed to redefine consumer experience in grocery stores, enhance operational workflows of warehouses, and streamline luggage handling in airports. This report introduces the Autocart project, detailing our journey in designing and constructing a universally adaptable autonomous cart. Our solution implements sophisticated navigation and mapping systems, object detection capabilities, and real-time localization to ensure reliable and autonomous cart performance across various environments. The following pages present a comprehensive narrative of our process, from the initial design stages to the creation of a functional prototype. We explore the challenges faced, the integrated technologies, and the potential implications of our work. Ultimately, this report illustrates our effort to bring this multipurpose autonomous cart from concept to reality, promising to impact multiple sectors by automating one of the most mundane yet essential tasks.

Acknowledgements

The Autocart team would like to thank supervisors Dr. Ramy Gohary and Dr. Ian Marsland for consistent feedback and guidance in the project's development. The team would also like to thank Dr. Ramy Gohary and Dr. Ian Marsland for their strong engagement with the project's ideas and challenges throughout its entire duration.

The Autocart team would like to thank Dr. Babak Esfandiari for providing valuable feedback during the progress presentation and for being consistently engaged with the project's development. The team would also like to thank Dr. Babak Esfandiari for acting as a second reader for the project's deliverables.

The Autocart team would like to thank Dr. Ioannis Lambadaris for comments on the project's scope, limitations, and future developments. The team would also like to thank Dr. Ioannis Lambadaris for assisting in a LiDAR connection issue and for acting as a second reader for the project's deliverables.

The Autocart team would like to thank Daren Russ, Patrick Fairs, and Professor Nagui Mikhail for additional guidance on the various electronic and mechanical aspects of the cart's developments.

Table of Contents

1. Introduction 7
1.1. Background 7
1.2. Problem Statement 7
2. The Engineering Project 8
2.1. Health and Safety 8
2.2. Engineering Professionalism 8
2.3. Project Management 9
2.3.1. GitHub 9
2.4. Jira 10
2.5. Justification of Sustainability for Degree Program 10
2.5.1. Justification of Sustainability for Software Engineering 10
2.5.2. Justification of Sustainability for Computer Systems Engineering 11
2.5.3. Justification of Sustainability for Communications Engineering 11
2.6. Individual Contributions 11
3. Requirements 13
3.1. Identification of Stakeholders 13
3.2. Requirements Elicitation 13
3.2.1. Functional Requirements 13
3.2.2. Non-Functional Requirements 13
4. System Design 15
5. Collision Detection 17
5.1. Introduction 17
5.2. Ultrasonic Sensors 17
5.2.1. Ultrasonic Sensor Use Case 17
5.2.2. Development of One Ultrasonic Sensor 18
5.2.3. Development of Multiple Ultrasonic Sensors 20
5.3. LiDAR Sensor 23
5.3.1. RPLIDAR A1M8 23
5.3.2. Algorithm Design 24
5.3.3. Implementation 28
5.4. Choosing a Sensor 29
5.5. Future Work 30
5.5.1. Adaptive Path Planning 30
5.5.2. Object Recognition and Classification 30
6. Motor Design and Control System 31
6.1. Introduction 31
6.1.1. Motivation 31
6.1.2. Design Scope and Scale 31
6.2. System Requirements 32
6.3. Control Mechanisms 32
6.3.1. Ackerman Steering 32
6.3.2. Differential Steering 32
6.4. Hardware Components 33

6.4.1. Motors 33
6.4.2. Forces Against the Wheel Calculation 35
6.4.2.1. Torque Calculation 36
6.4.3. Pico 37
6.4.4. Motor Driver 38
6.4.5. Encoders 41
6.5. Software Control Systems 42
6.5.1. PID Speed Controller 42
6.6. Turning 47
6.7. Prototype I 50
6.8. Prototype II 51
6.9. Schematic 52
6.10. Future Work 53
7. 2D-LiDAR Localization 55
7.1. Introduction 55
7.2. Initialization of map 56
7.2.1. Orthogonal Regression 56
7.2.2. Preloaded Map 59
7.3. Test Positions 60
7.4. Orientation 61
7.5. Localization Algorithm 62
7.6. Precise Angle 63
7.7. Future Works 64
7.7.1. Improving Localization in Crowded Environments 65
7.7.2. Enhancing LiDAR Performance Through Hybrid Sensor Solution 65
7.7.3. Live Map Forming 66
8. Sound Localization 67
8.1. Introduction 67
8.1.1. Motivation 67
8.1.2. Exploratory Phase 68
8.2. Design 70
8.2.1. Correlation 70
8.2.2. Localization Algorithms 75
8.2.2.1. Time of Arrival (ToA) 77
8.2.2.2. Time Difference of Arrival (TDoA) 79
8.2.3. Signals Used 81
8.2.3.1. Gaussian Noise 81
8.2.3.2. Linear Chirps 82
8.2.3.3. Sampling Rate and Accuracy 86
8.3. Equipment Used 86
8.3.1. Raspberry Pi (RP) Pico (RP2040) 86
8.3.2. RFM69HCW Wireless Transceiver 87
8.3.3. Arduino-Pico Core 87
8.3.4. Speakers, Microphone, and DAC Converter 88

8.4. Proof of Concept 88
8.4.1. Setup (Demo) 88
8.4.2. Deployment Diagram of the Setup 90
8.4.3. Circuit Diagrams of the Setup 90
8.5. Future Work 92
9. Integration 94
9.1. I²C Communication 94
9.1.1. Raspberry Pi Communication with Pico 94
9.2. Socket Communication 97
9.2.1. C++ to Python 97
9.2.2. Python to MATLAB 99
9.3. Making the Robot Move in a Predefined Path 102
10. Budget 105
11. Applications 107
12. Reflection 109
13. Conclusion 112
References 113
Appendix A: Code 116
Appendix B: Progress Report 166

1. Introduction

1.1. Background

The purpose of development in areas of autonomous vehicles is to increase efficiency and convenience for private and public settings, and also in day-to-day life of the average individual. This could include instances where the primary objective is the transport of an object between two points. The vision for this project is to develop an autonomous vehicle that could be applied in settings such as within a grocery store or a warehose, or as a food delivery robot.

1.2. Problem Statement

The development of an autonomous cart system as a general baseline for a wide variety of applications is confronted with a significant challenge: achieving precise indoor localization. In other words, it is crucial that an autonomous cart be able to accurately determine its own position within an environment and understand where its destination lies relative to itself. An additional layer of complexity is introduced when considering a setting in which there is constant motion and potential obstacles impeding the cart's path. These aspects will be the central focus of the project, along with the development of other essential systems required the truly autonomous and safe operation of a versatile cart.

2. The Engineering Project

2.1. Health and Safety

Ensuring the well-being of our team and the safe operation of equipment is an important consideration in the development of our project. This section presents the safety protocols and design considerations that have been carefully integrated into the project.

To ensure the safety of students and faculty at Carleton, it was important to adhere to all basic rules and safety precautions, especially while working with electronics and mechanical assembly. Before performing any electrical work, the team made sure to turn off power before touching any wires or components, use insulated tools and gloves, and avoid contact with water or metals, maintaining a safe environment.

A risk assessment was conducted to identify potential hazards associated with the Autocart's operation. Mitigation strategies and operational safety measures were developed for each identified risk. For instance, the cart is equipped with collision avoidance to safeguard the wellbeing of users and passersby. It also has a manual override system that shuts down the operation of the cart in any emergency to prevent accidents.

2.2. Engineering Professionalism

The engineering students on this team highly practiced professionalism throughout the development of the Autocart project. Each member upheld respect, integrity and commitment throughout the progression of the project. Work was completed in a timely and ethical manner. Risk assessment and consideration for health and safety were also discussed beforehand to ensure that the prototype design would be safe to demonstrate to an audience, and for future researchers to replicate the design.

The team showed high appreciation to their supervisors and experts outside our team that gave us assistance and peer-reviewed the initial stages of our design. Their guidance ensured that all shortcomings that could lead to poor quality design were addressed in our system.

Furthermore, despite the diverse backgrounds of the team members – encompassing software engineering, computer systems engineering, and communications engineering – an environment of mutual respect existed, encouraging the exploration of varied perspectives and ideas. Conflict resolution was handled with professionalism demonstrating a commitment to preserving the integrity of our collaboration.

2.3. Project Management

2.3.1. GitHub

Our team used GitHub to facilitate the collaboration among teams. GitHub is a version control platform that allows developers to create, manage and share their code. We opted to use this to help us progress through the many stages of development. The use of branches helped different teams work towards the same solution. A screenshot of our GitHub repository is shown below in Figure 1.

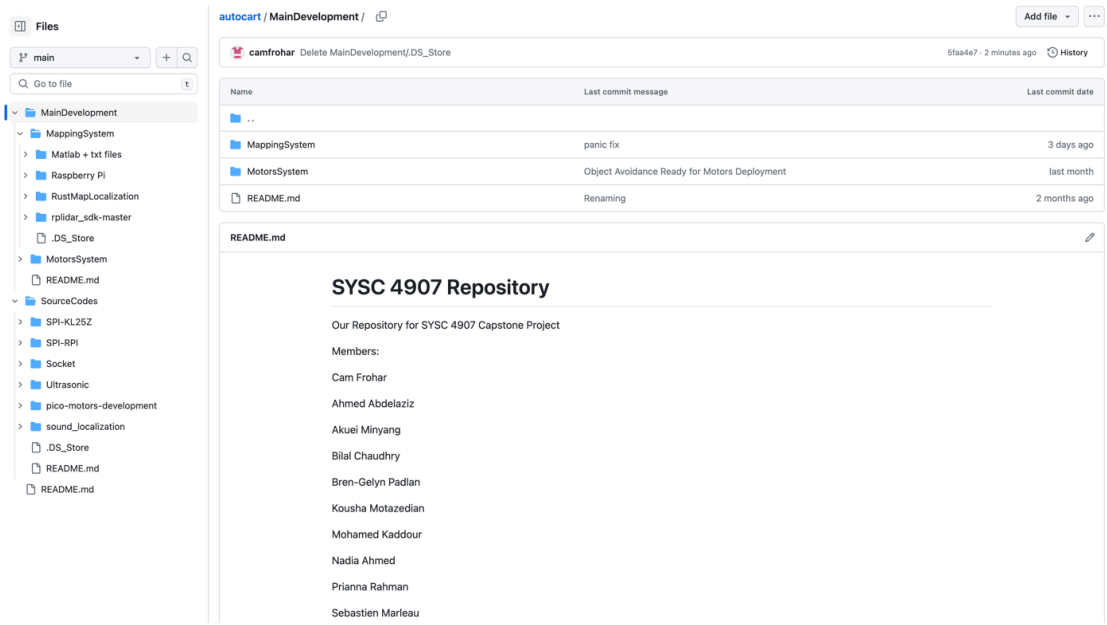

Figure 1: Screenshot of GitHub Repository.

As seen in the figure above, the left navigation pane shows all the different folders and files used in main development, as well as source codes to use as reference. The main development folder contained all of the code that was to be deployed to our final project.

2.4. Jira

Our team used Atlassian Jira to monitor and manage our progress throughout the year. Atlassian Jira is a tool that allows for task assignment, task tracking and overall project management. Our team opted to use Jira to improve our team's organization and time-management, this way, we could meet our goals and deadlines. A screenshot of our Jira page is shown below in Figure 2.

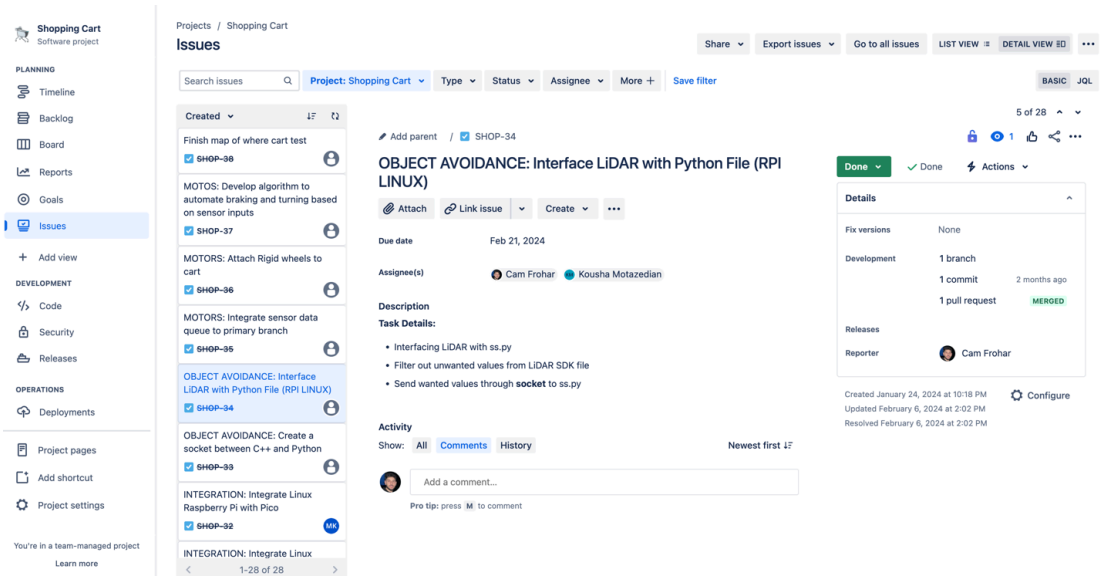

Figure 2: Screenshot of Team Jira Page.

An example of a task is shown in the figure above. Deadlines, assignees, and descriptions of tasks are all provided help us monitor our project progress. Additionally, our team's Jira was integrated with GitHub to assign branches to specific tasks, this way we could quickly access the actual progress of the task.

2.5. Justification of Sustainability for Degree Program

2.5.1. Justification of Sustainability for Software Engineering

Much of the project dealt with software development using the Python, C, and C++ programming languages. With the use of these languages, the valuable knowledge of embedded programming was built upon from the tasks of this project. Learning how to interface with different systems and programming languages was required for this project as the cart was heavily reliant on multiple systems being integrated together to have a functioning product. Finally, using the knowledge of algorithms, algorithms

for the cart's movement were carefully created to ensure proper movement and mobility for the cart.

2.5.2. Justification of Sustainability for Computer Systems Engineering

This project delved into the many concepts taught in Computer Systems Engineering. One example of a concept we were taught is being able to interface many microcontrollers using a variety of communication protocols. Another example is integrating an encoder with a PID controller, to be able to control the speed and turning of the robot. There are many other examples of concepts of Computer Systems that we put to use, such as sensor fusion, embedded systems design, power management, and actuator design/control. These concepts were acquired through the various courses and projects throughout our degree.

2.5.3. Justification of Sustainability for Communications Engineering

Various aspects of this capstone are related to material taught in the Communications Engineering stream. For instance, the software development languages used in this capstone include Python and MATLAB, which are taught across various courses in this stream. The indoor localization aspect of the capstone is also heavily related to Communications Engineering. Specifically, in the context of sound localization, many wireless communications concepts such as line of sight (LOS) signals and multipath had to be considered. In addition, concepts such as signal correlation and processing were implemented. Overall, the knowledge gained from a wide variety of courses in the Communications Engineering stream was utilized in the capstone, with notable contributions to indoor localization.

2.6. Individual Contributions

Table 1 below that outlines the individual project contributions of each area of the project.

Area	Contributers
Motors and Mechanics	Akuei, Bilal, Mohamed
LiDAR Localization	Ahmed, Akuei, Sebastien
Sound Localization	Ahmed, Bren, Prianna, Sebastien
Collision Avoidance	Cam, Kousha, Nadia
Integration	Ahmed, Cam, Kousha, Mohamed, Sebastien

Table 1: List of Project Contributions

Table 2 below that outlines the individual report contributions of each area of the project.

Content	Contributers
Introduction	Bilal
The Engineering Project	Bren, Cam, Kousha, Nadia, Prianna
Requirements	Mohamed
System Design	Nadia
Collision Detection	Cam, Kousha, Nadia
Motor Design and Control System	Akuei, Bilal, Mohamed
2D-LiDAR Localization	Ahmed, Nadia
Sound Localization	Bren, Prianna, Sebastien
Integration	Cam, Mohamed, Sebastien
Budget	Kousha
Application	Cam
Reflection	Sebastien, Prianna, Ahmed
Conclusion	Ahmed, Cam, Prianna, Mohammed

Table 2: List of Report Contributions

3. Requirements

3.1. Identification of Stakeholders

1. The Autocart Team designing, building, developing, and testing the cart.
2. The supervising professors and the second reader professors who are interested in the project's progress and success.
3. The department that's providing the funding and budgeting for the project.
4. The relevant users and potential buyers that may vary depending on the application of the cart.
5. The public who may be surrounded by the cart would be concerned with safety measures taken regarding collision avoidance.

3.2. Requirements Elicitation

3.2.1. Functional Requirements

1. The vehicle shall be able to move forward and backward.
2. The vehicle shall drive straight.
3. The vehicle's speed shall be easily modifiable.
4. The vehicle's speed shall be able to stop.
5. The vehicle shall be able to perform pivots.
6. The vehicle shall be able to make slight adjustments given angles.
7. The vehicle shall be able to detect objects within a surrounding range.
8. The vehicle shall avoid objects by stopping if the object is within a surrounding range.
9. The vehicle shall be autonomous.
10. The vehicle shall be able to reach the points of interest.
11. The vehicle shall utilize an accurate indoor map for navigation.
12. The vehicle's system shall provide a way of viewing the vehicle's location on the map.
13. The vehicle's system shall provide a way of stopping the vehicle remotely in the case of unexpected behavior.

3.2.2. Non-Functional Requirements

1. The vehicle shall be quick in object detection with reaction times of less than 1 second.
2. The vehicle shall be quick in engaging brake functionality after object is detected with reaction times of less than 2 seconds.
3. The vehicle shall operate continuously and without interruptions for as long as it's powered on.
4. The vehicle shall operate securely avoiding issues of deadlocks, race conditions, and fatal runtime errors.
5. The vehicle shall ensure easy modification and extensibility in relation to the addition of plugins and features.

6. The vehicle shall be able to swiftly recover from faulty and unexpected states relating to movement, object detection and localization.
7. The vehicle shall be efficient in power consumption rlated to movement and motor functionality.
8. The vehicle shall ensure safety through robust brake mechanisms when traversing densely populated locations to avoid or mitigate injury.
9. The vehicle's automation shall be available at all hours of the day and every day.
10. The vehicle's software must be designed in a way to ensure scalability.
11. The vehicle's software must be well documented to ease modification.
12. The vehicle shall allow easy configuration of movement (and motor) speed.
13. The vehicle's remote access capabilities to instantly halt all processes in case of emergency shall have reaction times of no more than 1 second.

4. System Design

Upon determining our project's functional requirements, we began designing a system architecture that would support the specifications of our cart. Central to our design choices was the decision to leverage both hardware and software to create a cohesive system capable of autonomous navigation and indoor localization.

This section will explain the system architecture of our project as the deployment diagram in Figure 3 illustrates. Given the importance of design choices to the project's success, the following strategic decisions were made.

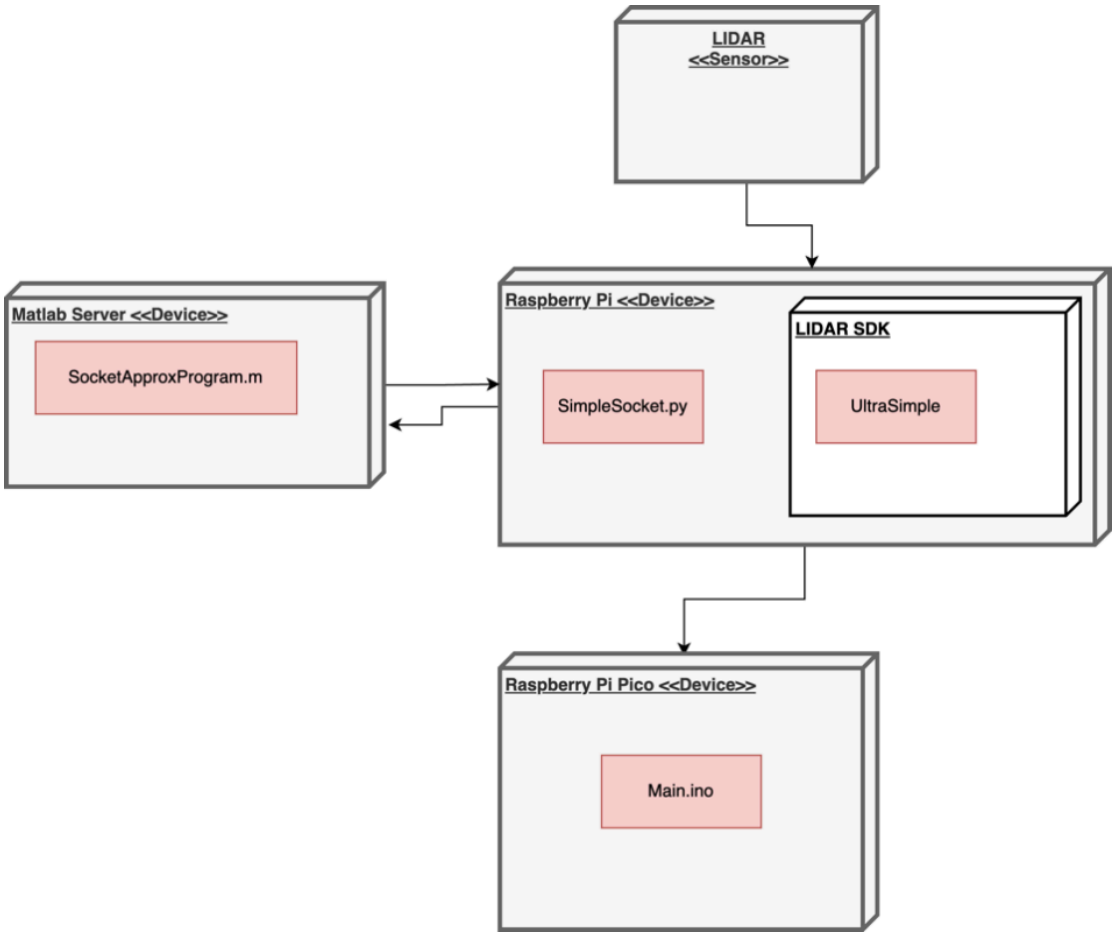

Figure 3: Deployment Diagram of the System Structure

The Raspberry Pi was chosen as the control unit, or the 'Main Brain,' of our system for its flexibility when it comes to integrating multiple subsystems, executing multiple complex programs, and managing communications between several different subsystems. As shown in Figure 3, the Pi runs a modified version of the 'UltraSimple' program from the LiDAR manufacturer's RPLiDAR Software Development Kit (SDK) for

interfacing with the sensor. It also runs the script ‘SimpleSocket.py’ that would allow the host communication with both the MATLAB Server and the Raspberry Pi Pico. This allows the Pi to receive information from the LiDAR, send that to the MATLAB server for processing, wait for a result, and then forward that result in case required to the motors via the Raspberry Pi Pico.

The control of the motors is carried out by a dedicated microcontroller, Raspberry Pi Pico. More details about this choice are referred to in Section 6.4.3. The ‘Main.ino’ program developed for Pico shall receive navigation commands over the I2C from the control unit and take actions of the motors according to the current system status.

On the other side, we have the LiDAR sensor integrated into the system. It is the most important component in providing real-time data necessary for both navigation and obstacle detection. It interfaces directly with the Raspberry Pi, where the data is processed and interpreted for decision-making.

5. Collision Detection

5.1. Introduction

The sensors team explored the importance of a collision detection system on our autonomous vehicle. We had to decide which sensor to use and how to use it in order to detect and avoid collisions efficiently and safely. We compared analog distance sensors, ultrasonic sensors, and time of flight sensors, and chose to pursue ultrasonic sensors. We had to make a lot of design choices to optimize the performance of our collision detection system, this included how many sensors to use and which microcontroller to control them. Upon deploying the ultrasonic sensors, we noticed some issues, which resulted in switching our focus onto a different sensor, the RPLiDAR A1 360-Degree Sensor. In this section, we will describe how we implemented both the Ultrasonic Sensors and the A1M8 LiDAR, as well as compare them and explain our design choice. Additionally, this section will provide insights on future work that can be done.

5.2. Ultrasonic Sensors

5.2.1. Ultrasonic Sensor Use Case

The first sensors the sensors team researched and experimented with for the collision avoidance system were Ultrasonic Sensors, specifically the HC-SR04 model. Ultrasonic sensors are proximity sensors that transmit sound waves and receive the echoes from those sound waves to determine the distance of any objects in front of the sensors [1]. These sensors are used quite frequently in applications for cars. These applications include parking assist, used for detecting any objects when parking and alerts the driver when it senses an object within a certain range, kick-to-open liftgates, where a person can put their foot underneath the back bumper of a car to open its truck, and many other object detection systems [2]. There are two important parts for an ultrasonic sensor to work. The first part is the transmitter and the second being the receiver, usually side by side with each other. The transmitter of the ultrasonic sensor is responsible for transmitting a sound wave, usually 40 kHz, which will reflect off an

object and will be captured by the sensor's receiver [3]. The below figure is a representation of these steps of how the sensor works.

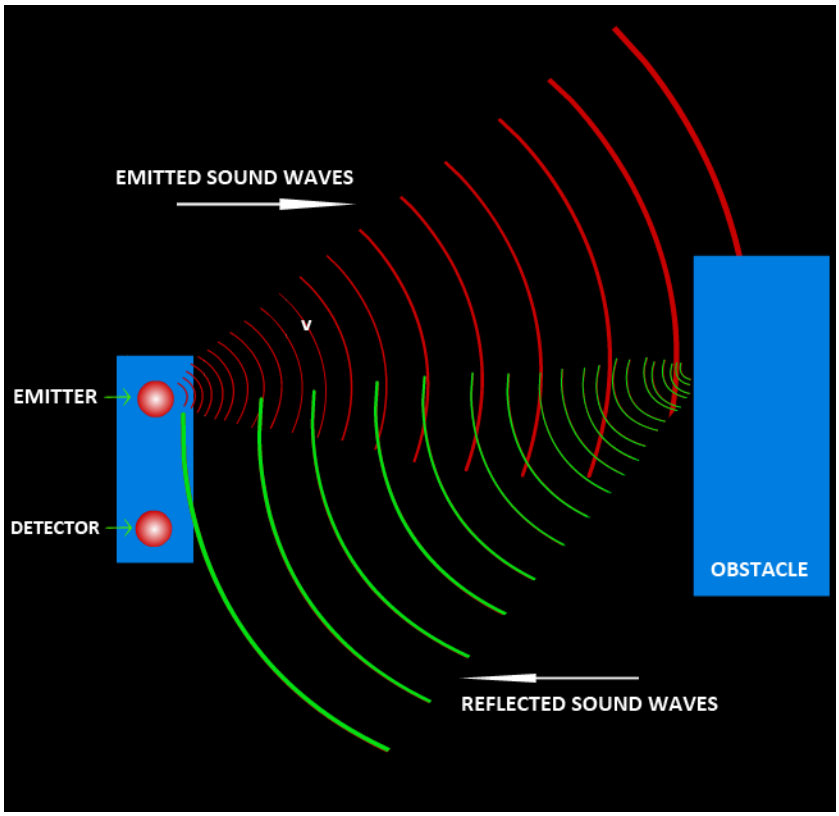

Figure 4: Ultrasonic Sensor Transmitting and Receiving Sound Waves Reflected From an Object [1]

To get the distance of the object from the receiver of the sensor, there is a simple formula that can be used: $D = \frac{1}{2}T * 343$ m/s, T being the time it takes in seconds for the sound wave to reflect back from the object, 343 m/s being the speed of sound, and D being the distance of the object in meters [1].

5.2.2. Development of One Ultrasonic Sensor

When the development of the system with the ultrasonic sensors commenced, at first, the KL25Z microcontroller was used. The team decided to start with developing the system with only one ultrasonic sensor and afterwards scale up. The HC-SR04 has 4 pins: Vcc, which powers the sensor, GND, to connect the sensor to ground, Trig, when given voltage will transmit a sound wave, and Echo, which is used to measure how long it takes for the reflection to come back. With that in mind, we started to write our code:

```
#Main method:
    loop:
        sendTrigger()
        wait(10ms)

#Interrupt:
    startTime = currentTime()

    #Wait for return pulse
    while(ECHO pin is High):
        wait()

    endTime = currentTime() - startTime
    distance = (endtime) * 0.0343 / 2
    print(distance)
```

Figure 5: One Ultrasonic Sensor Psuedocode

The above pseudocode was the proposed plan for the interaction with the sensor. The code will enable the trigger pin every 10 milliseconds in a loop so that it is continuously looking for objects in front of it. Once a sound wave is sent, there is an interrupt on the rising edge of the echo pin. The echo pin is set to high when a sound wave has been sent and the sensor has not received its reflection yet. This means that there needs to be a calculation for how long the pin is set to high so that the value for the later calculation. To do this, variables starttime and endtime were created to capture the time when the pin is set to high until the pin is low. After that, the distance formula mentioned before was used to get the distance of the object.

This is when the team switched over to the Raspberry Pi Pico. Even though the development was to be done using a different microcontroller, the structure of the code was the same. The code was converted from the KL25Z microcontroller to make it compatible with the Raspberry Pi Pico. Once the original code was adjusted for the new microcontroller and the hardware was properly set up, testing of the code commenced where adjustments and improvements were made.

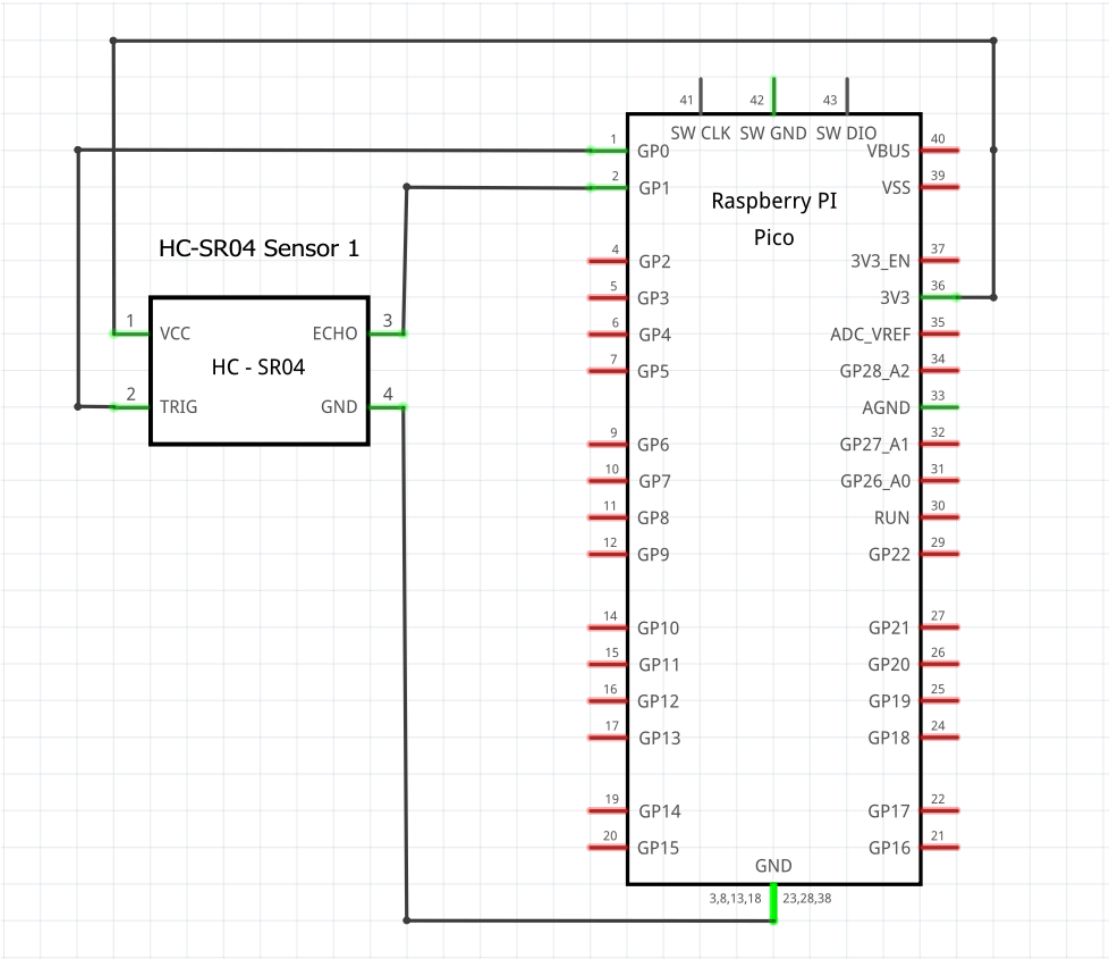

Figure 6: Schematic of the Pico Connections to the Ultrasonic Sensor

5.2.3. Development of Multiple Ultrasonic Sensors

From the tests, it was found that the sensor was able to detect objects up to 2 meters. One of the shortcomings of using one sensor was that the sensor has a very limited field of view to the sides and may not be able to properly detect objects. It was decided that the best way to counteract this was by adding more sensors to compensate for the blind spots. Adding more sensors to the board required changes to both the code and the hardware setup. This was fairly straightforward however as most of the changes required included adding more interrupt handlers and accessing more pins. LEDs were also added to the system for there to be a visual representation of when the sensors detected something.

```
#Main Method

Loop:
    #Transmit soundwaves for the three sensors
    sendTrigger1()
    sendTrigger2()
    sendTrigger3()

    delay(30)

    #Compute the information gathered (done for each sensor)

    pulsewidth = endTime - startTime
    distance = (pulsewidth * 0.0343) / 2
    if((0 < distance) and (distance < 60)):
        LEDon()


#Interupt Routine (one for each sensor)

    if(echoPin == HIGH):
        startTime = mircos()
        pinHigh = True
    ElseIf(pinHigh)
        endTime = mircos()
        pinHigh = False
```

Figure 7: Three Ultrasonic Sensors Psuedocode

Next, all of the sensors had to be connected to the microcontroller. This was simple as all the sensors were able to share the same VcPin and Ground Pin, there was only a need for separate pins for both the Echo and Trigger Pins:

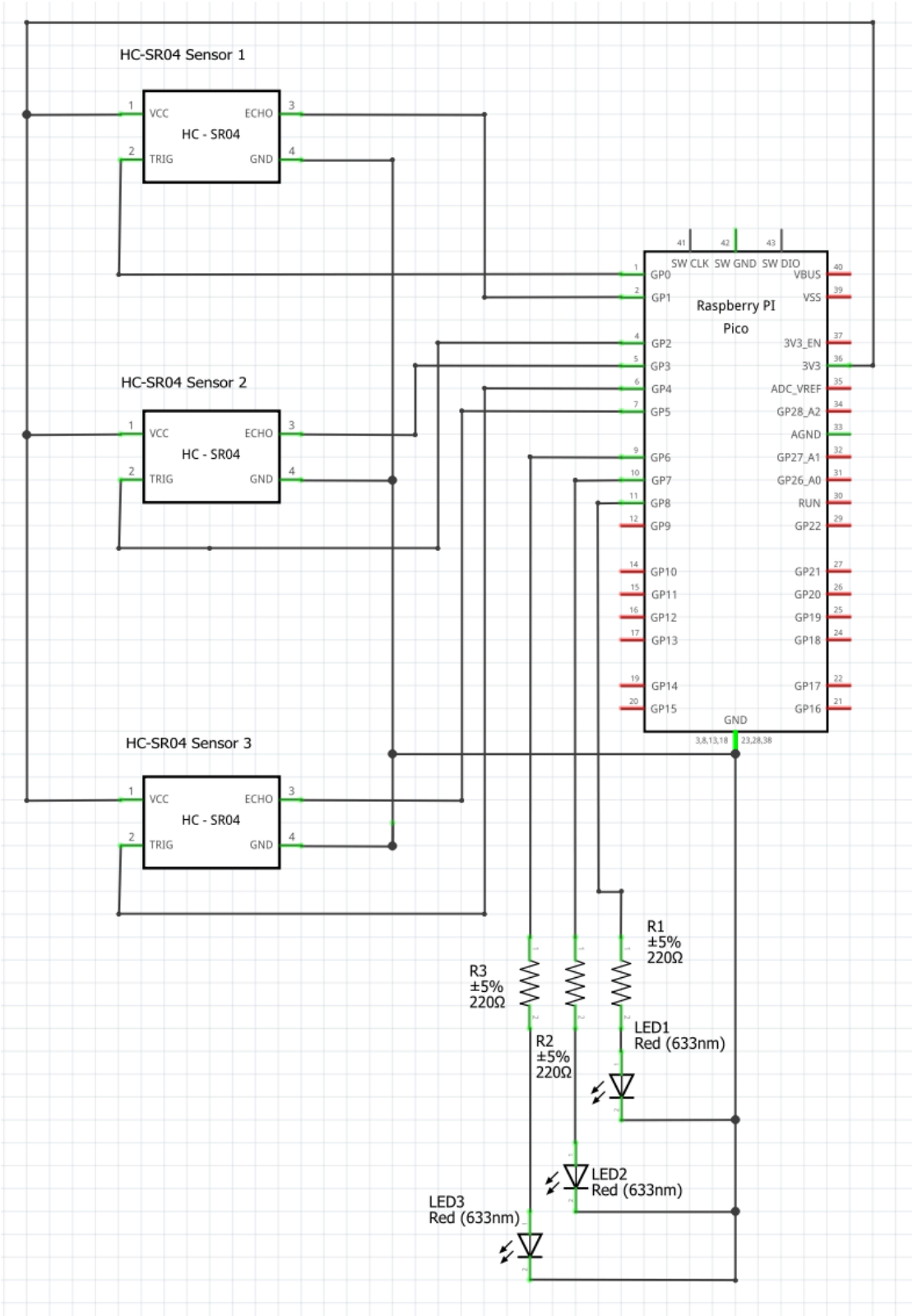

Figure 8: Circuit Diagram of the Three Sensors with Their LEDs

Extensive tests were done on the system with three ultrasonic sensors, and the team felt that the product in the end was good enough to be implemented in the cart as the collision avoidance system.

5.3. LiDAR Sensor

5.3.1. RPLIDAR A1M8

The second sensor we researched was the RPLIDAR A1. LiDAR sensors emit eye-safe laser beams into the environment and measure the time it takes for the laser to bounce back, to determine the distance to the surrounding objects [4]. Doing this thousands of times every second will allow the LiDAR to map its surroundings. The RPLIDAR A1 specifically, is a two-dimensional LiDAR that uses a servo motor that rotates clockwise, allowing for a 360° full-scan detection of its environment [4]. The LiDAR has many applications, including collision detection and indoor mapping. An image of the LiDAR is shown below in Figure 9.

Figure 9: Slamtec RPLIDAR A1M8 2D 360 Degree Sensor [4]

This sensor in particular had the strongest accuracy for a LiDAR that fit our budget, so we chose to pursue it. Our idea was the deploy this LiDAR on the top of our robot, with an unobstructed view of the environment, detecting people, objects, and walls. That being said, we needed a LiDAR that was able to detect obstacles quick enough, and accurate enough. Below in Table 3 are some of LiDAR's specifications.

Specification	Value
Power Supply	5 V
Detection Range	12 meters
Sample Rate	8000 per second
Sample Duration	0.125 milliseconds
Scan Rate	5.5 Hz
Scanning Frequency	0.30 degrees
Communication	UART
Angular Range	0-360 degrees

Table 3: RPLIDAR A1M8 Specifications [4]

We planned on plugging the LiDAR directly into the Raspberry Pi, so there was no issues on power supply requirements. The detection range of 12 Metres was more than enough, considering we only required our robot to detect obstacles within 1-2 metres. The sample rate of 8000 per second meant that the LiDAR would capture about 1450 different angles (0 to 360 degrees) every 0.125 milliseconds [4]. 1450 different data points would be able to capture all obstacles, so this also met our requirements.

5.3.2. Algorithm Design

Upon all of our requirements being met, we purchased the LiDAR and now had to shift our focus over to how to implement a collision avoidance system with this sensor. We had to design an efficient way to read sensor data in real-time and use the data to determine if the robot needed to make an emergency stop. We had to ensure that the calculations we made weren't too complex and didn't cause bottlenecks. We began trying simple algorithms, like one where we create a circular detection range around the robot and stop when an obstacle within 50cm is detected. A simple drawing of this design is shown in below in Figure 10.

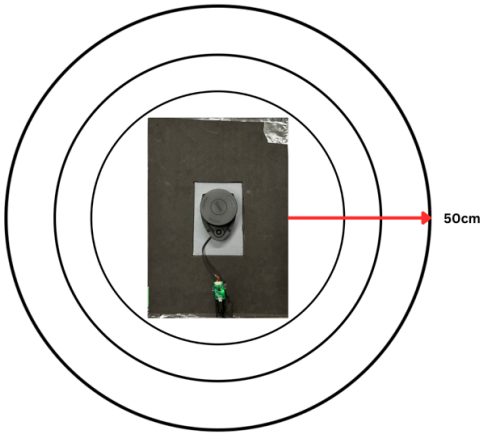

Figure 10: Simple Collision Detection Design

Although this algorithm worked to avoid collisions, it would also cause the robot to stop at times it did not need to stop. The robot would stop when an obstacle was detected behind it, as well as to the side of it. We quickly realized that we needed to create a stronger algorithm to only send a stop signal when the obstacle was in front of the robot. A drawing of this improved design is shown below in Figure 11.

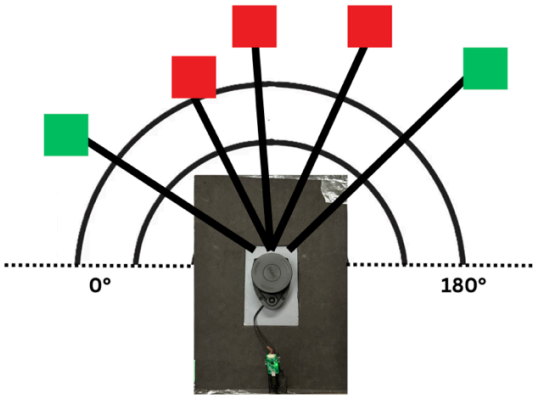

Figure 11: Improved Collision Detection Design

To implement this design, we needed to create an algorithm that that was strong enough to avoid only the obstacles in the robot's path. We created an algorithm that used the angles provided by the LiDAR to determine if an obstacle is found to be within the robot's path. Let's say an obstacle is detected by the lidar, and the distance to that obstacle is determined, we can use this as the hypotenuse. The LiDAR measurements would also give us the angle to the hypotenuse, relative to an origin. Now that we have both the angle to the obstacle as well as the hypotenuse, we can use a formula to calculate the adjacent distance to that obstacle:

$$\cos\theta = \frac{\text{adjacent}}{\text{hypotenuse}} \tag{1}$$

Once the adjacent distance is found, we can determine if the distance is large enough that our robot can continue driving without colliding. To do this, we would need to use the dimensions of the robot to find the desired stopping range. Figure 12 below is a schematic of our robot's dimensions.

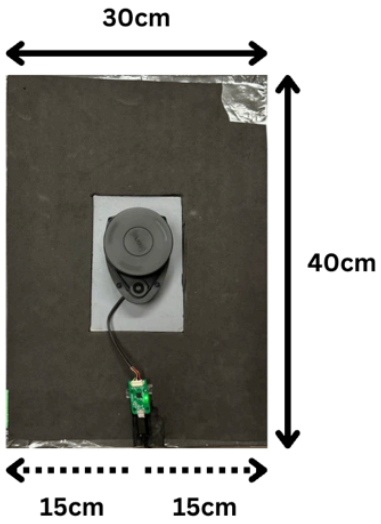

Figure 12: Robot Dimensions

Considering the distance from the LiDAR to the side of the robot is approximately 15 cm, we can create a stopping range of any obstacles with an adjacent distance of less than 16cm. This design will allow the robot to continue driving when an obstacle is not in the robot's path. A drawing of this example is shown below in Figure 13.

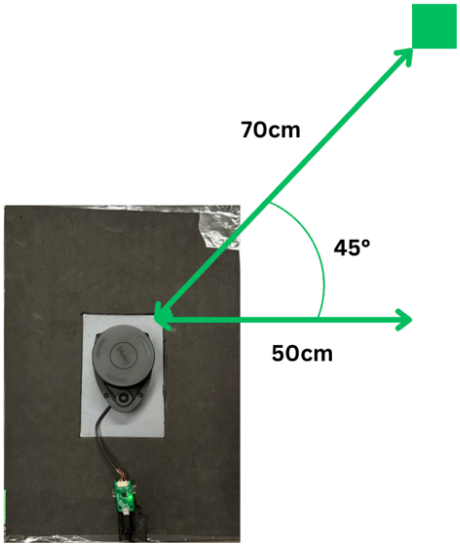

Figure 13: Example of Obstacle not in Robot's Path

In this example, 70 cm and 45° are given to us by the LiDAR, allowing us to compute for the adjacent distance, which is 50cm. In this case, since 50cm is greater than the 16cm stopping range, the robot is allowed to continue driving forward. In contrast, this design will also allow the robot to stop its motors when an obstacle is determined to be within the robot's path and less than one metre away. A drawing of this example is shown below in Figure 14.

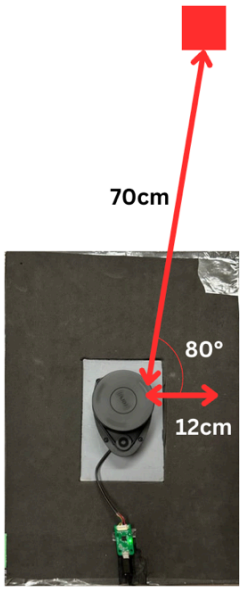

Figure 14: Example of Obstacle in Robot's Path

In this example, 70cm and 80° are given to us by the LiDAR, allowing us to compute for the adjacent distance, which is 12cm. In this case, since 12cm is less than the 16cm stopping range, the motors will be signalled to stop until the obstacle is no longer detected, in which it can then continue moving forward.

5.3.3. Implementation

This design worked successfully and was implemented in our final design. As previously mentioned in Section 4, our system design consists of a Python file (ss.py) that acts as the intersection between all other files, in other words, all communication goes through this file. In this file is where we decided to implement our collision detection algorithm. A simple pseudocode of our algorithm is shown below:

```
#Finish One LiDAR Rotation
if(LiDAR.RotationComplete()):

    #Obsatcle Found
    if (obsactle is < 1m and obsatcle is between 0° and 180°):

        #In Robot's Path
        if (adjacent < 16cm):
            collision = True
            motors.STOP()

        #NOT in Robot's Path
        else
            collision = False
            motors.FORWARD()

    #Obsatcle Not Found
    else
        collision = False
        motors.FORWARD()
```

The design waits for a LiDAR rotation to be fully completed, which takes approximately 0.125 seconds. It will then determine if any obstacle is found to be within 1 metre of the robot. If so, it will compute its adjacent distance and determine if it requires the motors to stop or not.

5.4. Choosing a Sensor

To compare the two sensors that were discussed in this section, both ultrasonic sensors and a LiDAR will accurately determine the distance to an obstacle. However, there is some overhead with ultrasonic sensors. Aside from the fact that an ultrasonic sensor only has a 15 degree and 400cm detection range, there are additional overheads with using them as the main sensor for collision detection on an autonomous robot. Firstly, you would need to cover the entire robot with ultrasonic sensors, and even then, the LiDAR will be able to better detect obstacles. This is because the LiDAR has a 360° range and emits approximately 1450 laser beams every 0.125 seconds. Second of all, requiring so many ultrasonic sensors would also require many GPIO ports that must support interrupts, meaning that we would need to use and integrate many microcontrollers with each other. The third reason is that with each of these microcontrollers come extra processing overhead and this would slow down our real-time processing.

Another issue with ultrasonic sensors is that it is not very accurate in detecting objects that are flat in front of it. If the object is positioned at an angle in front of the ultrasonic sensors, it would not be able to properly detect it due to the reflection of the transmitted sound wave not being reflected at the sensor, but instead being transmitted elsewhere:

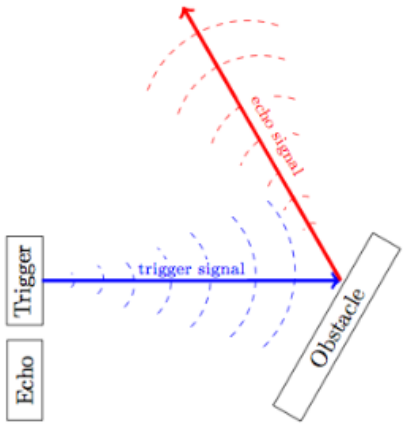

Figure 16: Example of the Sound Wave Reflecting Away from the Sensor

Since the LiDAR samples every 0.3 degrees at a very fast rate, it unlikely that it will miss the angled object, unlike the ultrasonic sensors. With all these ultrasonic sensor overheads, the RPLIDAR was the clear choice.

5.5. Future Work

The current implementation of the RPLIDAR A1M8 sensor on our autonomous cart has laid a strong foundation for robust collision avoidance. However, the field of robotics and sensor technology is rapidly evolving, and there are several avenues we plan to explore to enhance our system's capabilities further.

5.5.1. Adaptive Path Planning

While our cart can avoid collisions, the next stage is to develop adaptive path planning algorithms. Our current collision avoidance strategy effectively stops the robot when an obstacle is detected, however, this is a reactive approach. The transition to adaptive path planning will shift the system from reactive to proactive, enhancing the robot's ability to navigate complex environments seamlessly. The goal is to enable the cart to dynamically calculate alternative paths around obstacles without stopping. Integrating AI and machine learning algorithms can allow the robot to learn from its environment and make smarter navigation decisions over time.

5.5.2. Object Recognition and Classification

The integration of computer vision and AI techniques for object recognition and classification capabilities into our LiDAR-based collision detection system is an ambitious goal that would significantly elevate the cart's interaction with its environment. This can lead to more intelligent decision-making processes, where the robot could distinguish between static and dynamic obstacles, or even identify specific items that require special handling.

To achieve this, we plan to implement a multi-faceted approach that combines the precision of LiDAR measurements with the contextual understanding provided by computer vision.

6. Motor Design and Control System

This section outlines the purpose and components of the Motor Control System as part of the prototype that was constructed.

6.1. Introduction

6.1.1. Motivation

To design an autonomous cart, irrespective of its purpose, a fundamental requirement is the presence of a chassis capable of performing wheeled maneuvers and carrying a load. This is no exception for autonomous carts, navigating an environment, while potentially acting as a guide or to carry a load purchase. This leads to the motivation behind the Motors Control System, as to perform such tasks autonomously, there must be a system capable of directing the cart based on the surrounding environment, and the intended destination for the cart.

The Motor Control System includes the physical components that make up the chassis of the cart as well as the software control system that is used to generate the control signals to drive the motors. The goal of this system is to develop a method of control that allows the cart to operate consistently enough to allow the execution of commands recieved from the localization software.

6.1.2. Design Scope and Scale

Given the project's objective to develop a proof of concept for an autonomous vehicle, the Motor Control System was designed to control and operate the prototype cart. This design features few mechanical features to reduce the complexity of the physical system and add emphasize the software of the control system and the integration with the other devices used for autonomous operation.

Designing a system for a smaller prototype led to a reduction in the restrictions and requirements compared to a full-scale implementation. However, the mechanics that are discussed in this section are still applicable to larger systems. As such, despite the reduced scale of the prototype, the principles and methodologies applied are transferrable to a full-scale implementation. The scalability of the prototype proves it to

be an adequate proof of concept, and demonstrates how the mechanical design and control system can be applied to larger systems.

6.2. System Requirements

The cart shall be capable of fulfilling the following requirements:
- Accelerate from 0 to 1 m/s in less than 2 seconds
- Perform emergency brakes, coming to a full stop in less than one second
- Carry a maximum load of 10 kilograms

6.3. Control Mechanisms

6.3.1. Ackerman Steering

Ackerman steering is a method of steering used in most vehicles that allows the front wheels to pivot at different angles. This allows the navigation of turns by having the wheels on the inside and outside of the vehicle trace out circles of different radii around the center of the turn [5]. The angle at which the wheels are directed will dictate the rate at which a turn is completed, as it follows the curved path around the turn center as shown in Figure 17. Ackerman Steering reduces tire slippage when following a curve, but also introduces greater mechanical complexity as both front wheels must be attached to an axel that must be capable of making small adjustments accurately. This method was initially investigated for application in the prototype but was discarded due to the mechanical aspects necessary for it's implementation.

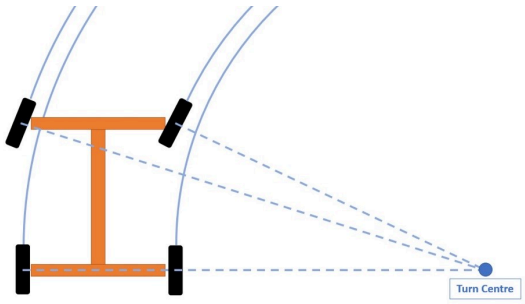

Figure 17: Depiction of a vehicle navigating a turn around the Turn Center using Ackerman Steering [5]

6.3.2. Differential Steering

Differential steering is another control mechanism that is commonly used in tracked vehicles and in skid steering systems. Differential steering functions by using fixed

wheels and applying greater torque to one side of the vehicle than the other [6]. This allows for motion and steering by controlling each motor independently. By driving each motor forwards at the same speed and output torque, the vehicle will drive straight forwards. If one side were to be slowed down, the vehicle would begin to follow a circle around the turning center. The rate of curvature is dependent on the ratio of the motor's speeds on both sides of the cart. Figure 18 provides a depiction on how differential steering is applied to a vehicle with 4 wheels.

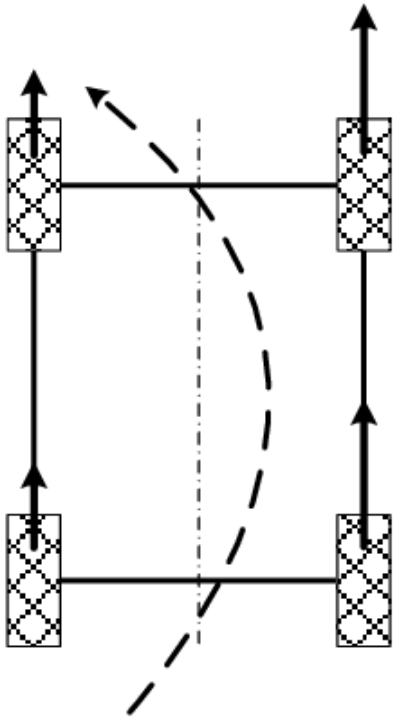

Figure 18: Performing a left turn using differential steering. Vectors displayed for each wheel indicate speed. [7]

6.4. Hardware Components

6.4.1. Motors

Motors are devices that convert electrical energy into mechanical energy. This is achieved by having an electric coil between two permanent magnets as shown below,

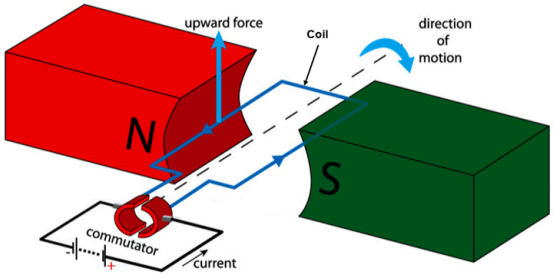

Figure 19: Components of an Electric Motor [8]

The current passing through the coil produces a magnetic field, which is then attracted to the magnetic field around it causing the coil to spin. The commutator is a rotary switch that ensures the magnetic field produced by the coil interacts with the magnetic field of the permanent magnets in a way that ensures constant rotation of the motor shaft in one direction[9]. The interaction of the two magnetic fields produces motion whose direction can be predicted by using the Flemming's left-hand rule which states that when the left hand's thumb, forefinger, and second finger are stretched out perpendicular to each other, if the forefinger represents the direction of the magnetic field and the second finger represents the direction of the current flowing through the conductor, then the thumb will point in the direction of the force experienced by the conductor. The force produces by passing the current through the coil can be used to control the wheels by attaching the wheels to the motors. To figure out what kind of magnets to use, we need to consider the total weight, W of the prototype we want and the resultant opposite forces acting on the wheels' motion. This means that cart at rest, will experience force equal to the product of its mass and the frictional force coefficient. For the cart to move the motors must produce enough torque to overcome this static force and continuously produce energy to keep the cart moving. While in motion the cart experiences friction force due to the contact between the wheels and the surface(concrete) and the rolling friction force. The calculation below will show how we choose our motors and the constraint we consider for our prototype.

Chosen Cart Parameters:

Total Mass M of the prototype $M = 10$ kg

Weight, $W = 98.1N$

Wheel radius, $r_{\text{wheel}} = 0.0485m$

Anticipated velocity $v = 1.5\frac{m}{s}$

Assumption:

We are starting from a stationary position.

To make the motors move the following must be achieved:

- Overcome static friction (concrete $\mu s = 1.0$)
- Maintain speed by overcoming kinetic friction (concrete $\mu k = 0.8$)
- Need to accelerate to $1.5\frac{m}{s}$ in x second time.

6.4.2. Forces Against the Wheel Calculation

Static Calculation

$$\begin{aligned} \text{Fsf} &= \mu s \times W \\ &= 9.81\frac{N}{\text{kg}} \times 10 \text{ kg} \\ &= 98.1N \\ &\approx 98N \end{aligned} \tag{2}$$

Kinetic Calculation

$$\begin{aligned} \text{Fkf} &= \mu k \times W \\ &= 0.8 \times 98.1N \\ &= 78.48N \end{aligned} \tag{3}$$

Rolling Friction Force, where C is the coefficient of rolling friction force

$$\begin{aligned} \text{Frf} &= CW \\ &= 0.65 \times 98.1N \\ &= 63.77N \end{aligned} \tag{4}$$

This implies that, to move a 10 kg cart on a concrete surface with rubber wheels from rest. We would need the motors to exert $90.1N$ of force. To continue moving the cart after overcoming the static force, we will need $142.25N$ of force, which is the weight cart in motion and the force due to the rolling friction force.

6.4.2.1. Torque Calculation

Torque is a measure of force that causes an object to rotate around an axel.Given that wheels of radius is r and Force is F, the magnitude of the torque can be expressed as[10]:

$$\tau = F \times r \tag{5}$$

If r=$0.0485m$ and F = $90.1N$, we can find the torque as in (6)

$$\begin{aligned} \tau &= 98.1N \times 0.0485m \\ &= 4.75785 \text{ Nm} \end{aligned} \tag{6}$$

This means that we require approximately 4.8 Nm of force to be generated by motors to overcome static force allowing the cart to move from rest. With four motors, one for each wheel, the 4 motors will share the weight of the cart meaning they will each carry a quarter of weight. Therefore the torque on each motor will be $1.2N$

$$\begin{aligned} \text{RPM} &= \frac{\text{speed} \times 60 \text{ sec}}{2\pi \times r_{\text{wheel}}} \\ &= \frac{1.5\frac{m}{s} \times 60 \text{ seconds}}{2\pi \times 0.0485} \\ &= 295.39 \\ &\approx 300 \end{aligned} \tag{7}$$

The motors need to operate at 300 rpm to move at the speed of $1.5\frac{m}{s}$

$$\begin{aligned} \text{Torque} &= 1.2 \text{ Nm} \\ \text{RPM} &= 300 \end{aligned} \tag{8}$$

Angular Velocity, ω Calculation

$$\begin{aligned} \omega &= \frac{\text{RPM} \times 2\pi}{60 \text{ seconds}} \\ &= \frac{300 \times 2\pi}{60s} \\ &= 31.42\frac{\text{rad}}{s} \end{aligned} \tag{9}$$

Power Calculation

$$\begin{aligned} P &= M \times \omega \\ &= 1.2\text{Nm} \times 31.42\frac{\text{rad}}{s} \\ &= 37.7 \text{ Watts} \end{aligned} \tag{10}$$

From the power above, we can get the motors voltage and current rating using $P = IV$. Choosing a 12V battery, the current rating will be

$$\begin{aligned} I &= \frac{P}{V} \\ I &= \frac{37.7W}{12V} \\ I &= 3.14A \end{aligned} \tag{11}$$

From the calculation above, we can conclude that our motors need to be rated as follows:

Motor Specification	**Value**
RPM	300
Voltage	12 V
Current	> 3.14 A
Torque	> 1.2 Nm

Table 4: Minimum Motor Requirements for Optimal Operation of the Cart

6.4.3. Pico

For direct motor control, the Raspberry Pi Pico H was utilized. The Raspberry Pi Pico uses a dual-core Arm Cortex M0+ processor and provides all the necessary functions needed for motor control including analog out for PWM, digital in for feedback, interrupts, and I2C [11]. For early prototypes, testing, and design, the FRDM-KL25Z microcontroller, which uses an Arm Cortex m0+ processor, was used. Ultimately, the decision was made to switch away from the KL25Z microcontroller for several rea-

sons. The main reason it was chosen was because there was a limited supply of the KL25Z boards, making them hard to come by at a reasonable price. The Pico boards were more readily available and cost effective. Another reason was the Pico's flexibility when it came to deciding how to design and develop the motor-control software proved to be an asset. While the KL25Z can only be designed in base C/C++ using the MKL25Z4.h library, the Pico can be programmed using python with MicroPython, C++ with the C++ SDK supported by the Pico, and with C++ Arduino libraries as well. The Arduino adjacent Mbed library APIs were ultimately chosen as they provided many useful abstractions of core Arduino libraries and a strong and reliable debugging interface. Another reason for choosing the Pico over the KL25Z was because of its more user-friendly debugging interface with the serial output. When using the Keil Uvision development kit, there were many inconveniences with debugging. There was a very limited number of options when it came to using log and trace-based debugging techniques.

6.4.4. Motor Driver

Motor output, as discussed previously, varies based on the voltage being supplied to them. The driving motors of the cart are rated for 12V, and a PWM signal from the Raspberry PI Pico microcontroller, which outputs the signal at a peak of 3.3V. To operate the motors effectively, an external power supply is used to provide the required 12V, however it supplies a constant DC voltage. To step up the PWM signal with the 12V supply, a component known as a motor driver must be implemented between the Raspberry PI Pico, battery, and the motor [12]. This layout is depicted in Figure 20. Another feature provided by motor drivers is direction control of the motor. As explained in Section 6.4.1, the flow of current through the coils in the DC motor dictate the direction of the Electromotive Force (EMF), and therefore the rotation of the motor shaft. To control the direction, the shaft rotates, the current flow thought the motor must be controlled. One common method of implementing this control is using an H-Bridge circuit. The H-Bridge circuit, as depicted in Figure 20. controls the direction of current through the motor [13]. S1 through S4 represent either switches or MOSFETs,

which, when activated, allow current to flow through. Closing S1 and S4 allow the current flow into the positive terminal of the motor. If S3 and S2 were closed instead, then current would flow into the negative terminal of the motor, causing the motor shaft to rotate in the opposite direction.

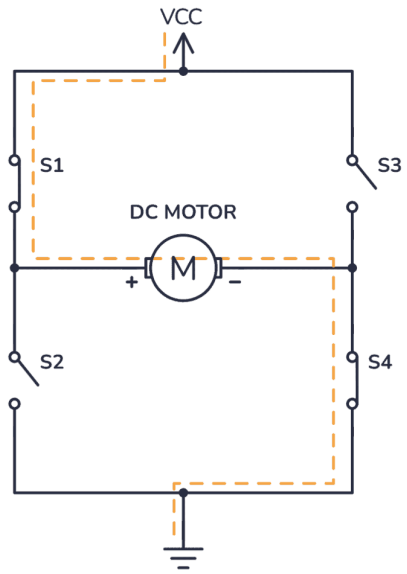

Figure 20: H-Bridge Circuit. S1 and S4 are closed, causing current to flow from the positive to negative terminals of the Motor [13]

Motor drivers are typically rated for a particular maximum current and voltage and must be selected with consideration to the stall current of the motor they operate. As stated earlier, the motors selected for the cart draw 360mA at no load, and up to 2.8A at stall. As such, the motor driver selected must be capable of handling at least 2.8A.

The motor driver selected for the cart was the BTS7960 High Current 43A H-Bridge Motor Driver [14]. This module can handle a peak current of 43A, which far exceeds the operational requirements of the selected motors. This driver was selected as it was available for significantly cheaper than other motor drivers capable of handling 2.8A, and due to a quick delivery that many other vendors did not provide. The motor driver is depicted in Figure 21, and the two BTS7960 B Half-bridge drivers are labeled.

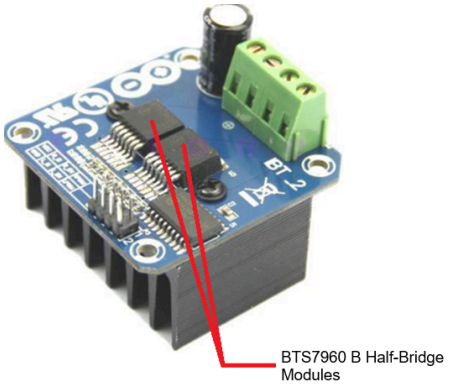

Figure 21: BTS7960 High Current 43A H-Bridge Motor Driver Module [14]

The two BTS7960 half-bridge drivers are composed of both a P-channel and N-channel MOSFET (metal-oxide-semiconductor field-effect transistor) each, to form a complete H-bridge. A MOSFET is a device that controls the flow of current between it's input source and output drain terminals using a control voltage applied to a gate terminal. Within the BTS7960, the control signal applied to the gate dictates whether the N-channel MOSFET or the P-channel MOSFET are enabled. In an N-channel MOSFET, when a positive voltage is supplied to the gate terminal, current flows from the drain to the source. Conversely, in a P-channel MOSFET, a negative voltage supplied to the gate causes current to flow from the source to the drain [15]. The control signal that allows the control of the MOSFETs and thereby the motor shaft's rotation is supplied from the input pins, which are depicted in Figure 22 below.

Figure 22: Input pin configuration on BTS7960 motor driver. Labels on Left side correspond with pins on right side [14]

VCC and GND act as the power and ground pins for the internal logic of the module. R_IS and L_IS are current sense pins, which provide an analog voltage output proportional to the current that lows through the right and left halves of the H-Bridge circuit. The R_EN and L_EN control pins enable forward and reverse drive on the module. R_EN activates or deactivates the right half-bridge, while L_EN controls the left half-bridge, and they must be signaled HIGH for the half-bridges to respond to the PWM signals. RPWM and LPWM are the forward and reverse PWM signal pins. The RPWM

pin is associated with one half-bridge, and the LPWM pin with the other, allowing for directional control based on which PWM signal is active.

In addition to this, the BTS7960 motor driver also provides protection against overtemperature, overvoltage, undervoltage, overcurrent and short circuits. These features help protect the motors from damage during operational use.

6.4.5. Encoders

While PWM signals being provided to the drivers individually allowed for the static setting of motor speed, wheel encoders were necessary to establish an on-the-fly feedback system that was based on the actual speed of the wheels. For this application, magnetic wheel encoders were attached to each of the four wheels which allowed for constant feedback on the wheel speed in RPM (Revolutions Per Minute). This is done by counting the number of pulses needed to complete a single revolution of the wheel. Magnetic wheel encoders count the changes in north and south poles using something called a Hall-Effect sensor to return the number of pulses over a set period [16]. This is shown below in Figure 23:

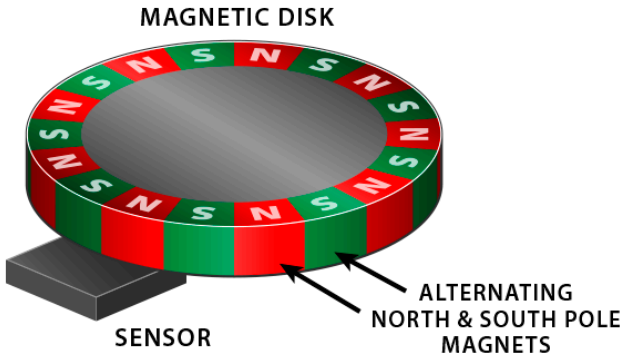

Figure 23: Magnetic wheel encoder showing the hall-effect sensor and magnetic pole pattern [16]

This period can then be artificially set using a software timer interrupt where the number of pulses is collected over the set period and then used to calculate the RPM. This mechanism is essential in performing accurate turns as well as establishing closed-feedback loops with PID controllers, which are further discussed in Section 6.5.1. In

the case of the encoders on this cart, six pulses would mark an entire revolution prior to being geared down, so there would be about ninety pulses per revolution when looking at the output shaft. This would in theory yield highly accurate results but there were several issues that limited this. The unpredictable physical conditions and mechanical limitations of the wheels, ground, and chassis resulted in the turns and adjustments being less accurate than expected. The greatest limitation in the application o

6.5. Software Control Systems

6.5.1. PID Speed Controller

Motor shaft RPM is dictated by the power supplied to the motor, but external forces such as friction from surface contact and applied loads act to reduce this performance. When constructing a system with 4 motors, these forces may act differently on each of the motors, which will result in each motor performing differently, and therefore inhibiting the cart's ability to function as intended. Another consideration is variable loads being applied to the cart, where adding more weight will reduce the operating speed of the cart below the required speed due to increased frictional forces.

A PID speed controller forms a closed loop feedback system with the motor and the microcontroller, providing reactive control over individual motors to achieve consistent operation. With a PID controller, a target speed, called the set point, can be selected. Using the actual RPM speed values determined using the motor encoders, the difference between the current speed and set point speed can be determined, and the PID controller will generate a new PWM control signal to deliver to the motors. Through this dynamic signal, the motors will adjust their output torque and speed relative to the external factors that interfere with their operation.

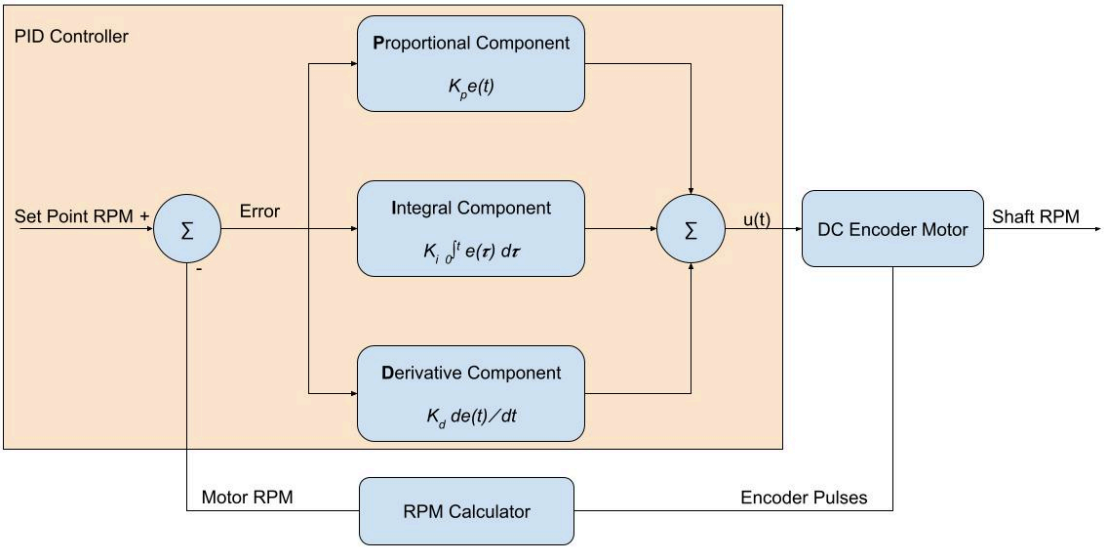

Figure 24: Block diagram of closed loop feedback loop with PID controller

The PID controller generates this control signal, $u(t)$, using three components: the proportional, integral, and derivative component. Each of these components generates a response based on the feedback error, and their individual responses are summed to generate a complete control signal. Each component can be adjusted and tuned using a gain value, K, which dictates the strength of a specific component [17]. The control signal that is generated is applied to the motor in the form of a PWM signal.

The proportional component depends solely on the error value. The proportional gain value, K_p, determines the strength of the output signal, as shown in Equation 12:

$$K_p * e(t) \tag{12}$$

For greater values of K_p, a larger control signal would be generated, resulting in a larger adjustment in motor output RPM. Values of K_p that are too large will result in very strong reactions to small errors and will cause the oscillations in the control signal as the errors shift from positive to negative relative to the set point. This can result in unstable systems. Smaller values of K_p may result in reactions that are minimal and can cause the system to react slowly to changes in operational condition [17]. These types of responses can be seen in Figure 25, where different values of K_p are shown relative to the set point, or reference.

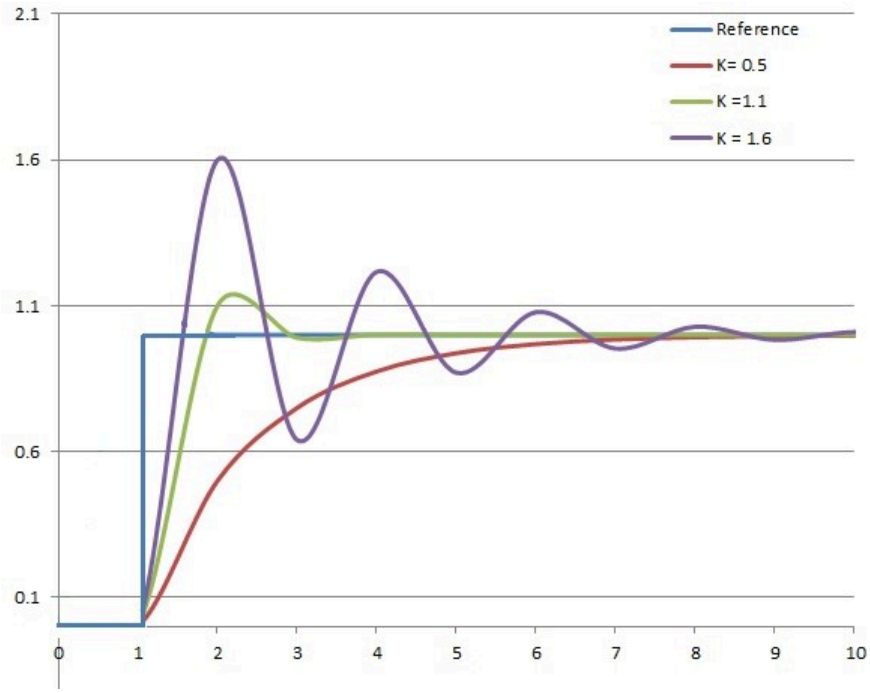

Figure 25: Motor speed response to varying values of K_p [18]

The Integral component reacts to the error over a period. This results in the component reacting minimally to large errors but acq ting continuously to reduce the accumulated error of the system. This helps the system slow arrive to a steady state error, where the system no longer oscillates. The gain variable K_i dictates the strength of the integral response and must be tuned to allow for the steady state error to reach zero [17]. This relationship can be seen in Equation 13:

$$I(t) = K_i \int_0^t \mathrm{e(t)}\ \mathrm{dt} \tag{13}$$

The effects of various values of the integral gain, K_i can be seen in Figure 26.

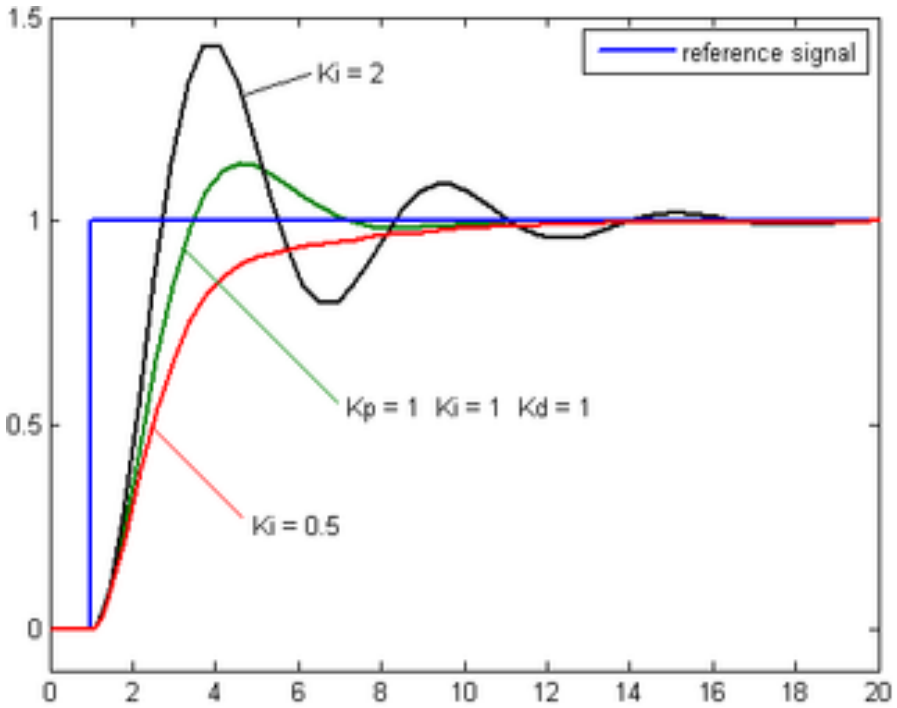

Figure 26: Motor speed response to varying values of Ki [18]

The derivative component controls the rate of change of the signal, slowing it down if the rate of change is very large. This relationship is shown in Equation 14:

$$D(t) = K_d \frac{\mathrm{de(t)}}{\mathrm{dt}} \tag{14}$$

increasing the value of the gain K_d results in the system to generate a stronger signal to oppose the rate of change [17]. The response for different values of K_d can be seen in Figure 27, where the larger values for K_d reduce the rate at which the signal climbs, and also smooths out the fluctuations .

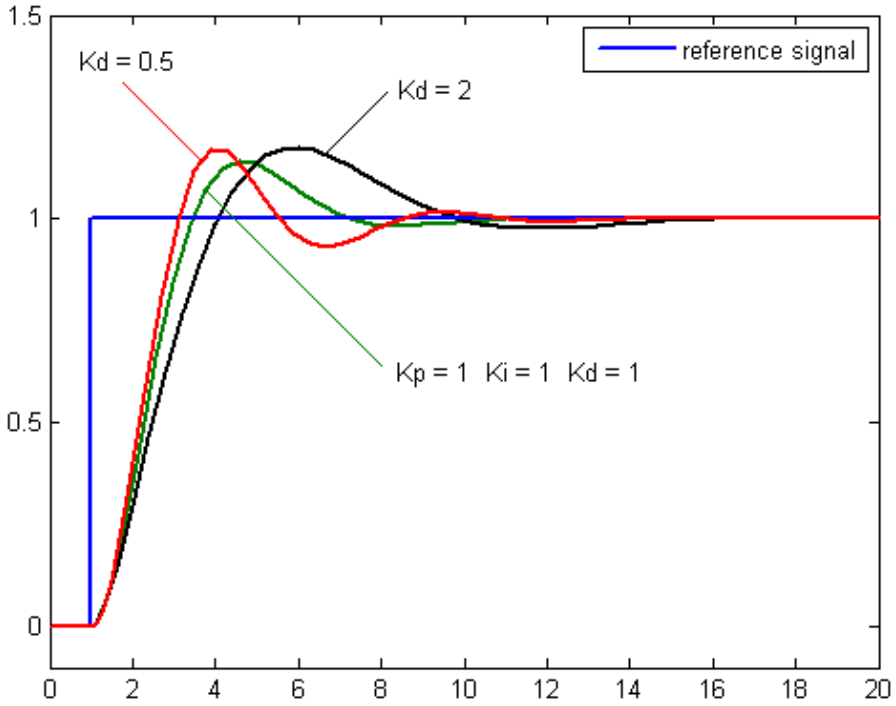

Figure 27: Motor speed response to varying values of K_d [18]

In the implemented system, the derivative component was omitted. Because of the component's tendency to react to a signal with a large rate of change, it often perceives noise in the system as instances where a large change occurred over a very small-time interval. It would then react strongly to this perceived inconsistency and disrupt the operation of the system [17]. Another factor is that since motor speed is calculated every 0.1s, the control loop operates relatively slowly, and as such large adjustments are common in short intervals. To implement the derivative component, the derivative gain would be required to be set to an almost insignificant value, which would make its inclusion redundant.

Each of these individual components are summed together to generate the control signal to supply the motors, as shown in Equation 15:

$$u(t) = P(t) + I(t) + D(t) \tag{15}$$

For the speed controller, the derivative component will be dropped, for the aforementioned reasons. As such a controller used is a PI controller, which can be represented similarly in Equation 16:

$$u(t) = P(t) + I(t) \tag{16}$$

The process of tuning the gain variables is primarily done by trial and error, where different values for each are set, and the system's ability to respond are evaluated. The proportional component is adjusted first to perform the adjustments, then the integral component was tuned to ensure that the system can reach a steady state error of zero. The pseudo code of the controller is shown below:

```
set_point = 100 RPM
error_integral = 0

# PID Speed Controller Method
get current_time in microseconds for adjustment
calculate time difference, deltaT, since last adjustment

loop: # for each motor
    get current motor speed synchronously
    calculate error between set_point and motor speed

    # proportional component
    kp = 0.7

    # calculate integral component
    ki = 2
    update error_integral with intergal calculation (e * deltaT)

    # generate control signal
    u = (kp * error +  ki * error_integral)

    # convert control signal to pwm signal (0-100)
    u = u/255 * 100
```

6.6. Turning

Steering of the motors was achieved by means of individually controlling the wheels and giving them different but ratioed voltage as discussed earlier in differential steering section. To achieve turning and adjusting the cart's motion we decided to control each wheel individually. We aimed to achieved two types of turning motion, these were pivot turning and slight turning to adjust the path of the cart while moving. These turns were achieved by supplying each motor different PWM signal. For instance, to achieve pivot left, the wheels on the left side of the cart were directed to spin backwards using the motor controller, and the right-side wheels were directed to spin forwards. Adjusting the orientation of the cart while moving was achieved by supplying the motor system with an angle to a point relative to the cart from the map. The angle was then used to adjust the orientation of the cart while in motion. This

was achieved by speeding up one side of the motor and maintaining the same speed on the other side of the motor.

To figure out how much force to give the wheels to facilitate turning the following formula and variables were used:

$$
\begin{aligned}
D &= \text{diameter of the wheels} \\
H &= \text{diagonal distance of between the wheels} \\
L &= \text{the length of the distance between the front and the rear wheels} \\
W &= \text{the distance between the wheels} \\
\theta &= \text{angle of turning} \\
N &= \text{Number of wheels turn to cover a distance x} \\
C &= \text{Number of encoders counts for one revolution of the wheel}
\end{aligned}
$$

The above distances were measured and used to get the formulas to be used to control the velocities of each wheel while turning. To achieve turning, each wheel will make a circle with the pivot wheel being the centre of turn. To get the RPM to supply to the wheel opposite diagonally to the current pivot, we find the total circumference of the turn.

$$
\begin{aligned}
&\text{Circumference of the turn} = 2\pi H \\
&\text{Circumference of the wheel} = \pi D
\end{aligned}
\tag{17}
$$

By dividing the total distance by the circumference of the wheel we can find how many times the wheel must turn to reach the total distance.

$$
N = \frac{2\pi H}{\pi D} = 2\frac{H}{D} \tag{18}
$$

Multiplying the above equation by number of encoders counts for one revolution gives us how many encoder pulses it takes to make a full 360 degrees turn.

$$
\#\text{ of encoder turn for a full turn} = NC \tag{19}
$$

To tailored down to a degree, we multiply the whole equation by $\frac{\theta}{360}$

$$
\begin{aligned}
\#\text{ of encoder turn for an angle of Ang} &= \theta * \frac{NC}{360} \\
&= \frac{\theta}{180} * \frac{H}{D}
\end{aligned}
\tag{20}
$$

From the above derivation, we can deduce the general formula for calculating number of encoders counts for a turn of any given angle by the formula below.

$$n = \frac{\theta}{180} * \frac{Hc}{D} \tag{21}$$

for H = diagonal distant

$$n = \frac{\theta}{180} * \frac{Lc}{D} \tag{22}$$

for L = distance between front and back wheel

$$n = \frac{\theta}{180} * \frac{Wc}{D} \tag{23}$$

for W = distance between either front or rear wheels

Figure 29: Chassis of the Cart Prototype[19]

Example of Calculations for a 90 degrees turn From the figure:

$$
\begin{aligned}
L &= 0.33m \\
W &= 0.20m \\
H &= 0.44m \\
C &= 6 \\
n_w &= \frac{90 * .20}{180 * 0.097} * 6 = 6.2 \approx 6 \\
n_L &= \frac{90 * .33}{180 * 0.097} * 6 = 10.2 \approx 10 \\
n_H &= \frac{90 * .44}{180 * 0.097} * 6 = 13.6 \approx 14
\end{aligned} \tag{24}
$$

This therefore means to make 90 degrees turn we need to give each wheel encoder counts as shown above.

6.7. Prototype I

An initial prototype was designed with the intention of testing the movement and turning mechanics explained previously. A small cart was acquired, which consisted of two 6 V DC geared motors. The small form factor meant it was easier to construct and test with, while still functioning under the same principles as a large cart. The main difference between this prototype and the intended design is that the prototype was a two wheeled model, as opposed to the four wheeled proposition. This simplified some aspects of turning, but the theory and tests were transferable to the second prototype later. An image of the prototype is shown below in Figure 30:

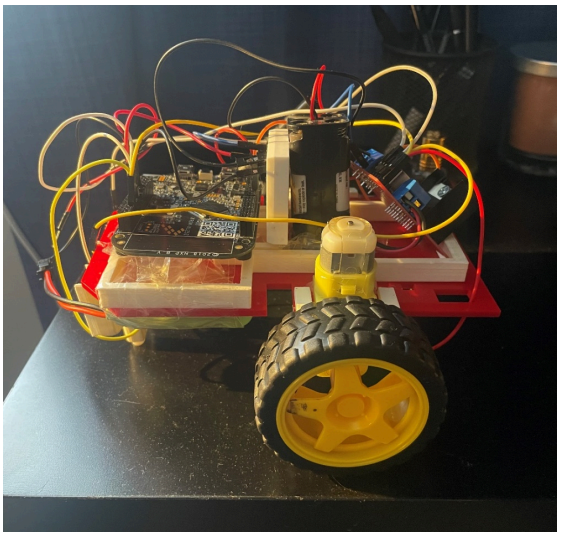

Figure 30: First cart prototype, controlled by the KL25Z MCU

Although the mechanics were transferable, there were several limitations with the prototype that restricted the number of features that could be tested. This was primarily due to the motors attached not being mounted with any encoders, and as such, motor speed could not be calculated, accurate turns could not be performed, and a closed loop feedback system with a PID controller was not possible to implement. These were factors that were deemed necessary for the second prototype constructed.

6.8. Prototype II

The second prototype featured a four-wheel design, with four 37mm 12V DC geared motors with encoders attached. The frame, motors and wheels were all purchased as a single set, making the installation a simple process. The cart was modified to include a platform to allow for space for electronic components, as well as a section for the battery, as shown in Figure 31 below.

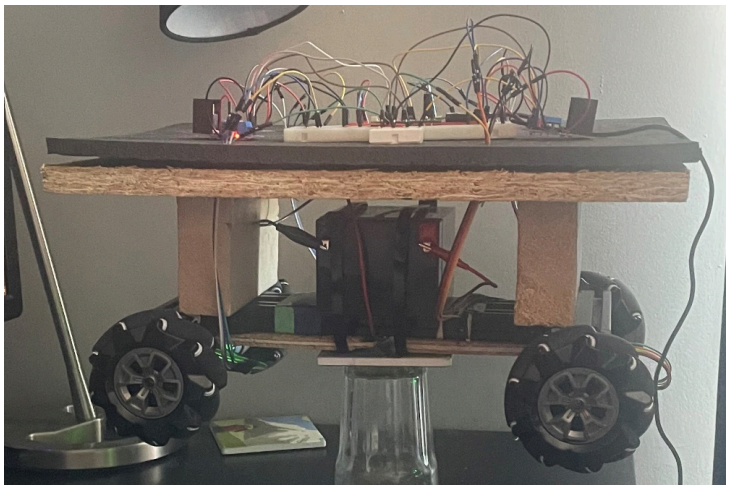

Figure 31: Image of second prototype cart with first platform and battery mounted

With this prototype, the mechanics with four wheels were verified, and, with the inclusion of encoders on each of the motors, RPM calculations, accurate turns, and a closed loop system with a PID controller for each motor could be tested.

There were several challenges encountered with the second prototype, including issues with weight distribution and the encoders. Given that the cart was purchased online, and assembled by hand, it was noticed that when the cart was run on the ground, some of the wheels did not make good contact with the ground and would

not gain enough traction to drive the cart. This would result in the cart not being able to go straight forwards, and always veering in one direction. It was assumed that this issue would be resolved when the platform is attached and battery is mounted, as the increased weight would press it down, but this only partially resolved the issue, as it was still clear that certain wheels were not carrying the same load as others. To alleviate this issue, the platform was re mounted in such a way that the wheels not making proper contact with the surface were carrying more weight. This helped to some extent, but the issue remained that some wheels had greater traction than others. Another problem that was encountered was that the encoder mounted on the front left wheel was broken, and no signals were readable from it. This meant that the front left motor was unreadable and could not be regulated accurately using a PID controller. This also made turning less accurate, as now an average of the encoder pulses elapsed on the left side was not entirely accurate. This caused complications in making the cart drive straight and would cause complications in the application of managing varied loads. A replacement motor and encoder were ordered; however, they did not arrive with enough time to install and test it. It was opted that the PID controller would remain, but both the front left and back left wheels would receive the same control signal based on the operation of the back left wheel. This was not a perfect solution, as the weight distribution on the cart and surface contact are not always the same on both wheels, but it was deemed sufficient for the prototype.

This prototype was used to test all the mechanics and features intended to be implemented and was used as the platform that would become the final design. Integration with the LiDAR and mapping was performed on this cart, and a new layer was added to support the LiDAR for localization and collision avoidance.

6.9. Schematic

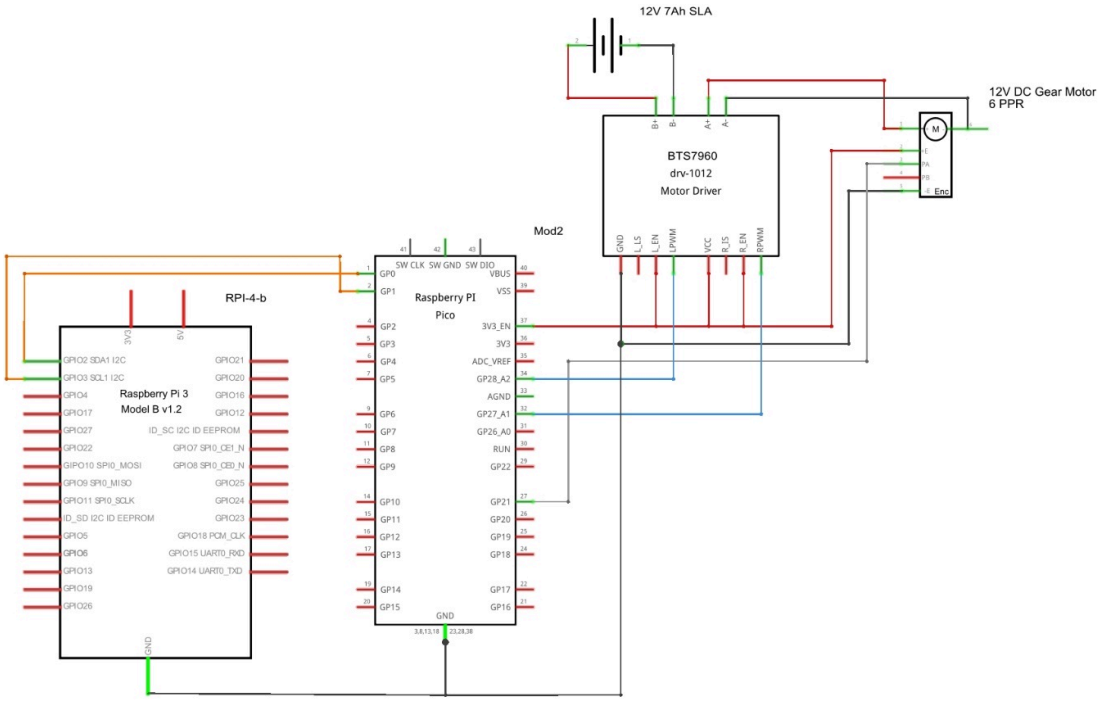

Figure 32: Schematic of Motor Control System

Figure 32 above shows a schematic containing the detailed pin mapping of the modules and their respective connections. The schematic was scaled down to only show one of the four motor connections. The motor shown in this schematic represents the front right motor of the vehicle. The schematic shows the wheel encoder interrupting on the Pico on GPIO 21 which is configured as a digital input pin. The motor driver PWM control is done by providing a reverse PWM signal and a forward PWM signal connected to the Pico's analog out GPIO 27 and GPIO 28 pins respectively. The L_EN and R_EN pins of the motor driver are driven at a constant logic high through the Pico's power supply as direction control is done exclusively through the two PWM signals. The 12V battery is connected to the motor driver directly to provide power to the motors. For I2c communication, both the SDA and SCL buses on the Pico are connected to the SDA and SCL pins of the Raspberry Pi. The pins being used for SDA and SCL are GPIO 0 and GPIO 1 respectively. The I2C mechanism is discussed in more detail in Section 9.1.

6.10. Future Work

The intention of the final product was to demonstrate a proof of concept on a smaller scale, with an emphasis on showcasing the cart's mobility and ability to handle inputs

from external modules. To ensure mechanical flexibility in the many possible applications of the cart, the cart would need to be scaled up in size and load-handling capability. This would require more powerful motors and wheels capable of carrying the higher loads necessary in shopping applications for example. More space for placing shopping items on the cart would be available after sizing up. The cart was kept at a smaller size due to financial limitations given the strict budget provided by the department. There were also some mechanical limitations when considering the equipment and time available and the expertise of the team. Therefore, the cart primarily acted as a smaller version of a potentially larger, and more mechanically complex product due to the complexities that would hold and because it allowed for the easier testing of fundamental software and hardware mechanisms.

7. 2D-LiDAR Localization

7.1. Introduction

The RPLIDAR A1M8 measures the time it takes for a reflection to be received, and uses $d = ct$, where c is the speed of light and t is the time taken divided by 2, since the LiDAR sensor is only attempting to measure the distance between itself and the surface the beam of light reflects off. This is important to note due to the sensor doing the same procedure for an entire revolution. The sensor gives a distance value only if a reflection beam returns to the sensor. If the angle at which the LiDAR hits is too obscure then the light beam is not reflected back into the sensor, therefore a reading cannot be obtained. It is best to imagine a person standing next to a LiDAR in a big field, at 0 degrees it sees the distance at which a person stands next to the LiDAR, at 0.33 degrees the LiDAR does not get a reflection as the field is so vast that it does not return a reflection within the range of the beam of light. Even though the difference in angles is so miniscule, there would still be no reading returned as to also say that possible interpolation is pointless due to know prior knowledge of any obstacles by the LiDAR, the data collection occurs in real time with no memory.

After an entire revolution done by the LiDAR, an image can be obtained at what the LiDAR saw at time X. If this is compared with a known map/location and the LiDAR is best matched what the global map is then you can locate where the LiDAR was. If a LiDAR image can be compared with something global then a location can be obtained and if a location can be obtained then a trajectory/path can be planned for, dependent on the accuracy of said location. For reference, LiDAR localization is very prevalent for autonomous vehicles particularly in agricultural, planning a route of where to feed certain crops water, depending on the vehicle's position. Simultaneous localization and mapping (SLAM) is used to realize a relative image and relate it to a global map which is mainly used in robotics. This was a good solution to our use case, but it was thought of from the planning stages that it would be much better to make a system and understand vehicle positioning to a deeper sense, which was done to a good standard as illustrated below.

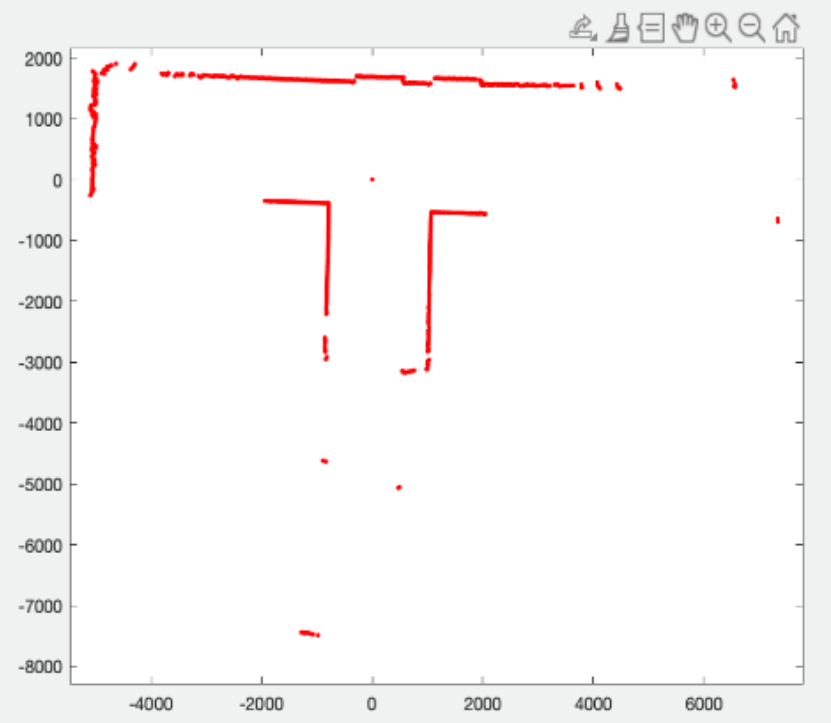

Figure 33: RPLIDAR A1M8 image within 1 revolution

7.2. Initialization of map

7.2.1. Orthogonal Regression

Once a plan was known for how to achieve LiDAR localization, there had to be steps taken towards realizing what the LiDAR would be localized to. After conducting some research, it was seen that it would be a good idea for the LiDAR to create its own map. Therefore, to realize the layout of the shopping centre or would mean that before invoking the robot to collect items it would need to do one thorough loop. This is to realize where its current position is relative to what it can see which is always something within the map. It is key to note that we are always comparing to the map with the possibility that the robot is tested miles and miles away from the actual shopping centre, the map is always the primary reference. The way that real-live map forming can be done is starting in a position, turning on the LiDAR and visualizing what it sees. LiDAR readings can be very inconsistent or non-existent due to where/what the light beams are interacting with. Hence, clusters had to be formed to remove noise or the null readings. The sensor does sends out beams of light at around 0.30 degrees, but it is not very consistent at this hence noise is created, readings at 0.15 spanning to 0.45 are

associated with 0.30 degrees since the sensor has an actual frequency of measurement every 0.30 degrees, as mentioned in Section 5.3.1. Lines of best fit to outline a structure would normally be using Mean Squared Error (, but this is with the notion that you have one fixed axis, meaning that the X is always stable (same value) and Y is varying. However, with the LiDAR data, both X and Y are changing, even though through the actual data points, as shown in the figure below, give the trajectory or rising (being a purely vertical line), the noise will cause the line of best fit to be a diagonal. This is because both the X and Y are changing and linear regression, another name for MSE, always attempts to match the line perfectly vertical to the line of best fit, trying to create the line of best fit with the shortest translation needed for points in dataset, the dataset in this case being that of the LiDAR.

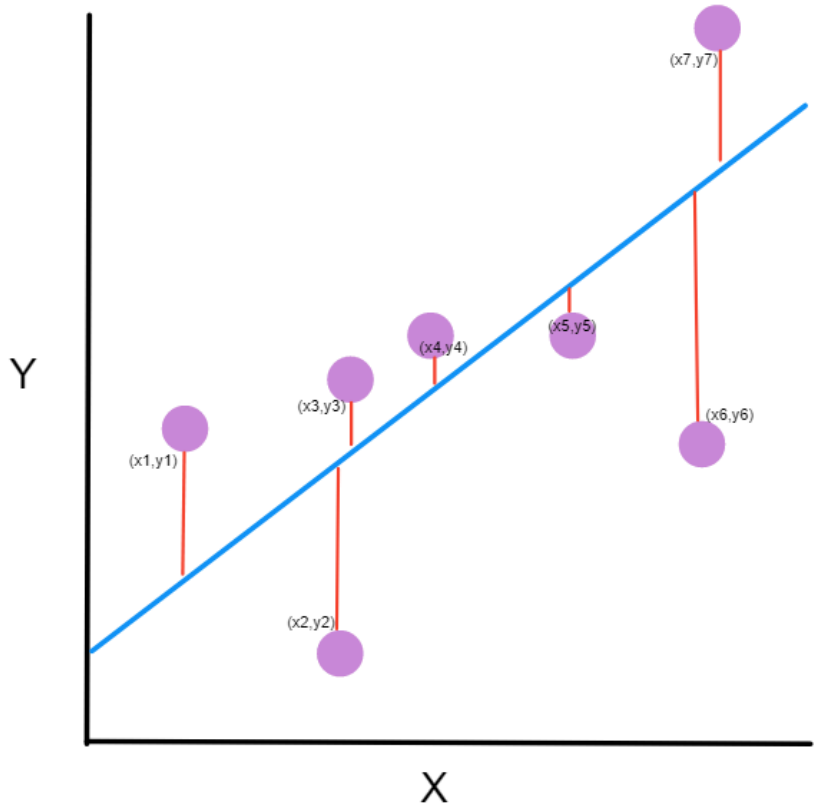

Figure 34: How linear regression creates a line of best fit [20]

Orthogonal Regression or Deming regression creates a perpendicular line between data point and a line of best fit, where the sum of all these points is at a minimum is where the line of best fit should be, this is illustrated in the figure below:

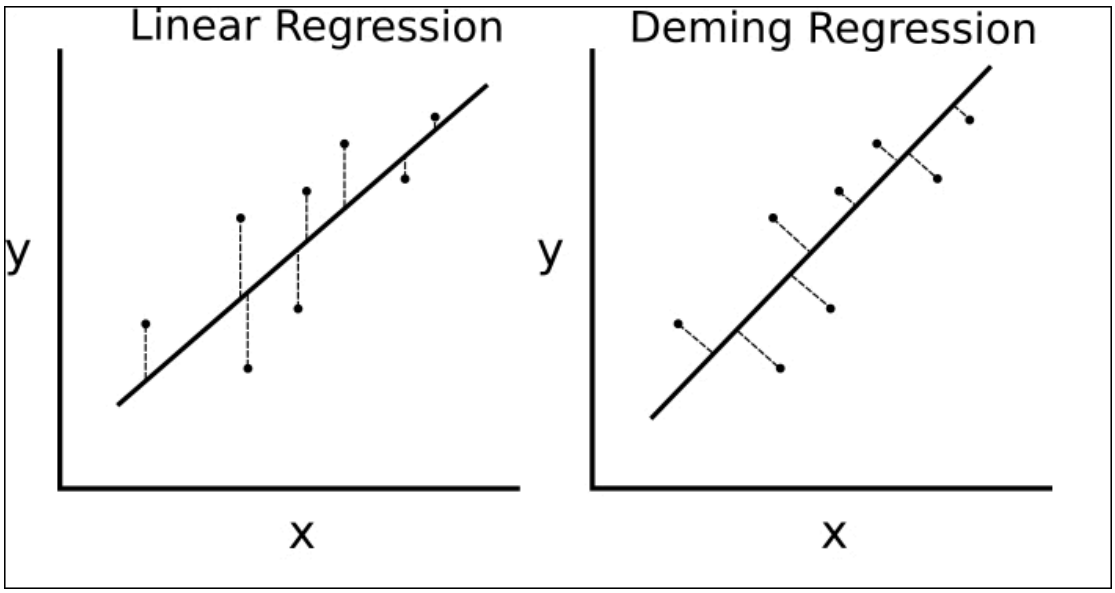

Figure 35: Deming Regression and Linear [21]

Once getting a line of best fit then we have an outline of a structure even at points where there are no readings. As mentioned, if clusters within threshold can be connected then we can outline a structure, even at 0.33 degrees there are no readings but there are at 0.60 degrees and at 0 degrees and the distance between these two points is less than T meters where T is threshold then these two points belong together, irrespective of 0.33 degrees returning no distance value.

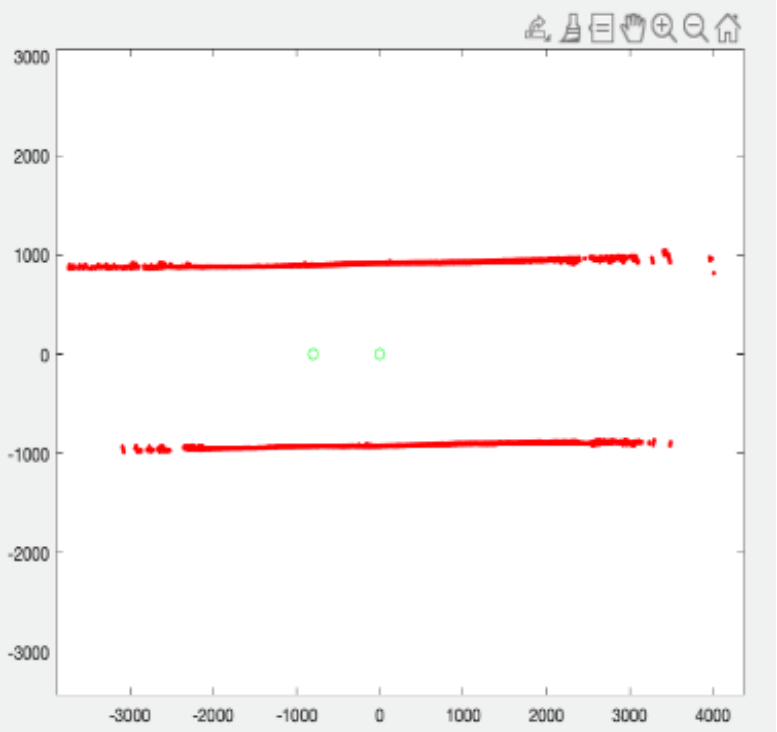

Figure 36: Two figures concatenated where the black lines demonstrated a 'correct' line of best fit laid on top of datapoints captured by LiDAR, in red

Concatenation of static measurements was done to a good standard, as illustrated below and showed the capability of creating a real-time map. There was a minor issue with corners as they would belong to the same cluster, where there should be two line of best fits as opposed to 1. While there was great efficacy to creating a map using this solution, it was thought to create a preloaded map as opposed to a real-time solution. This was a good study into mathematical proofing and understanding Deming regression.

7.2.2. Preloaded Map

A preloaded map was done by measuring out where the supposed shopping centre should be, it was modelled on the fourth floor in Mackenzie building. A version of the map was created which did not account for intricacies such as the depth of doors and glass, to achieve a higher precision value for position this must be created. A metric for error will be referenced later on in this section and for this metric to be at a minimum, the reference map will have to be as detailed as possible.

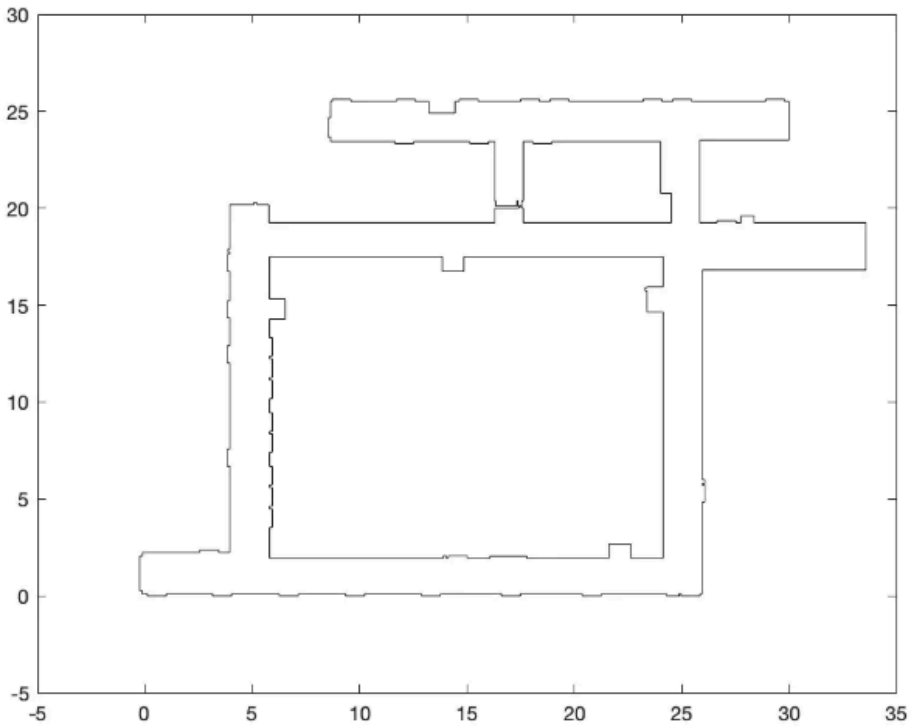

Figure 37: Measured map realized into software

7.3. Test Positions

After this map was realized into software, it was designed whereby there would be reference positions plotted all throughout the map. These would be simulated and would emit lines every 10 degrees to give a picture of what the sensor sees at position X. The figure below demonstrates how these positions look like and how one individual test position would collect data. This data is like that of real data because it stops once the beam of light reaches an obstruction. Position of reference points start from min (x,y) to max(x,y).This picture is shows less of which test positions are used, in actuality there are 22000 test positions in this structure, one every 10cm.

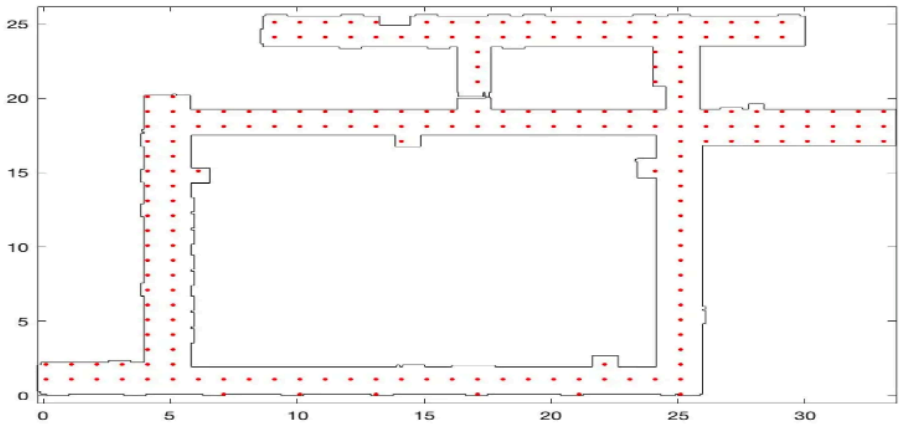

Figure 38: Test positions plotted throughout map

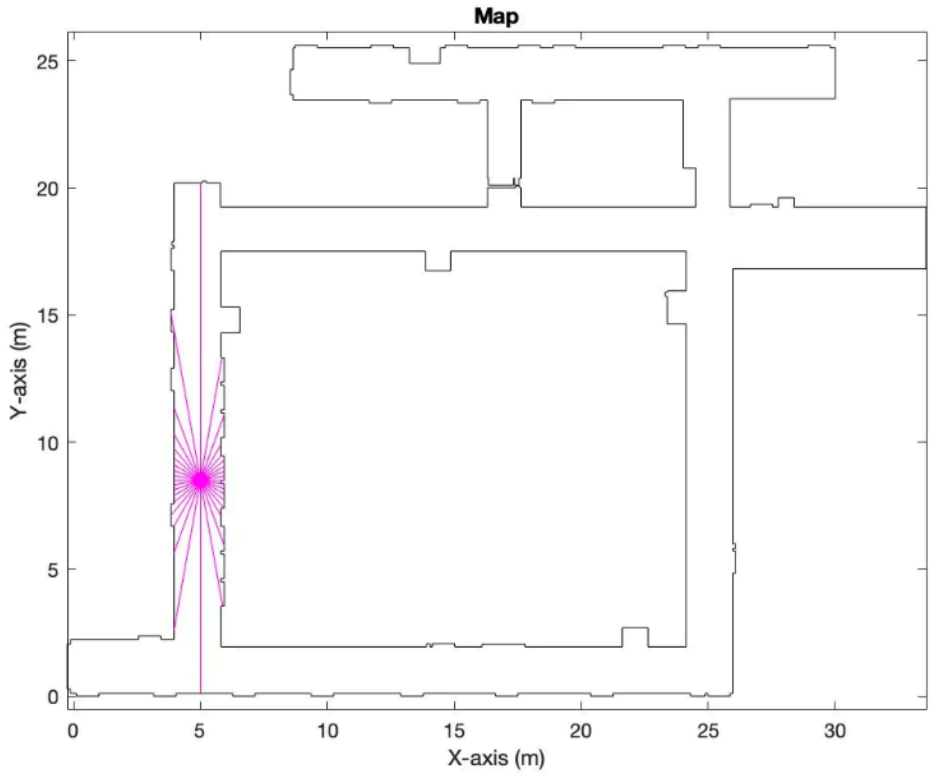

Figure 39: Simulated data captured of test positions

7.4. Orientation

Orientation in an issue to be solved because it gives a sense of odometry, even when we know the correct position, we can have an error metric for position, but this cannot show the direction of motion. This would be fine in a static case but for this use case orientation must be solved. To do this, a best position must be found first and then an error metric must be calculated at every 10 degrees, where the error metric is at a

minimum, is the direction in which the robot is facing. To do this, best to shift all the angles to a global axis so the measured data and the reference data are compared the same, therefore the real data must be shifted to polar 0, which is on the positive X-axis. This is imperative in figuring out how to move and turn the vehicle.

7.5. Localization Algorithm

The localization algorithm goes as follows:

```
Load Pre-saved Map
Load Pre-Saved Test Positions and their measurements
Collect LiDAR data
Shift LiDAR data to have 0 degrees start from 90 degrees
Initalize angle list between 0 to 360, every 10 degrees
Set last position to be NaN

For i = 1: length(angle_list)
  Current_angle = angle_list(i)
   Define theta range, LiDAR is not specific so must have a range to
associate with a certain angle.
  Measure amount of times LiDAR measures at 0 degrees, gives weight of
angle. Weights at angles appended into an array
  Mean_distance = mean (distance at the indexes where 0 is)
  If weight == 0
    Mean_distance = 0
  End

End

For j = 1: length (Test Positions)
  Circular shift of all Angular Weights by the angle currently testing,
If at angle 180, shift by 18 since it is the 18th index.
  Do the same process for shifting the mean distance
   $"Metric" = sum(abs (D_"test_position"(j) - "mean_distance"(j)) *
"Angular Weight"); $
  [Minimum metric(j), minimum index(j)] = min(metric)
End

[~,Idx] = min(Minimum metric]
Test_index = min Index(idx)
Approximated Position = Test_position(Test index)
If last position is available
This approximated position must be within a meter of last_pos and must be
facing forward (180) of last position
Orientation = angle_list(test_index)
Precise angle = orientation + (-5:0.5:+5)
For na = 1:length(precise angle)
Current angle = precise angle(k)
```

```
For nx = 1:length(angle_list)
  Angle = angle_list(k)
  Look_angle = angle – current_angle
   Figure out theta range for look angle, as identified before to get
weights.
  Mean distance = mean(distance(theta_range indexes))
  If weights == 0
    mean distance =0
  end
end

Precise_metric = $sum(abs("Known Approximated position" – "Mean Distance")
* "Precise Weights")$
[~,minIdx] = min(Precise_metric)
Angle = precise angle (minIdx)
Send the angle(precise angle), approximated positon (x and y values) to
python program.
```

7.6. Precise Angle

Importance of precise angle is that when you are far from a point of interest in distance but the angle is very narrow, without the use of precise angle value it will not adjust properly and the vehicle will hug the wall which is not good. The use of precise angle keeps the vehicle always adjusting so it stays on a consistent path, almost straight line. The use of 0.5 is good enough for precision and fast enough for computation time. In comparison, the loop to find position runs through 22000 test positions while the precise angle just runs 20 times due to 0.5 degree increments, the computation processing would be very minimal to find and would be very helpful to keep the cart from avoiding walls. Not to be mistaken with avoiding obstacles, walls are accounted for in the map already. The successful implementation of 2D-LiDAR localization leads to a few opportunities for future improvements. The RPLIDAR A1M8 system's capabilities have been successfully used for basic localization and mapping in a controlled setting. Future improvements will focus on strrenghtening localization accuracy and system robustness in dynamically changing and busy environments, where the existing setup may have limits.

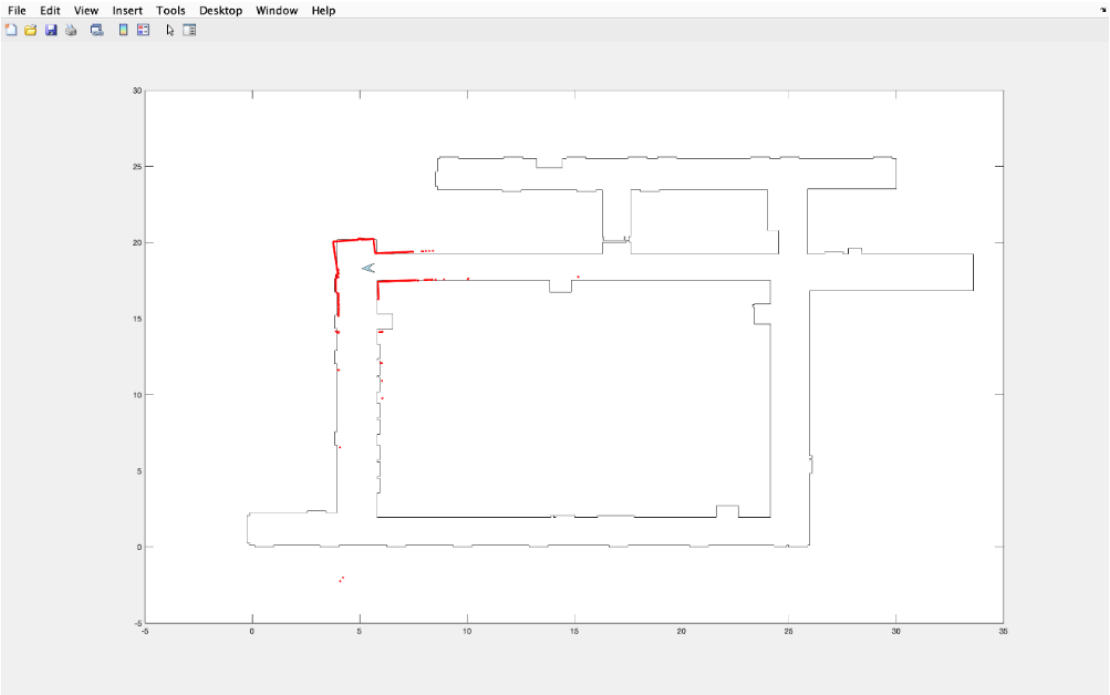

Figure 40: LiDAR positioning working in real-time

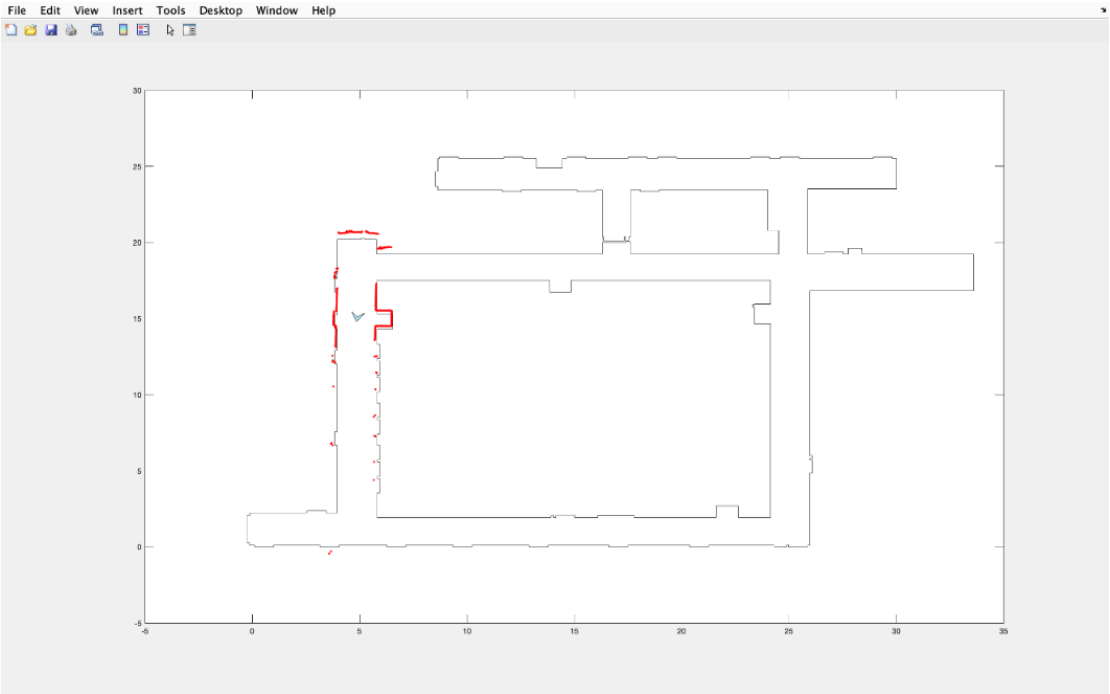

Figure 41: LiDAR positioning working after shift in position and orientation

7.7. Future Works

The successful implementation of 2D-LiDAR localization leads to a few opportunities for future improvements. The RPLIDAR A1M8 system's capabilities have been successfully used for basic localization and mapping in a controlled test area with minimal bystanders. During testing, we noticed that when the number of people surrounding

the robot reaches ten or more, they start to interfere with the LiDAR readings that only expects to see walls, for the most part. Future improvements will focus on strengthening localization accuracy and system robustness in dynamically changing and busy environments, where the existing setup may have limits.

7.7.1. Improving Localization in Crowded Environments

In the future, we aim to improve our localization system to guarantee high accuracy even in crowded places. One way of doing this could be to place a depth-sensing LiDAR on the robot. This sensor could detect specific symbols or "signatures" attached to ceilings or high on walls, which tend to be less likely to be concealed by humans and other obstructions. These symbols can be artificially placed to be specifically recognizable, or we can also leverage existing objects found on ceilings such as light fixtures, fire alarms, and signage. By referencing the markings against a pre-mapped pattern, the system could continuously readjust the robot's position with greater accuracy.

7.7.2. Enhancing LiDAR Performance Through Hybrid Sensor Solution

As we work to improve the localization capabilities of our robotic systems, we have to tackle the inherent limits of LiDAR technology, specifically its interaction with transparent surfaces and susceptibility to interference from intense light. LiDAR sensors work by producing laser beams that reflect off surfaces to determine distance.

Transparent surfaces, like glass, present a distinct problem. Glass can either reflect, absorb, or allow laser beams to pass through, depending on the angle of incidence, laser wavelength, and glass type. This inconsistency frequently leads to unreliable data from glass surfaces since the LiDAR cannot reliably identify a reflection. In an area with many glass surfaces, such as a shopping mall, this can result in inaccurate mapping and localization.

Another issue is interference produced by intense ambient lighting. LiDAR systems detect objects using light, and when the environment is filled with light—whether direct sunshine or reflections from bright surfaces—it can "drown out" the LiDAR's laser pulses. This is similar to attempting to listen to a calm conversation in an extremely

noisy environment. The sensor may struggle to distinguish between its own laser light and ambient light, resulting in inaccurate measurements or missing data points.

To address these constraints, we will do research on hybrid sensor systems that integrate LiDAR with other types of sensors that are resistant to such interferences. By combining LiDAR, RADAR, and ultrasonic sensors, we can compensate for each sensor's shortcomings with the strengths of others. RADAR, which employs radio waves, is unaffected by transparent surfaces and can identify things in a wide range of illumination situations, including strong light. Ultrasonic sensors, which use sound waves, can identify things regardless of illumination conditions and are especially effective in close-range detection circumstances.

The integration will include a smart fallback system in which, if LiDAR data is determined to be inconsistent or inaccurate, the system can temporarily rely on data from RADAR and ultrasonic sensors. Sensor fusion algorithms will be developed to produce a cohesive dataset that the robot may use to accurately understand its surroundings.

7.7.3. Live Map Forming

As discussed in Section 7.2.2, a live map would be much more dynamic as opposed to having a preset map. In the preloaded map, the robot must start in a unique section so the error metric would give a very low error. If the robot starts in a non-unique position the error metric would be very close and the minima could show the wrong value, this will then give the wrong point of interest and the actual robot would move incorrectly. It would not go into the wall due to the object avoidance subsystem but would still move incorrectly. If a live map can be formed by the actual robot, not one where the robot interacts with to get a localisation position, this could be very useful for achieving a very dynamic and the use for error metric would be redundant, a focus on odometry and orientation would be more central if real-time map forming.

8. Sound Localization

8.1. Introduction

8.1.1. Motivation

One of the well-known positioning system used worldwide is the Global Positioning System (GPS). This system uses satellite-based navigation, however, its use case can only be applied to outdoor localization as it requires line-of-sight (LOS) between the satellites and users [22]. Indoor localization is an emerging area of study that requires a reliable and accurate positioning system indoors in order to perform various tasks.

Sound-based localization is one of the possible solutions that can be adopted to enhance the precision beyond what GPS can provide. By utilizing acoustics, we can achieve finer-grained measurements that meet the requirements of various indoor positioning applications such as indoor emergency response systems, warehouse management systems, and navigation in large buildings such as malls, hospitals, and airports. Furthermore, the use of acoustics offers practical advantages in implementing precise time-synchronization mechanisms. Unlike light-based systems that rely on the speed of light for time measurements, sound travels at a significantly slower speed, which reduces the sensitivity of the position estimation algorithm to microsecond errors during synchronization.

The overarching goal of developing a subsystem for indoor localization using sound is to enable an object or person to determine its own position based on time-difference-of-arrival (TDoA) measurements from known positions of sources of sound. As proof of concept, a microphone can know its location in a small area by triangulating the received sound from three speakers placed in predetermined spots. By utilizing the robust property of linear chirps and leveraging TDoA and cross-correlation techniques, this subsystem aims to provide an alternative localization solution tailored specifically to operate in a grocery store.

8.1.2. Exploratory Phase

Before the final design choice was made for this subsystem, various approaches were explored to address the goal of developing a precise indoor positioning solution.

Bluetooth modules can be used to perform indoor localization. By using Received Signal Strength Indicator (RSSI) values, the receiving device can estimate its approximate position by comparing these values to the known signal strengths of the beacon's places at specific locations. This method would have been an optimal solution for indoor localization. However, its readings can often present significant noise and susceptibility to interference. Bluetooth signals operate at 2.4 GHz frequency, which is the same frequency as Wi-Fi networks, thereby increasing the likelihood of interference.

Visual markers are another choice for localization. Visual markers, such as QR codes or barcodes, can be placed on the floor or ground to enable the robot to determine its position within the area. However, this approach would present potential complexities for the project as it would require the integration of machine learning models to interpret the captured images of the markers. Additionally, it could potentially constrain the robot's speed as it would need to maintain a specific speed threshold to ensure precise and vivid photo capture.

Using electromagnetic (EM) waves for localization posed challenges due to their fast propagation, making them highly sensitive to synchronization errors. Consequently, utilizing sound for indoor localization became a better approach due to its slower propagation speed resulting to a more precise time synchronization. However, achieving this precise synchronization required ensuring simultaneous emission of sound from the transmitters. Therefore, a challenge would be to find a reliable transmitter (speaker) synchronization method. Packet transmission via UDP sockets over Wi-Fi using a laptop setup was initially tested for this. While Wi-Fi networks typically offer high data transfer rates and minimal latency, environments with heavy network traffic may lead to increased latency and packet loss. Additionally, it was difficult to control the latency caused by the operating systems of the laptops when trying to emit sound from speakers or record sound from the microphones. As an alternative, transmission

of UDP packets via Ethernet cables was explored which provided an improvement in consistent latencies. Even with this approach, recording and emitting the sound at consistent delay still posed problems due to inconsistent delays in acquiring sound devices from the host operating system.

Raspberry Pi Pico microcontrollers, specifically utilizing the RP2040 chipset, were used as a transition from the laptop setup. These microcontrollers allowed us to avoid the causes of inconsistent latencies from the previous setups. The previous approach was thus replicated with one microcontroller interfaced with a speaker and the other with a microphone. The figure below demonstrates this setup:

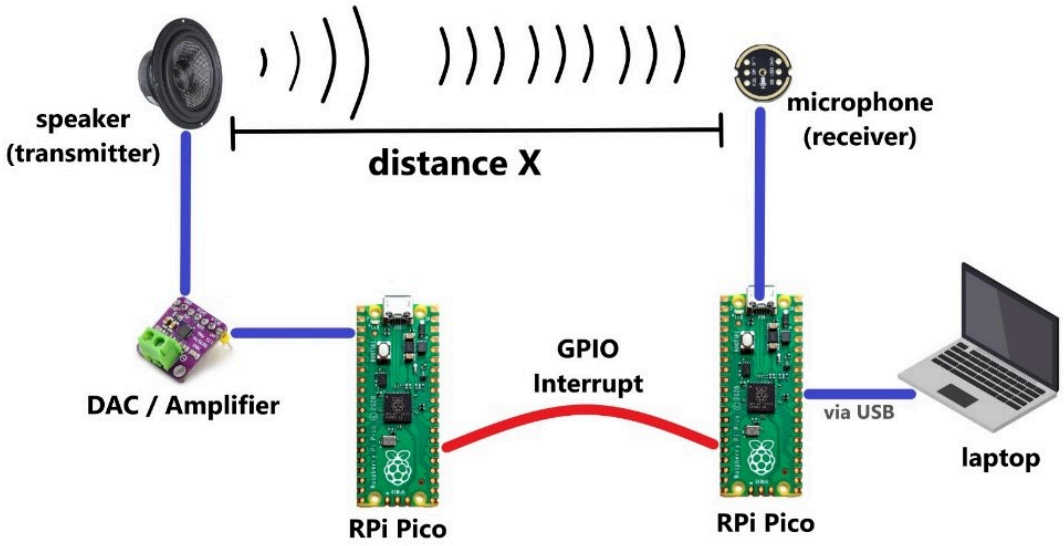

Figure 42: Setup for distance measurer using 1 microphone and 1 speaker

To initiate the recording process on the receiver side, a signal was transmitted from the transmitter's GPIO pin to trigger an interrupt in the receiver's program. Since Direct Memory Access (DMA) and buffering techniques were used in both the sound input and output, special care had to be done to ensure that when the interrupt was sent and received, there was a consistent delay before the sound samples were sent or recorded. This eliminated the issue of inconsistent latencies, but this wired communication approach limited the deployment opportunities. Eventually, sub-GHz trans-

ceivers (RFM69HCW) were employed instead of the GPIO interrupts. Later sections further discuss how this approach was implemented.

8.2. Design

8.2.1. Correlation

The first step in locating something via sound waves or other forms of propagating waves is to be able detect when they are received. The rationale behind this is that if one knows both when a signal was received and when it was transmitted, the difference between these two gives the total time the signal travelled through the air. Multiplying it by the speed of the medium, such as the speed of sound or speed of light, will get the distance the receiver must be from the transmitter. This is the Time of Arrival concept and is explained in further depth in Section 8.2.2.1.

There are many ways to do this. The simplest is to assume the signal transmitted is of much higher power or volume than the other signals in the environment. One could detect an increase in average power above a certain threshold and declare that the sound is received. This is shown in the figure below:

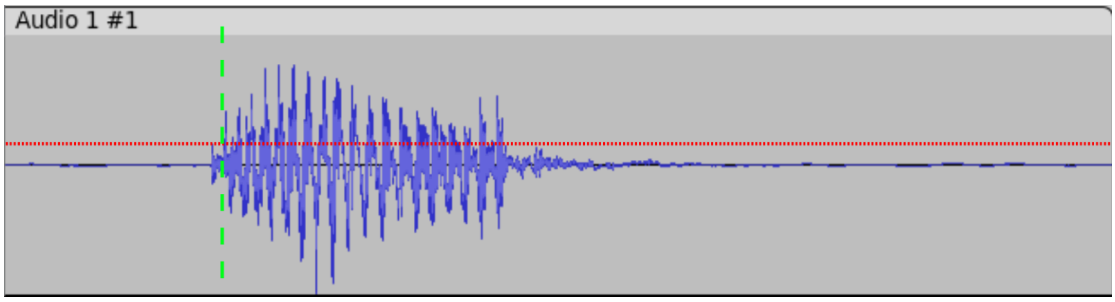

Figure 43: An example sound signal. The red dotted line shows the volume threshold, and the green dashed line the time at which the signal passed the threshold.

A better approach is to use a matched filter, also known as a correlation. Matched filters can be shown to be the optimal linear filters for maximizing Signal to Noise Ratio (SNR) [23, section 4.2.1]. One of the requirements to use this approach is that the receiver must know the transmitted signal ahead of time. The receiver, while listening continuously, can measure every time a new sound sample is recorded how much the last N number of samples received resembles the expected signal. This can be understood as a sort of convolution, where instead of the receiver checking the resemblance

every time a new sound sample is recorded, a batch of a certain length of samples are processed at a time. They are convoluted with the expected sound flipped in the time domain. The equation would thus be:

$$\text{Correlation}(t) = \overline{f(-t)} * g(t) \tag{25}$$

The flip in the time domain is necessary, as a convolution usually flips one of the signals in the time domain before multiplying it with the second signal at every point in time. By doing a flip before the convolution, this flip is prevented from occurring, making both signals multiply against each other whilst both being in their original un-time-domain-flipped states. The flipped signal must also be complex conjugated, but in the case of real numbers such as sound, this has no effect.

When the length of both the signal and of the recorded sample are large, this equation can be calculated more efficiently in the frequency domain. The equation for this is given below:

$$\text{Correlation}(t) = \mathbb{F}^{-1}\left(\overline{\mathbb{F}(f(t))}\mathbb{F}(g(t))\right) \tag{26}$$

Finding the sample at which this function is at its peak will give the moment in time at which the signals most resembled one another.

This was implemented in Python using the following code:

```
import numpy as np

def correlate_and_find_delay(rec, noise):
    rec_fft = np.fft.rfft(rec)

    noise_freq_conj = np.conj(np.fft.rfft(noise))

    cross_corr_freq = noise_freq_conj * rec_fft
    cross_corr = np.abs(np.fft.irfft(cross_corr_freq))

    k_max_ind = np.argmax(cross_corr)
    k_max = cross_corr[k_max_ind]

    return k_max_ind, k_max
```

This was performed in the discrete domain. One of the assumptions of the Discrete Fourier Transform (DFT) is that the signal is continuously repeated. Because of this, special care must be taken when using the DFT of a signal. In the case of convolutions or correlations, the output will be that of a circular convolution, where the first samples are used as part of the correlation for the end samples. This is demonstrated in the next few figures below.

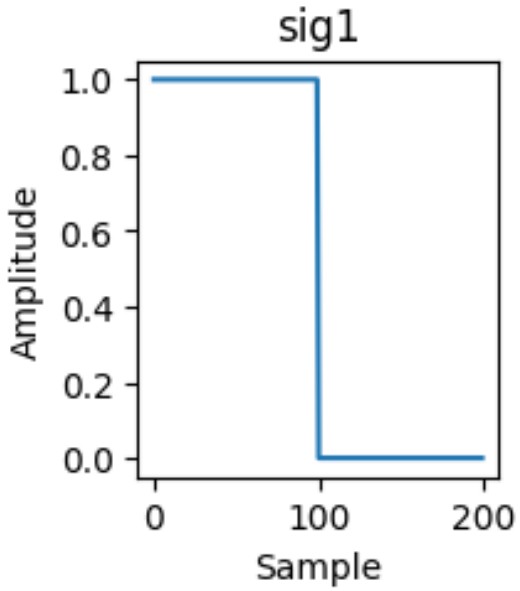

Figure 44: One period of sig1: a square wave signal.

The figure above shows a signal which is equal to 1 for the first half, then zero for the second. It is expected that the autocorrelation (i.e. the signal correlated with itself) would show a maximum value at the first sample with index 0 when the signals are perfectly overlapped, then would partially correlate until sample 100. This is demonstrated in the following figure:

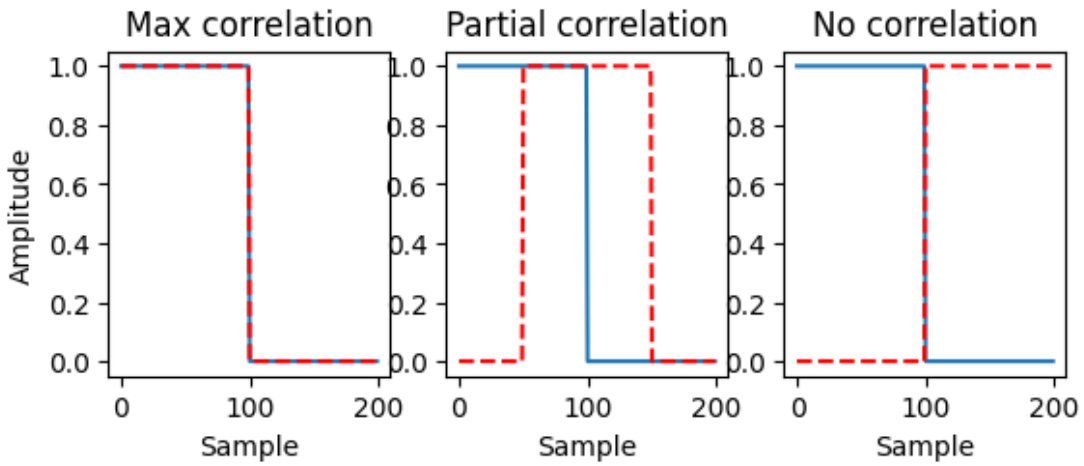

Figure 45: Demonstration of how much sig1 overlaps with itself at various points in time, to explain the result of the autocorrelation.

It would be expected for the correlation to be 0 for all samples after 100 as well, but as seen in the next figure, this is not the case.

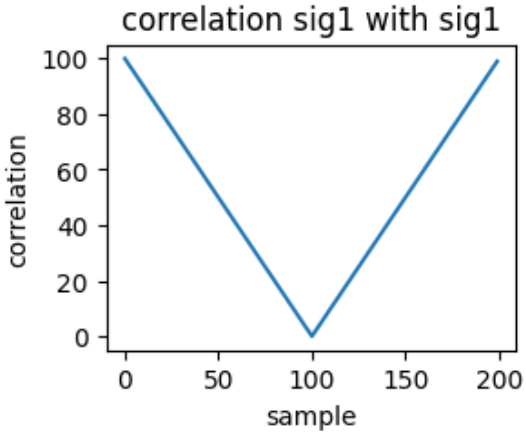

Figure 46: The autocorrelation of sig1.

This is due to the convolution wrapping around, as show below.

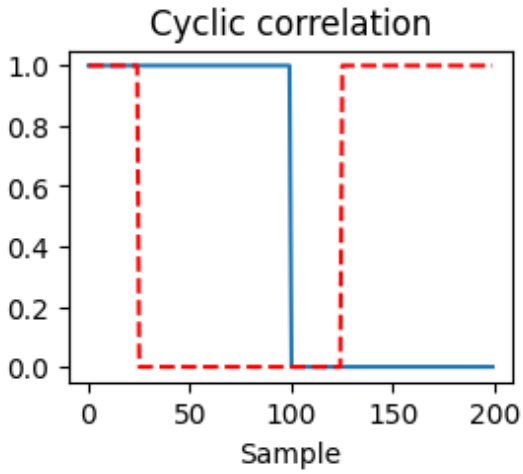

Figure 47: Demonstration of a cyclic correlation by overlapping sig1 with another shifted and wrapped around version of sig1.

The correlation thus steadily rises as more and more of the 1's wrap around to the start and get multiplied by 1's in the other sig1.

In applications where the signals are not cyclical and this assumption does not uphold, once could for example only use the first 100 samples of the correlation, before the correlation has to opportunity to wrap around. To aid in this, signals can be zero-padded.

In the case of using correlations for localization using the Time Difference of Arrival (TDoA) technique, explained in detail in Section 8.2.2.2, the assumption of the cyclical sounds can be fulfilled by having the transmitters transmit their signal a loop continuously, and by doing a correlation on the captured signal from the receiver of the same length in time as the transmitter's signal. This allows the use of the full output of the correlation. Peaks in the correlation towards the end can be interpreted as the signal being received before the recording started, where the end of the recording contains the start of the transmitted signal, and the start of the recording contains the end of the signal. An example demonstrating this using spectrograms this is shown below, using the chirps described in Section 8.2.2.1.

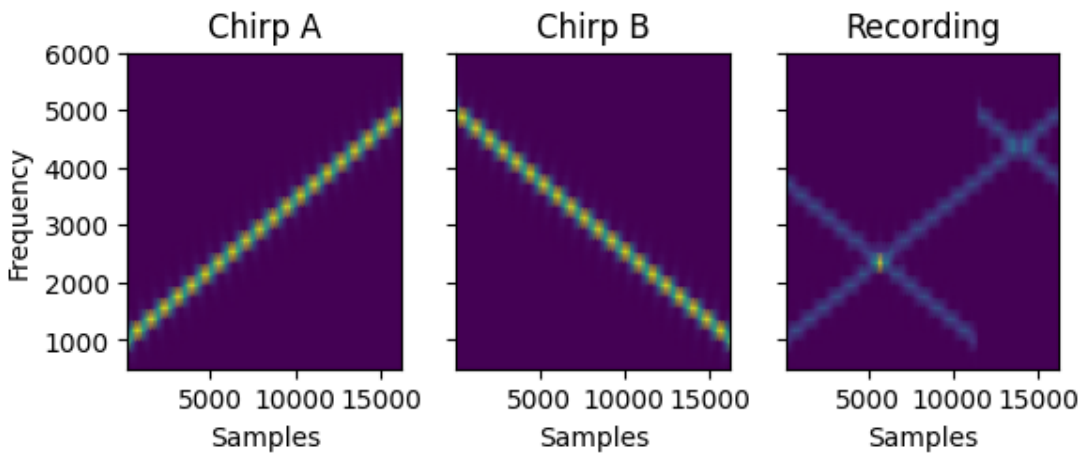

Figure 48: Spectrogram of two linear chirps, and a recording of these chirps repeating in time, limited to the same amount of samples as the chirps.

Chirp A can be found starting at sample 0. Chirp B however, its end is seen at the beginning of the recording, and its start later in the recording, near sample 12500.

The figure below shows the same recording, but represented by correlations instead of spectrograms. The peaks demonstrate the position of the start of the chirps.

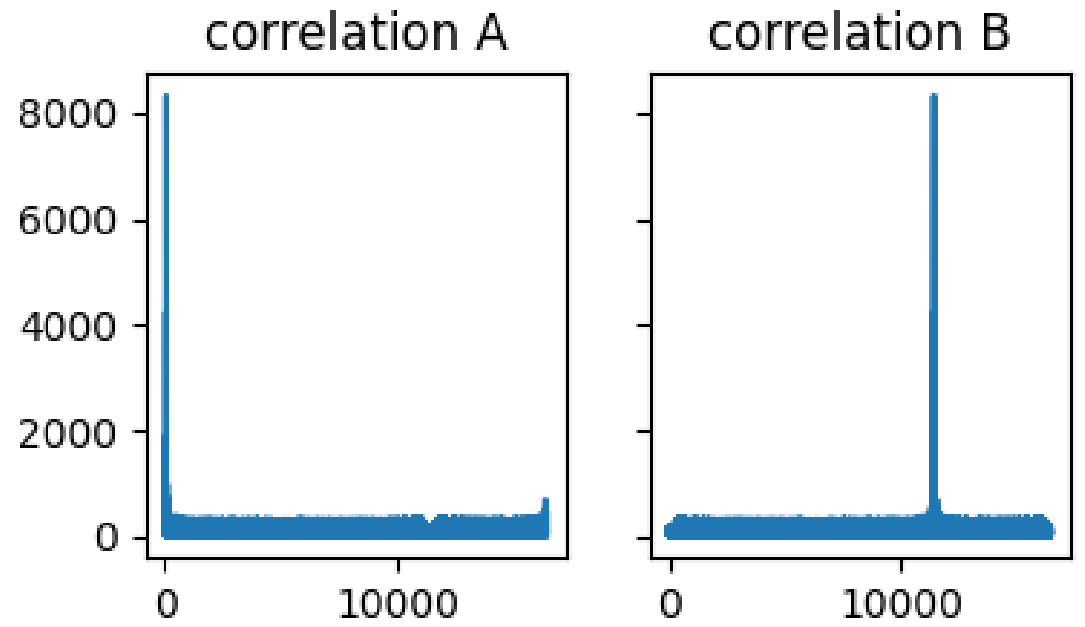

Figure 49: The correlation of the recording with respect to chirps A and B.

8.2.2. Localization Algorithms

Once the signals have been located in time using something like a matched filter, algorithms such as Time of Arrival (ToA) and Time Difference of Arrival (TDoA) can be

used to find the position of an object of interest. Both algorithms require the position of the reference entities to be known in advance. The figure below shows the setup used during our demo, explained in further detail in Section 8.4, the 'reference entities' in this case being the speakers.

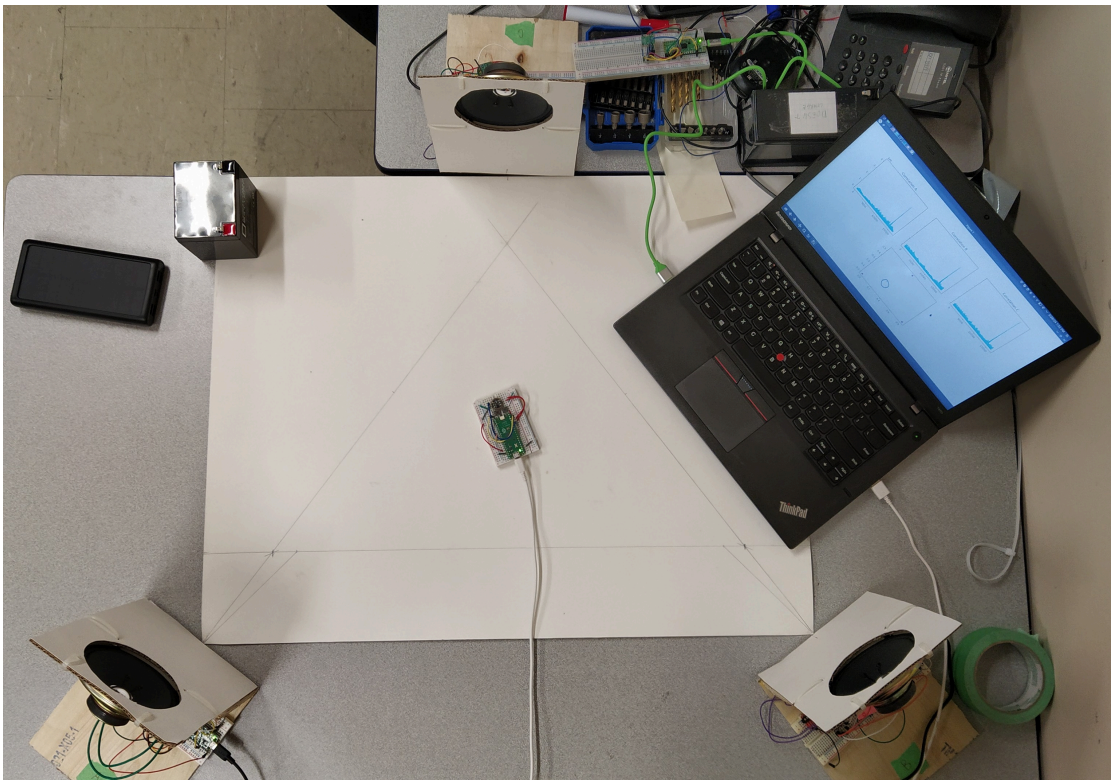

Figure 50: Image of the setup used during the demo.

These would be translated to numerical coordinates. As shown in the plot below:

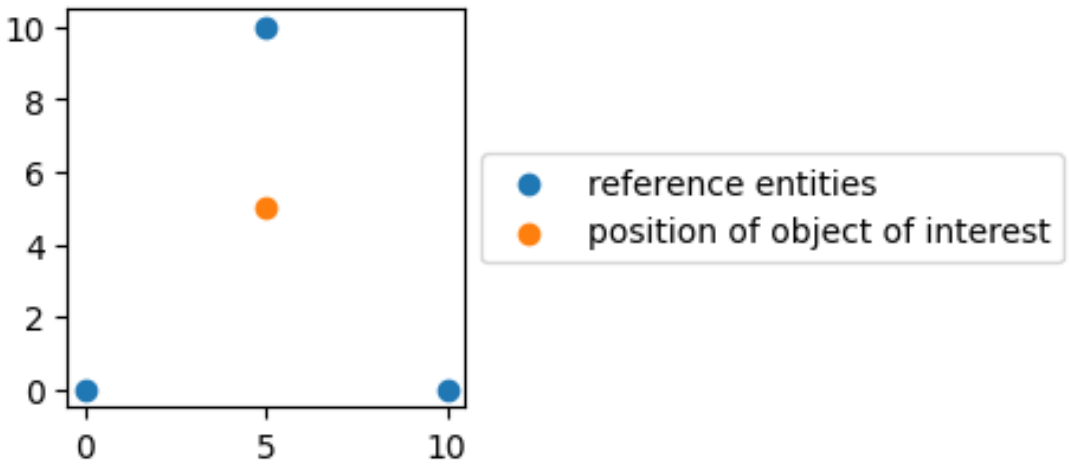

Figure 51: A plot representation of the demo setup.

In this case, speaker ‘A’ would be at position (0,0), speaker ‘B’ at position (10,0), and speaker ‘C’ position (5, 10).

The other input to the algorithms is the correlation time, explained in the previous section. If the reference entities are transmitters and the object of interest a receiver, then the object of interest would capture signals for a certain length of time, then find the point in time when the recording most resembles the signal transmitted by each speaker. Each transmitter would need to transmit a different and orthogonal signal.

The other way around is also possible, where the object of interests are the transmitters, and the points of interest are the receivers. In this case the points of interests would be the ones capturing signals and doing the correlations.

8.2.2.1. Time of Arrival (ToA)

With Time of Arrival, the time at which the signal was transmitted is known. When this is the case, the difference in time between when the signal was heard and when it was transmitted can be taken. This will give the time of flight of the signals, denoted here by T_A, T_B, and T_C. where T_s is the time at which the signals started playing.

$$\begin{aligned} T_A &= T_{\text{heardA}} - T_s \\ T_B &= T_{\text{heardB}} - T_s \\ T_C &= T_{\text{heardC}} - T_s \end{aligned} \tag{27}$$

These values can be multiplied by the speed of the medium (denoted here by c and given the value of $343\frac{m}{s}$ for the speed of sound), which will give the distance at which the receiver must have been from the transmitter.

$$\begin{aligned} R_A &= T_A \cdot c \\ R_B &= T_B \cdot c \\ R_C &= T_C \cdot c \end{aligned} \tag{28}$$

Since the location of the reference entity is fixed (ie. the location of speakers A, B, and C), then the position of the object of interest must be somewhere in a circle around the reference entity, at the distance calculated above.

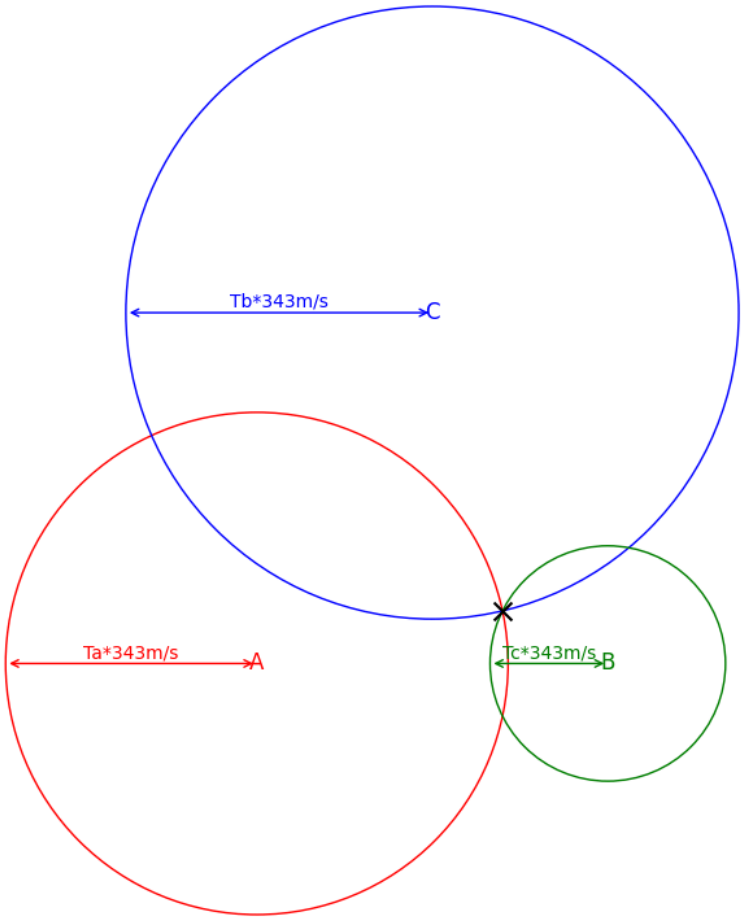

Figure 52: Three circles showing the distance at which the object of interest must be relative to the reference entities

The position of the object of interest must be the intersection of these three circles. This can be defined by the equations below:

$$R_A = \sqrt{(A_x - x)^2 + (A_y - y)^2}$$
$$R_B = \sqrt{(B_x - x)^2 + (B_y - y)^2} \tag{29}$$
$$R_C = \sqrt{(C_x - x)^2 + (C_y - y)^2}$$

One can solve for x and y using numerical or other means.

8.2.2.2. Time Difference of Arrival (TDoA)

In Time Difference of Arrival, the time at which the signals were transmitted is not known. By taking one of the correlation times as reference for a difference though, this start time can be mathematically cancelled out.

$$\begin{aligned} t_A &= T_{\text{heardA}} = \frac{R_A}{c} + T_s \\ t_B &= T_{\text{heardB}} = \frac{R_B}{c} + T_s \\ t_C &= T_{\text{heardC}} = \frac{R_C}{c} + T_s \end{aligned} \tag{30}$$

$$\begin{aligned} t_B - t_A &= \frac{R_B}{c} + T_s - \left(\frac{R_A}{c} + T_s\right) \\ t_C - t_A &= \frac{R_C}{c} + T_s - \left(\frac{R_A}{c} + T_s\right) \end{aligned} \tag{31}$$

In Equation 31, it can be seen that T_s will cancel out for both $t_B - t_A$ and $t_C - t_A$.

When multiplied by the speed of the medium again, we can substitute the equations used for circles for R_A, R_B, and R_C, such as in the ToA section, and get:

$$\begin{aligned} c \cdot (t_B - t_A) &= R_B - R_A \\ c \cdot (t_C - t_A) &= R_C - R_A \end{aligned} \tag{32}$$

$$\begin{aligned} c \cdot (t_B - t_A) &= \sqrt{(B_x - x)^2 + (B_y - y)^2} - \sqrt{(A_x - x)^2 + (A_y - y)^2} \\ c \cdot (t_C - t_A) &= \sqrt{(C_x - x)^2 + (C_y - y)^2} - \sqrt{(A_x - x)^2 + (A_y - y)^2} \end{aligned} \tag{33}$$

Instead of three circles like in the ToA case, here we get two hyperbolic curves, their intersection being the position of the object of interest.

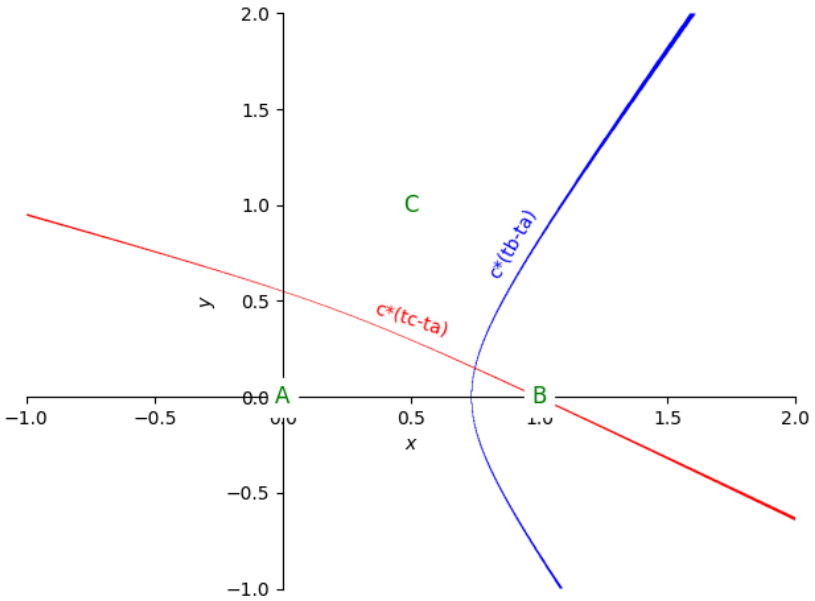

Figure 53: Intersection of two hyperbolic curves given by the TDoA equations, their intersection showing the position of the object of interest.

x and y can be solved for numerically or algebraically. In this project, we used an algebraic solution described by Bertrand T. Fang [24], in the case where the z plane of the object of interest is known ahead of time.

With three reference positions, unlike in the ToA case, TDoA can have two solutions. Most of the time but depending on the layout of the reference entities, this extra solution is outside the zone of interest, and can be ignored.

TDoA also has some zones which are more accurate than others. The figure below illustrates this, which was taken from [25].

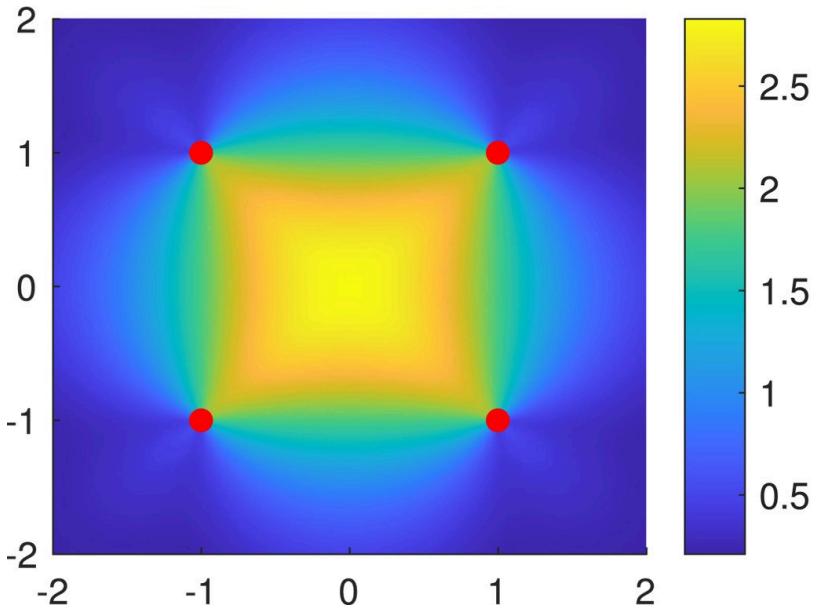

Figure 54: The Dilution of Precision (DOP) with TDoA in the case of 4 reference entities.

8.2.3. Signals Used

To an extent, any signal can be used between the transmitters and the receivers. An optimal signal would be one that both is resistant to noise, where the correlation would still show a clear peak in the presence of loud background noise of all types, and where the correlation would still show a strong peak in the presence of the Doppler effect, where either the transmitter or the receiver is in motion.

One type of signal that fulfills this requirement is Gaussian noise. Theoretically, the autocorrelation of gaussian noise should be 0 at all points in time other for the peak. In practice, this gaussian noise is band-limited by both the transmitted and the receiver, and so does not have this perfect theoretical property.

8.2.3.1. Gaussian Noise

Gaussian noise was used in our project to some success. We were successfully able to implement localization with it in a demo. Some challenges were encountered though, such as being unable to maximize or at least equalize the power output between the sounds at each transmitter. This is due to the signal having values greater than 1.0 or smaller than −1.0 after being put through a band pass filter, requiring a normalization afterwards. The performance of this type of signal with movement was also found to be very poor. When the receivers and transmitters were perfectly still, the correlations

would peak at relatively high values, but with the slightest movement the correlation peaks would decrease by orders of magnitude.

8.2.3.2. Linear Chirps

Linear chirps were used in the final version of the demo. Linear chirps vary in frequency linearly over time, demonstrated by the spectrograms shown below.

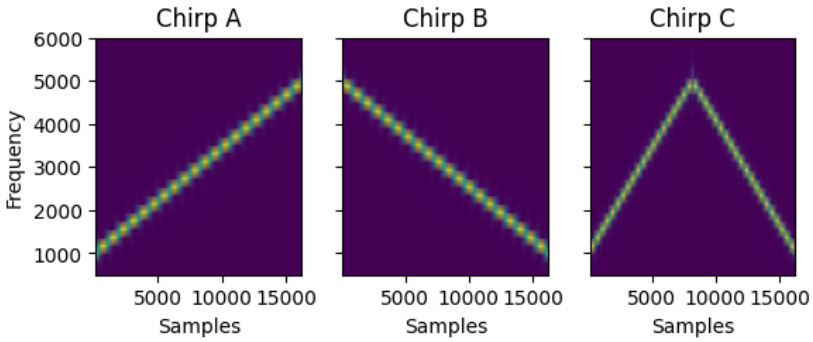

Figure 55: The three linear chirps used in this project

Chirp A being an 'up chirp', chirp B being a 'down chirp' and chirp C varying twice as fast being an 'up down chirp'.

These chirps are very band-limited and have fairly consistent power over the defined range of frequencies, as shown by the plot of their Fourier Transforms below:

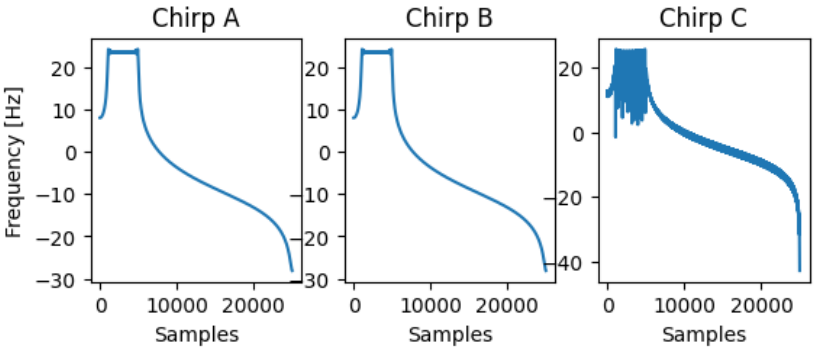

Figure 56: Fourier transform of chirps A, B, and C.

The starting and ending frequencies were chosen to be 1000 Hz and 5000 Hz. These frequencies are almost guaranteed to be received by microphones designed to capture human voice, such as phone and laptop microphones, or the INMP441 microphone used in this project.

A chirp can be defined as:

$$x(t) = \cos\left(\pi\left(t\frac{\beta}{\mathrm{T}}\right)t\right) \tag{34}$$

Where β is the slope of the chirp, which would be positive for an up chirp and negative for a down chirp, and T is the duration of the chirp.

Chirps can be characterized by the product βT. The higher this product, the sharper the autocorrelation peak, as shown in the figure below from [23]:

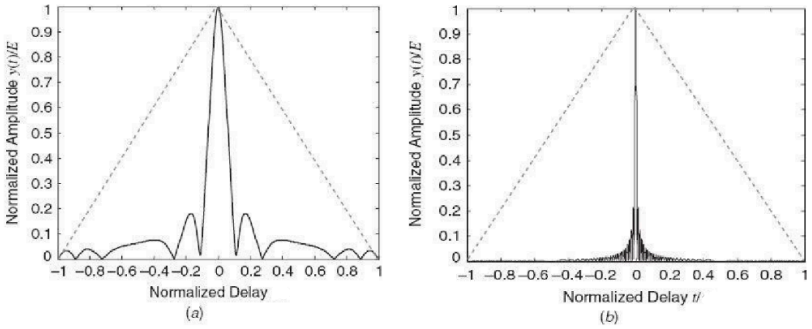

Figure 57: Figures showing the effect of βT on the autocorrelation of a chirp. (a) has a βT =10, and (b) βT=100.

The chirps A, B, and C used in this project have $\beta\mathrm{T} = 4000$, with their autocorrelations shown below:

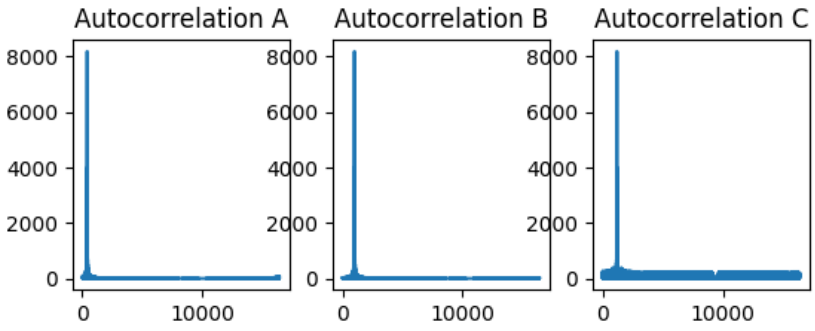

Figure 58: Autocorrelation of chirps A, B, and C used in this project.

Increasing both β, which is related to the size of the frequency band swept by the chirp, and T, the length of the chirp, will narrow the peak of the autocorrelation. Having a narrower peak will reduce the chance of being off by a few samples when finding the max correlation.

Increasing T will also increase the total amount of power transmitted by the chirp, which will raise the SNR and consequently the value of the autocorrelation peak.

It can also be shown that when either the receiver or the transmitter is in motion, the correlation will not peak at the correct moment, it will have a certain error. This error is defined in [23, section 4.6.4] as:

$$E_t = -\frac{\mathrm{T}F_D}{\beta} \tag{35}$$

[23, section 4.6.4] also says that the peak value of the correlation will be reduced by a factor of:

$$A_{\text{reduction}} = 1 - \frac{F_D}{\beta} \tag{36}$$

Where F_D is defined as follows in [23, section 2.6.1]:

$$\begin{aligned} \beta_v &= \frac{v}{c} \\ F_D &= \frac{2v}{(1-\beta_v)\lambda} \end{aligned} \tag{37}$$

For this project, the objects of interest such as people and automated carts are likely to travel at less than 5 m/s. At this speed the error and peak reduction (and therefore SNR) factor would be:

$$\begin{gathered} v = 5\frac{m}{s} \quad c = 343\frac{m}{s} \\ f_{\text{MAX}} = 5000 \text{ Hz} \quad f_{\text{MIN}} = 1000 \text{ Hz} \\ \beta = 12207 s^{-2} \\ T = \frac{\text{number of samples}}{\text{sampling rate}} = \frac{2^{14} \text{ samples}}{50 \text{ kHz}} = 0.32768 s \\ f_0 = \frac{f_{\text{MAX}} - f_{\text{MIN}}}{2} = \frac{5000 - 1000}{2} = 2000 \text{ Hz} \end{gathered} \tag{38}$$

$$\lambda = \frac{c}{f_0} = \frac{343}{2000} = 0.1715m$$
$$\beta_v = \frac{v}{c} = \frac{5}{343} \tag{39}$$

$$\begin{aligned} F_D &= \frac{2v}{(1-\beta_v)\lambda} \\ &= \frac{2 \cdot 5\frac{m}{s}}{\left(1-\frac{5}{343}\right) * 0.1715m} \\ &\approx 60 \text{ Hz} \end{aligned} \tag{40}$$

$$\begin{aligned} E_t &= -\frac{TF_D}{\beta} \\ &= -\frac{0.32768s \cdot 60 \text{ Hz}}{12207s^{-2}} \\ &= 1.6106 \text{ ms} \end{aligned} \tag{41}$$

$$\begin{aligned} A_{\text{reduction}} &= 1 - \frac{F_D}{\beta} \\ &= 1 - \frac{60\text{Hz}}{12207} \\ &= 0.9951 \end{aligned} \tag{42}$$

These chirps seem suitable for this application. The amplitude reduction is excellent, but the 1.61 millisecond time shift could cause issues, especially in the case where the object of interest is moving towards one speaker, and away from another. This time shift would be opposite for both speakers.

A solution to this issue described in [23, section 4.6.4] would be transmitting both an up and down chirp and averaging the results of both at the receiver. This will cause the time difference Et to cancel out.

Increasing beta by increasing the frequency range swept by the chirps would also help, as would decreasing the duration of the chirps.

8.2.3.3. Sampling Rate and Accuracy

The sampling rate is a limiting factor to the resolution at which an object can be localized. Assuming the autocorrelation of the signal has a peak at exactly 1 sample, this is made evident in the case of ToA. The correlation will only be at a maximum at a certain integer number, the sample index, and in ToA this is directly related to the distance between the reference entities and the object of interest. Increasing the sampling rate increases the number of samples in the signal, which will increase the amount of samples in the correlation. Considering a one transmitter, one receiver case, the worst-case scenario would be when the signal is heard at the receiver exactly between two samples.

$$
\begin{aligned}
F_s &= 50 \text{ kHz} \\
E_t &= \frac{1}{50 \text{ kHz}} \cdot \frac{1}{2} = 10\mu s \\
c \cdot E_t &= 343\frac{m}{s} \cdot 10\mu s = 3.53mm
\end{aligned}
\tag{43}
$$

If the sampling frequency is 50 kHz and the medium is acoustics, this would induce a maximum error of 3.53 mm.

The resolution at which the position is measured, as in the difference from when the correlation peak is at one sample vs the sample directly after, would be double this maximum error, around 7 mm.

8.3. Equipment Used

8.3.1. Raspberry Pi (RP) Pico (RP2040)

The microcontroller used for sound localization was the RP Pico. This microcontroller includes features such as a dual ARM Cortex-M0+, 30 GPIO pins, and various peripherals including UARTs, SPI controllers, I2C controllers, and eight programmable input/output (PIO) system machines [26]. Although the microcontroller does not have Inter-IC Sound (I2S) built in, the PIO pins can be used to implement it. As shown in the deployment diagram (Section 8.4.2), five of these microcontrollers were used for sound

localization: one for time synching the speakers, three for the speakers themselves, and one for the microphone. For the speakers, the Pico was connected to a digital-to-analog (DAC) converter and the RFM69HCW wireless transceiver (discussed in further detail in the next section). For the master controller, the Pico was connected to a singular RFM69HCW wireless transceiver. Lastly, the microphone was connected to a Pico directly.

8.3.2. RFM69HCW Wireless Transceiver

The RFM69HCW transceiver module was used in the capstone. It is a sub-1 GHz module which operates in the ISM license-free band. The transceiver features FSK, GFSK, MSK, GMSK and OOK modulations along with programmable narrow-band and wide-band communication modes [27]. The RFM69HCW was used to perform communication between a controller Pico and three worker Picos (for the speakers). Specifically, the controller Pico used its transceiver to broadcast a message to the transceivers connected to the worker Picos. This allowed for the proper time synchronization between the speakers, which ultimately enabled precise measurements using sound localization.

8.3.3. Arduino-Pico Core

The Arduino-Pico core allowed for interfacing the Pico microcontroller with the Arduino IDE. The Arduino programming language was used to create various functions to provide functionality for controlling the master microcontroller and the speakers. Although the sound localization was successful, there were a few issues encountered when using the Arduino-Pico core. For instance, the current functionality requires that the serial be written and read from. However, it was found that sometimes the serial could not be read after receiving messages from the RFM69HCW. In addition, during the demonstration at the poster fair, it was discovered that the speakers began to desynchronize after a relatively short duration. This issue required to consistently re-synchronize the speakers by broadcasting a message from the controller transceiver module to the worker transceiver modules via the serial monitor in the Arduino IDE.

8.3.4. Speakers, Microphone, and DAC Converter

The speakers used for sound localization were purchased from Addison Électronique. They were attached to a wooden stand for ease of use and testing, as demonstrated in Figure 59. For the microphone, an NMP441 omnidirectional module was used. This module has an I2S interface, which allows for direct analog-to-digital (ADC) conversion to take place without the need of an external module. In addition, a DAC converter was used to convert the digital linear Chirp sound samples into an analog sound signal played by the speakers. The precision and accuracy of sound localization was ultimately limited by the sampling frequency and lowpass filters on the NMP441 microphone and DAC converter.

8.4. Proof of Concept

This section described the simplified design concept of sound localization in this project. The actual setup used to demonstrate this design was elaborated in the Setup section. The Deployment diagram section presented a visual representation of the flow of data and signals between the components. Finally, the circuit diagram showed the electrical connections of the components. Additionally, the code used for the setup can be found in Appendix or Source Code file in the GitHub repository.

8.4.1. Setup (Demo)

As seen in Figure 59 below, the setup comprised three speakers positioned within a 22in x 28in unobstructed workspace, each connected to a Raspberry Pi Pico (RPi Pico) equipped with a DAC and an RMF69HCW transceiver (See Section 8.4.3 to see schematics). These RPi Picos were programmed to emit distinct sound sequences. These Picos were plugged into the wall during testing. Additionally, a microphone, controlled by another RPi Pico connected to a laptop, captured the emitted sounds. The laptop ran a Python program to receive and process these sound bytes, generating correlation plots to monitor the microphone's sensitivity to each speaker and an axis map to determine the position of the target object.

Figure 60 below displays correlation plots along with an axis map. In the axis map, measured in meters, the three speakers are depicted as base stations (purple dots), and the microphone was represented as the target object (light blue dot). As proof of concept, the map is interactive, allowing users to draw on it. By pressing a designated button, users can initiate drawing, and another button clears the drawing. Inside the axis map in Figure 60, it is visible that a circle is drawn by the user.

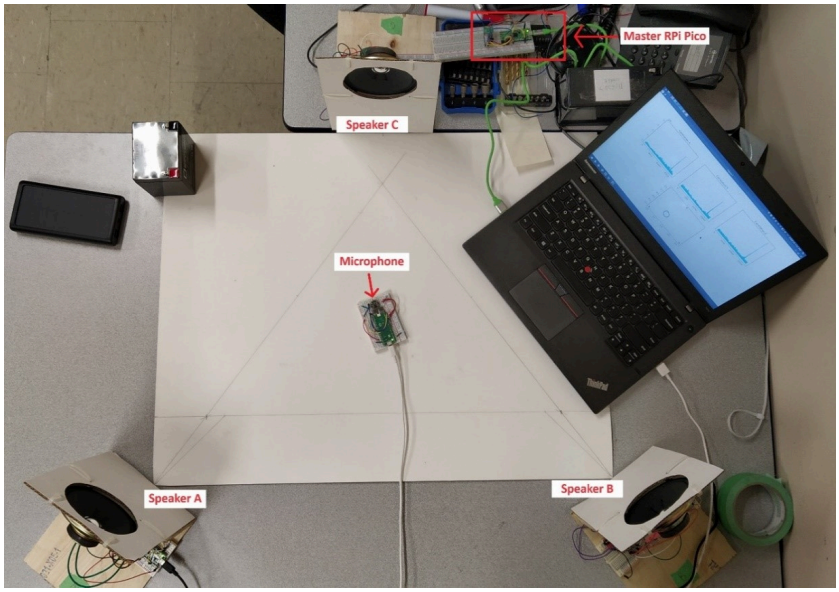

Figure 59: Setup of the sound localization demo

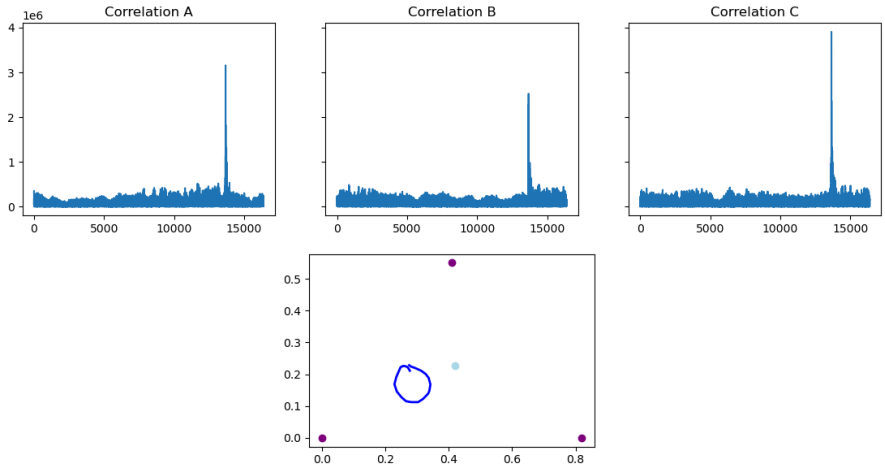

Figure 60: Interface showing the sound correlation of each speaker and the map of the testing area (bottom)

8.4.2. Deployment Diagram of the Setup

Figure 61 below illustrates the simplified design concept for sound localization. A master RPi Pico with a RFM69HCW transceiver broadcasted a signal to the speaker RPi Picos, triggering the emission of unique linear chirps. Simultaneously, the microphone RPi Pico captured this audio data, converted it into sound bytes and transmitted them to the laptop via seral communication. The local server, in this case the laptop, processed this audio data to calculate the position of the target. Section 8.2 above elaborated on the techniques used for audio processing and object localization.

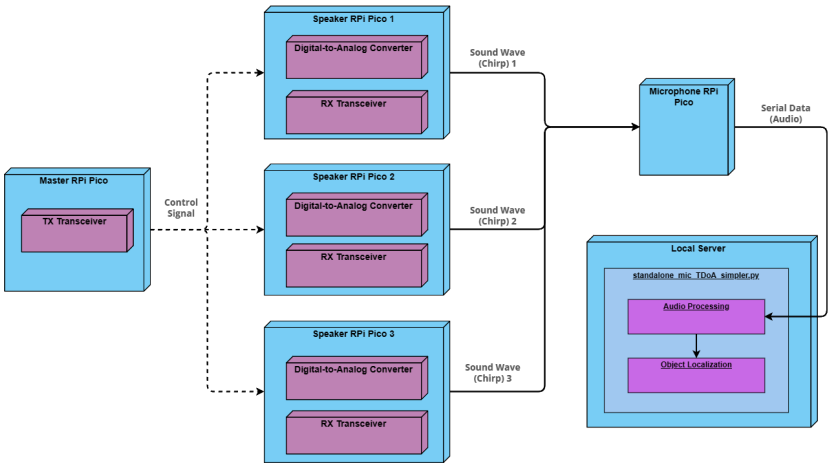

Figure 61: Deployment diagram of simplified design concept of sound localization

8.4.3. Circuit Diagrams of the Setup

The following figures below illustrate the electrical connections and components within a circuit. Figure 62 depicts the setup for the three speakers. In this setup, the RPi Pico GPIO pins 20, 21, and 22 are connected via I2S communication to the DAC's Bit Clock (BCLK), Left/Right Clock (LRCLK), and Data In (DIN). Additionally, the RPi Pico is connected to the RFM69HCW transceiver via SPI communication, with GPIO pins 16, 17, 18, 19, and 2 linked to the MISO, NSS, SCK, MOSI, and DIO0 pins of the transceiver, respectively. It is worth noting that the RFM69HCW is connected to an antenna via its ANA pin.

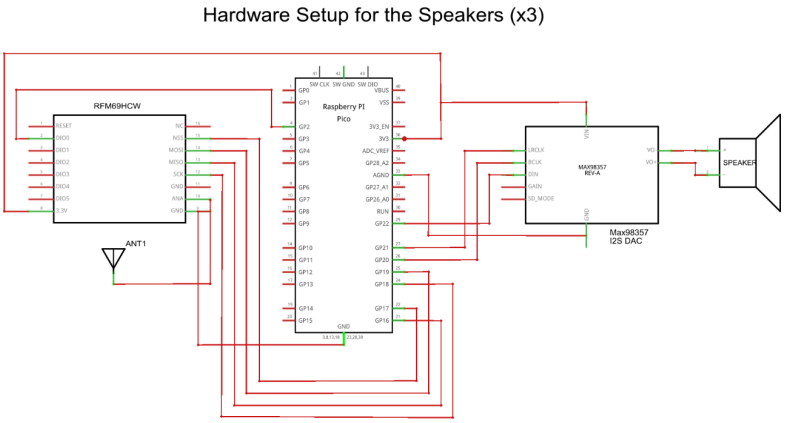

Figure 62: Circuit diagram of the transmitter speakers with DAC and RPI Pico

Figure 63 is the same as the above figure without the DAC and speaker connections. This circuit connection is used for the master RPi Pico that broadcasts signal to the speaker RPi Picos.

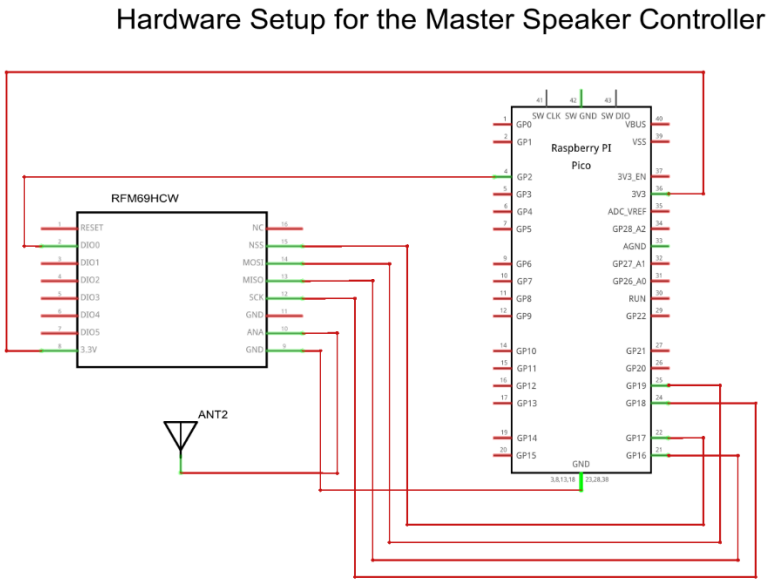

Figure 63: Circuit diagram of the master transceiver with the RPI Pico

Lastly, the circuit for the microphone RPi Pico is shown in Figure 64 below, utilizing I2S communication. The omnidirectional microphone, INMP441, is connected with its Serial Data (SD), Serial Clock (SCK), and Word Select (WS) pins linked to RPi Pico's GPIO pins 11, 12, 13, respectively.

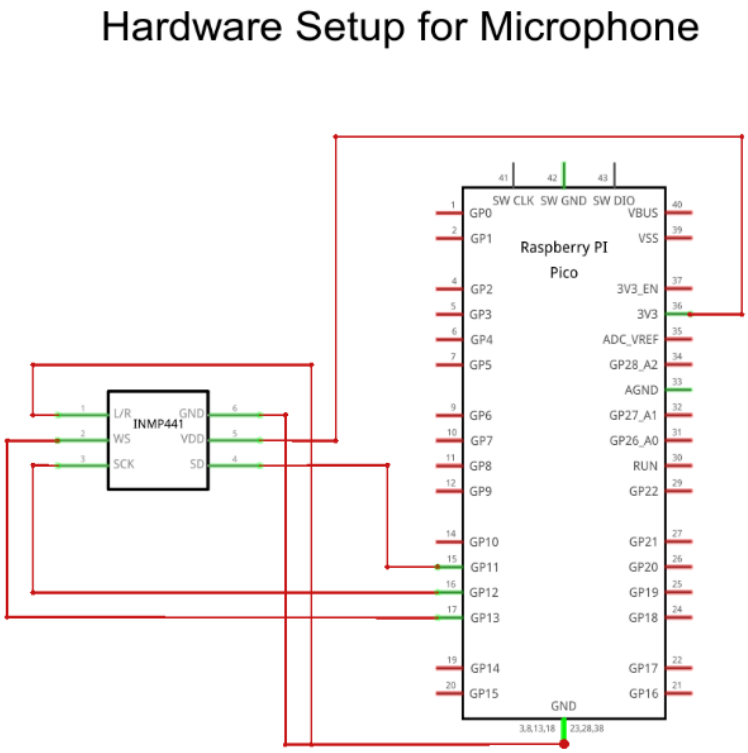

Figure 64: Circuit diagram of the receiver microphone with the RPI Pico

8.5. Future Work

Regarding future work, there are many ways in which sound localization can be expanded upon. One method to improve sound localization would be having the speakers installed on the ceiling. In a grocery store environment, this would be the most beneficial as it would be less visible to the customers and angling the speakers from the ceiling would allow for a better line of sight. In addition, there could be multiple speakers installed on the ceiling, then the 'best' speakers (ie. the ones having the best correlation strength) could be used in the TDoA/ToA algorithm. This dynamic decision of choosing the best speakers would provide a more accurate and precise localization result.

Another implementation for the future would be using sounds that are inaudible to the human ear. Sounds with frequencies higher than 20 kHz must be used to accomplish this. However, the fact that service animals which can easily hear sounds of very high frequencies will be present in the store should also be considered.

In addition, another extension for sound localization would be using the official RP Pico SDK or using a microcontroller which has I2S built in with official support for the RFM69HCW. As previously mentioned, the RP2040 microcontroller has programmable input/output pins, which were used to implement I2S. However, having a microcontroller with this already implemented would be more time efficient. Using the RP Pico SDK would also eliminate the need for using the Arduino-Pico core, which had a few problems as previously mentioned in Section 8.3.3.

Finally, the sound localization method could be fully integrated with the cart. The current indoor localization with LiDAR mapping provides good results, however the use of sound localization could provide even more precision and accuracy, with the theoretical best being around 7 mm.

9. Integration

9.1. I²C Communication

9.1.1. Raspberry Pi Communication with Pico

To allow for wired communication between the Pico and the Raspberry Pi 3, a consistent and suitable communication protocol had to be established and implemented physically and then configured at each node. The two primary candidates were SPI, which stands for Serial Peripheral Protocol, and I2C, which is Inter-integrated circuit. Both protocols utilize a similar master-slave architecture. I2C utilizes two wires (or buses) to transport data and clock through SDA and SCL respectively. SPI, however, requires four wires to properly connect the two modules.

I2C was ultimately chosen. I2C, unlike SPI, is half-duplex, meaning each module can both receive and transmit data, but not at the same time. SPI is full-duplex meaning both modules can receive and transmit data at the same time. This was not a problem in the project's implementation as between the Raspberry Pi and the Raspberry Pi Pico, one-way transportation for data was all that was needed (Data was sent from the Raspberry Pi to the Raspberry Pi Pico). Another reason for choosing I2C as opposed to SPI was due to the chosen Pico's limitations, both hardware wise and when examining the configuration required in the software. The primary issue with implementing SPI was the inability to utilize the libraries effectively due to the lack of robust support for the protocol. Both the arduino-pico library and the ArduinoCore-mbed library for the Pico hosted many errors and inconsistent results in early testing. The main tradeoff in utilizing I2C was the hit in speed. Where SPI supports up to 10 mb/s transmission speed, I2C only supports up to 5 mb/s [28]. With extensive field testing though, this limitation proved to be irrelevant as data did not need to be transmitted this fast and I2C proved to not be the bottleneck in speed transmission anyways.

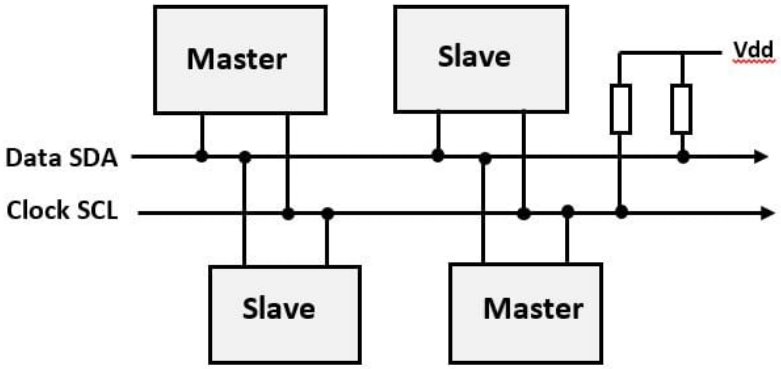

Figure 65: SDA and SCL Buses for I²C communication [29]

As seen in Figure 65 above, I²C primarily works by registering both masters and slaves to a common bus. The SCL bus is connected to both modules' GPIO pins through a wire. In the project's application, the Pico was configured as the receiving slave and the Raspberry Pi 3 was configured as the transmitting master. The slave registers itself on the SDA bus using a customized address (for example 0x12). The Raspberry Pi then directs its data transmission to the specified address and sends a byte or a stream of bytes. The data is then received by the Pico and is handled and processed. This is done through an interrupt.

The library utilized on the Pico end was the Mbed I2C library which wraps the already existing Arduino library, Wire, to better fit the Pico's architecture. The Pico program is interrupted once data is received. The interrupt is serviced, and the data is stored in a buffer. The interrupt-based implementation removes the need for the constant polling of the wire.

For the Raspberry Pi, a Python program was created implementing the smbus library, allowing us to interface and communicate with the Pico. The I2C address for the Pico was hardcoded in the Python program and the address was confirmed using the i2cdetect -y 1 terminal command to ensure that the Raspberry Pi was able to recognize the I2C connection and ensure that the right address was being used.

For the design of data transfer for this communication bus, a stream of bytes is sent periodically and is expected on the receiving end. Three bytes are sent as shown in Figure 66 below:

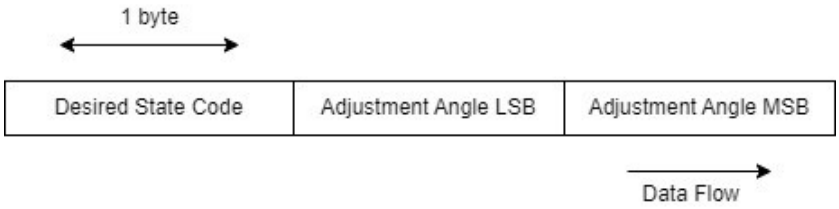

Figure 66: Byte arrangement for data transfer over I²C

The adjustment angle is the angle of the cart when compared to the nearest point of interest placed in the hallway. The points of interest are used to indicate a change in state for a turn or to angle the cart towards the center of the current hallway. This is to ensure that the cart remains in the center of the hall and doesn't collide with the walls. The control flow of the received data is shown below in Figure 67:

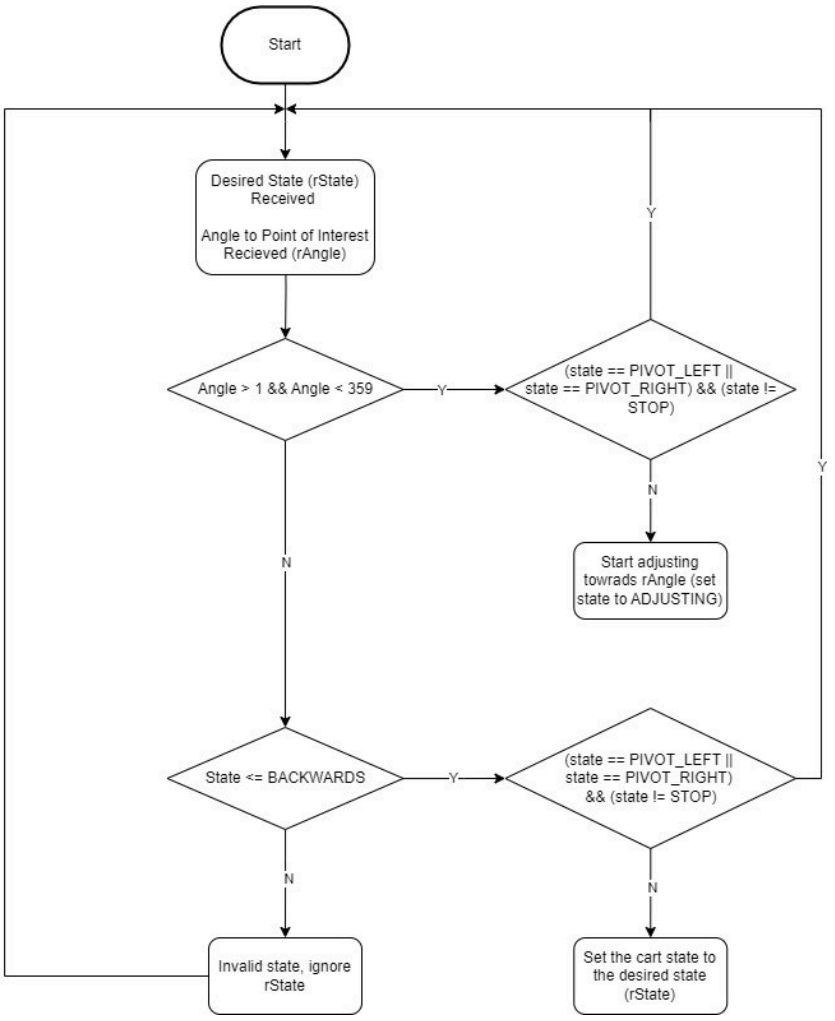

Figure 67: Control flow diagram on the Pico end.

Upon receiving a stream of data from Raspberry Pi and, by extension, the LiDAR, the angle is first processed to check whether the cart has veered away from the center of

the hall. An accuracy of one degree is used in this check. To avoid interrupting pivots, adjustment requests are ignored when a cart is already pivoting. No adjustments occur when the cart is stopped as well. If no adjustment is required, the passed in state is checked to see if it's valid. If the state is valid, the new state is checked (the stipulations of the adjustment branch apply here).

9.2. Socket Communication

9.2.1. C++ to Python

Our physical LiDAR interfaced with a C++ file through UART communication, this C++ file would read 1450 different LiDAR data points every 125 milliseconds. As our system architecture relied on reading real-time LiDAR data into a python file, we opted to create a socket to allow for this transmission. That being said, our team strategically modified the C++ file to send values through a socket to a Python file, as the Python file acted as the intersection between all other files. To do this, we had to send 8000 datapoints every second, with every datapoint being a singular laser emission. Datapoints consisted of angle, distance, and quality. Figure 68 below outlines the structure of each datapoint.

```
datapoint = LiDAR.receive()

datapoint.distance <--- "distance in millimetres"
datapoint.angle <------ "angle of laser"
datapoint.quality <---- "either 0 or 47"
```

Figure 68: Datapoint structure

LiDAR sensors laser beams into the environment and measure the time it takes for the laser to bounce back. However, if the laser beam does not bounce back, the data is considered inconclusive. As our team identified this major drawback, our team needed a way to identify these inconclusive scans. This is where the quality of each datapoint comes into play. As seen in the figure above, each datapoint has a "quality" attribute which is either 47 or 0. This attribute represents the conclusiveness of the datapoint. If the quality is 47, the laser bounced back to the LiDAR and the datapoint is valid. On the contrary, if the quality is 0, the laser did not bounce back to the LiDAR and

the datapoint is invalid. Our team designed a way to filter out unwanted data before socket transmission. A pseudocode of this design is shown below in Figure 69.

```
datapoints = LiDAR.fullRotationData()

#Iterate all datapoints
for (i < datapoints.len):
    #Only if datapoint is VALID (quality 47)
    if(datapoints[i].quality == 47):
        sendSocket(datapoints[i]) # Send Data
```

Figure 69: Pseudocode of Data Filtering

After filtering unwanted data, our data was now ready for socket transmission to the Python File. We needed to design a way to send every datapoint (distance, angle) in real-time, without any delay. We did this by packing the datapoint into a struct and opening a socket on a specified port between the C++ and Python files. We opened the socket on Local Host since both files were transmitting through the same host, and the port was chosen at runtime. A pseudocode of this transmission is shown below in Figure 70.

```
# C++ file (main.cpp) - SENDING

#open socket
socket = createConnection(IP Address, Port);

#send data
struct values datapoint; # Create struct
datapoint.degrees = LiDAR.readDegrees(); # 4 bytes
datapoint.distance = LiDAR.readDistance(); # 4 bytes
sendValues(&python_socket, datapoint); # send 8 bytes



# Python File (simpleSocket.py) - RECEIVING

# open socket
socket = bind(IP Address, PORT)
socket.listen()

# receive data
data = socket.receive(8) # receive 8 bytes
print(data.distance) # 4 bytes
print(data.angle) # 4 bytes
```

Figure 70: Pseudocode of Transmission

As seen in the pseudocode above, each datapoint was a maximum of 8 bytes. 4 bytes for the distance and 4 bytes for the angle. The datapoint is packed as a struct and received on the other end.

9.2.2. Python to MATLAB

The MATLAB program is responsible for calculating the position of the cart based on the data from a full revolution of the LiDAR. MATLAB is unable to run on ARM based processors, such as the one in the Raspberry Pi used as the 'main brain' of the Autocart. It was therefore necessary to run the MATLAB program on another computer and communicate between this computer and the cart over Wi-Fi. Wi-Fi having varying data rates and latencies impacted the way the communication link was designed. The design is shown below in the form of a sequence diagram:

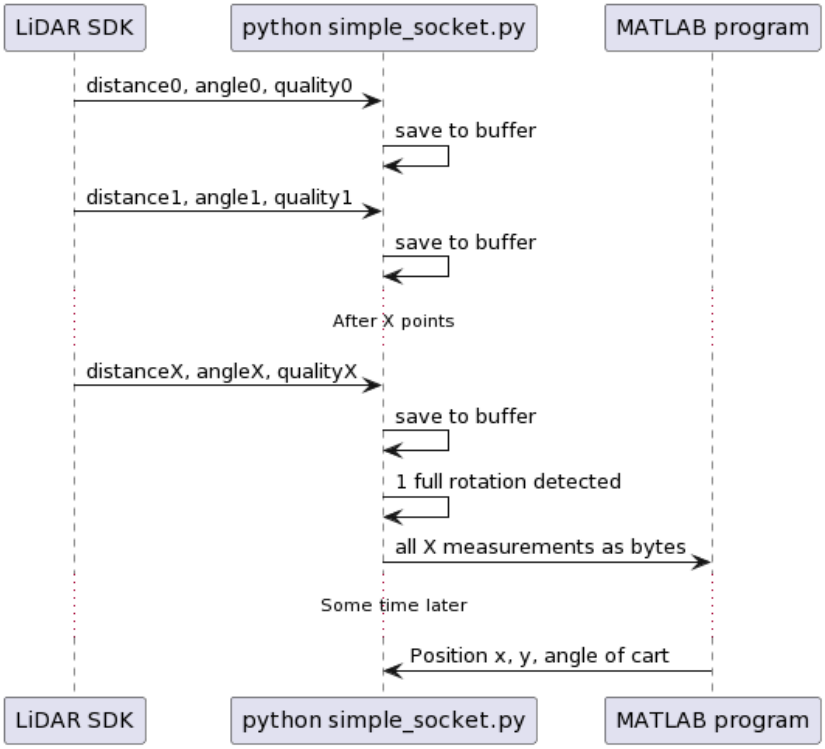

Figure 71: Sequence diagram for the communication of data between the LiDAR SDK, the Python program and the MATLAB program.

The Python program will only receive the useful measurements from the LiDAR SDK, measurements which have quality greater than 0. This means there are a varying number of measurements every rotation. It is then responsible for collecting them in a buffer and detecting when a full rotation has been completed. This is accomplished by

checking the angle measurement. When the difference between the current measurement and the previous is negative, such as in the case when the previous angle=359° and current angle=0°, a rotation has completed. It then sends the data to the MATLAB program and awaits a response.

The data is sent over a TCP socket, in a similar fashion to what was implemented between the LiDAR SDK and the Python program, as explained in the previous section. One key difference is that blobs of data containing measurements for a full rotation are sent, instead of the measurements one by one. To accomplish this, a header is added to the data detailing how many measurements it contains. The MATLAB program first reads this header, then waits until it has received that amount of measurements before beginning the position calculations.

A TCP socket was chosen over UDP to assure that the data is received at the MATLAB program without having to worry about packets being received out of order or not at all.

In the case that the communication with the MATLAB program is too slow, or that the MATLAB program takes too long to compute, the next revolution's worth of data is not sent. This is to prevent queuing up older data over the socket. Using this method assures that only the most recent revolution is handed off for computation to the MATLAB exprogram. A sequence diagram detailing what occurs when a reply is not received in time is shown below:

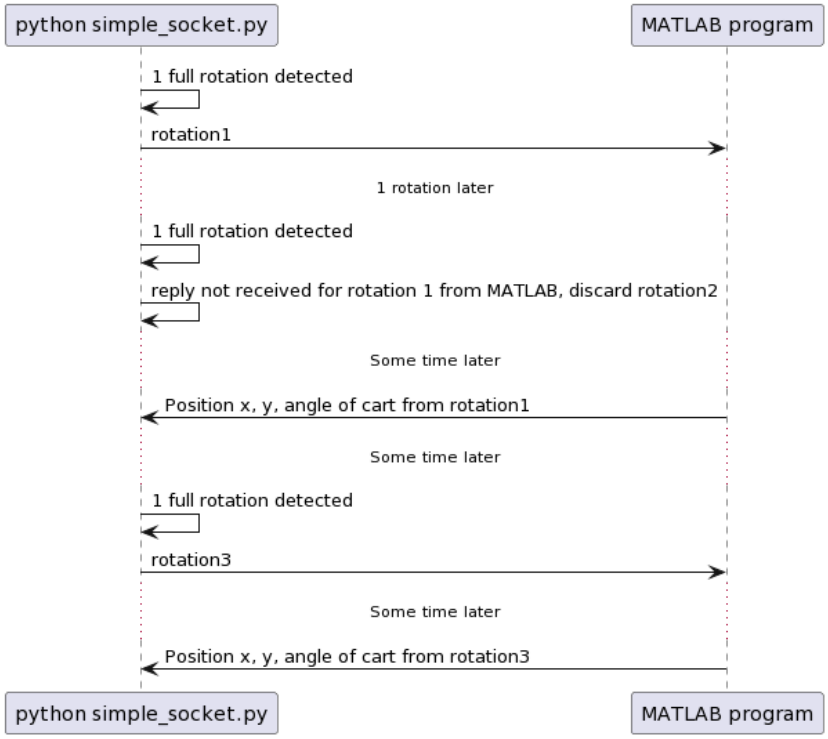

Figure 72: Sequence diagram showing the Python program's behavior when it does not receive a response from MALTAB quickly enough.

This is not the optimal solution, as if the network bandwidth is good enough to transmit the data in time, the only problem being the latency, then this scheme will get fewer measurements than if the Python program simply sent every rotation. In the case of 4th floor Mackenzie though, the location in which the demo was held, the network bandwidth is not consistently high enough for this. In some locations, the bandwidth low enough that the revolution data builds faster than it can be sent over the Wi-Fi connection. If the data for every revolution was sent in this case, the returned position estimate from MATLAB would be based on earlier and earlier revolutions as time progresses. As this continues, the queued data would build up on the sending side, and at some point, the TCP socket buffer would overflow.

Efforts were taken to translate this MATLAB program to the Rust programming language in order to allow it to run natively on the Raspberry Pi and avoid undesired scenarios that would occur if the cart was unable to communicate with the MATLAB program for prolonged periods of time. The Rust program seemed to function for the

most part, but due to time constraints we were unable to verify that it worked as well as the MATLAB program, and thus used the MATLAB program for the demo.

9.3. Making the Robot Move in a Predefined Path

Once the position of the cart is acquired in the 'Main Brain' Python program based on a LiDAR revolution sent to the MATLAB program, communication with the Raspberry Pi Pico motor controller can be done to move the cart to in the direction of a desired position.

The planned trajectory of the cart is shown in the figure below:

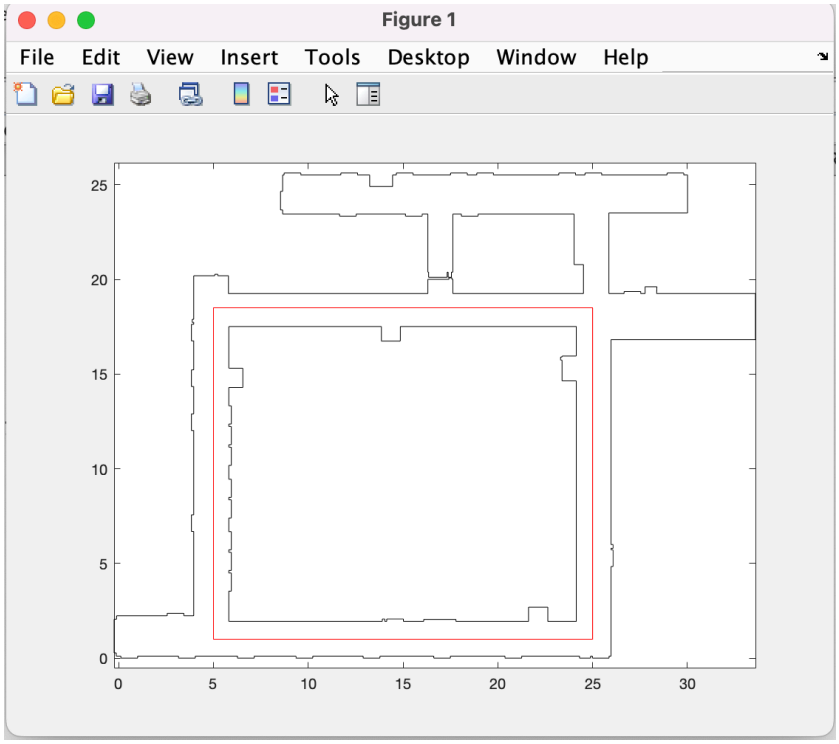

Figure 73: Map showing the planned trajectory of the cart.

To accomplish this, different "points of interest" were set up in a list in the 'Main Brain' Python program. The cart is directed towards the first point of interest in this list. When the cart gets within a certain threshold of the point, say half a meter, this point is removed from the list, and the cart is directed towards the next point.

Using this methodology, points were defined with x and y components based on the hand measured map used in the MATLAB program, and are shown in the figure below:

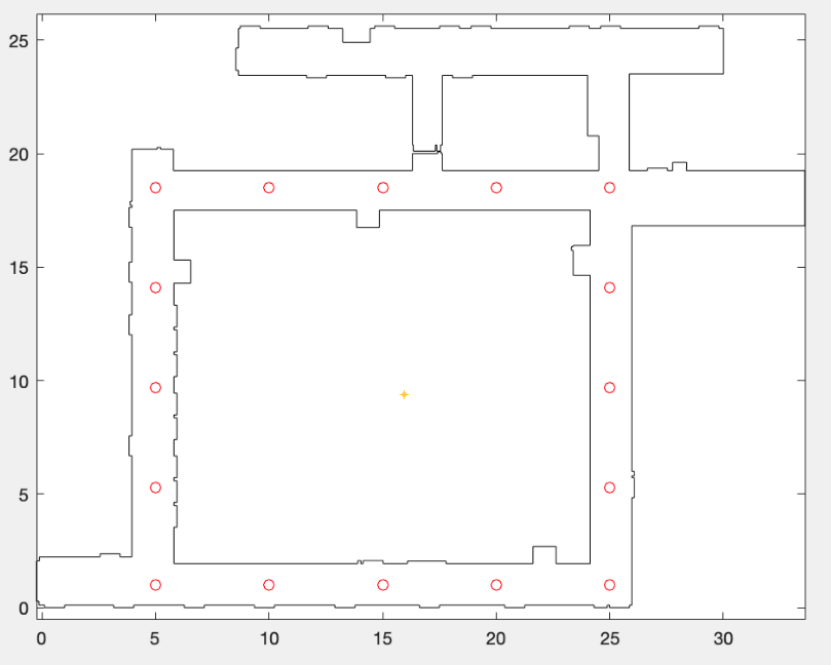

Figure 74: Points of interest defined to accomplish the trajectory shown in Figure 73.

The data available from the MATLAB program, as well as the information needed to send to the motor microcontroller is shown in the following figure:

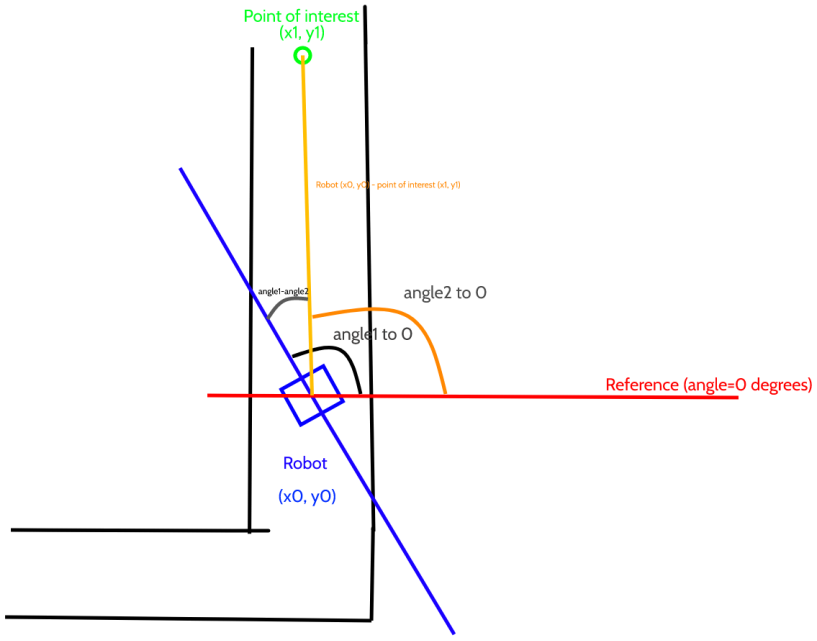

Figure 75: A drawing of the cart in a hallway of 4th block Mackenzie 4th floor, with annotated information

The MATLAB program provides the robot's cartesian coordinates as well as the orientation to polar 0. These are denoted as (x0, y0) and angle1 to 0 respectively in the

figure above. A line can then be drawn from the cart's current location (x0, y0), and the point of interest, (x1, y1). The angle of this line, subtracted with the angle1 to 0 from MATLAB, results in the angle adjustment required for the robot to face the point of interest directly.

When this adjustment angle is relatively low, instructions are sent to the motor microcontroller to adjust the direction of the cart while still rolling forwards. If the angle is above a certain threshold, say 75 degrees, then a message to stop and pivot is sent instead.

Originally in early tests only four points of interest were defined, those in the corners. This functioned but it was found that the cart would not be centered in the hallway, instead tending to the left. This was due to numerous issues, such as the cart not driving very straight to begin with due to a broken encoder, as well as the MATLAB program not returning very fine angle measurements and problems related to inconsistent Wi-Fi connections. Steps were taken to measure the angle more finely in the MATLAB program, as well as making the left side of the cart roll faster than the right to fix the left tending cart motion. Adding more points of interest was also found to help solve this issue.

10. Budget

For the Capstone Project, each team was given a $500 budget. Since the project was comprised of two teams, the overall budget for the project was $1000. Estimation of costs are summarized below:

Purpose	Item	Buyer	Cost	Amount	Shipping	Total Cost
Lab 2	LCD Screen	Kousha	$16.24	3	$8.00	$64.09
Pin Headers	112pcs male/ female pin headers	Bren	$20.99	1	$0.00	$23.72
Lab 3	20pcs IR emitter and receiver diodes	Bren	$9.09	1	$0.00	$10.27
Sensor Mini Project	Adafruit Ultimate GPS Module	Kousha	$41.95	1	$16.00	$65.48
Localization	Slamtec RPLIDAR A1M8	Prianna	$147.00	1	$0.00	$166.11
Localization	INMP441 Omni-directional Microphone Module (5)	Prianna	$27.88	1	$0.00	$31.50
Localization	RP Picos, RF TXRX modules	Prianna	$71.50	1	$0.00	$80.80
Localization	MAX98537a DAC/amp modules (5)	Seb	$20.88	1	$0.00	$23.59
Localization	8 ohm 3 W speaker driver (4)	Seb	$14.00	1	$0.00	$15.82
Localization	RP picos (3 non-soldered headers, 2 with)	Seb	$30.95	1	$2.00	$37.23
For Cart	Robot Chassis, 4wd	Bilal	$153.58	1	$0.00	$173.55
For Cart	Motor Drivers	Bilal	$61.00	1	$0.00	$68.93
For Cart	Extra Motor	Bilal	$19.48	1	$25.50	$50.83
For Cart	Battery Replacement	Kousha	$43.00	1	$0.00	$48.59
Poster Fair	Posters	Nadia	$214.00	1	$0.00	$241.82
Total Spent						$1102.34

Table 5: Expenses for the Project

11. Applications

As our autonomous robot project progresses, it may need to adapt itself to the environment that it's being applied to. One of the biggest areas of future work lies in scaling up and implementing a real-world application for the robot. This can include applications such as grocery carts, food delivery, or warehouse automation. Each application, however, has its own separate considerations that must be addressed.

Implementing a self-driving grocery cart may be very challenging. A grocery store will have many robots driving around the store at once, this will increase the chance of collision. Additionally, crowded areas may worsen the accuracy of the indoor LiDAR localization features. Scaling up to a grocery cart will also require adding a basket and potentially a robot arm on the robot. Automating a grocery store may also require aisle navigation and product localization, for the robot to find certain products by itself.

Figure 76: Crowded Grocery Store [30]

A food delivery autonomous robot must include a compartment to keep the temperature of the food stable until completion of delivery. The physical robot chassis and hardware components will also need to withstand all weather conditions, as well as varying terrain in the outdoor environment.

Figure 77: Food Delivery Robot [31]

A warehouse can be automated with the help of self-driving robots, through transportation of heavy objects. Of course, this would require the robot chassis to be scaled up to withstand heavier loads. Additionally, working in a warehouse requires extra safety features to be added to the robot, including more sensors and failsafe mechanisms.

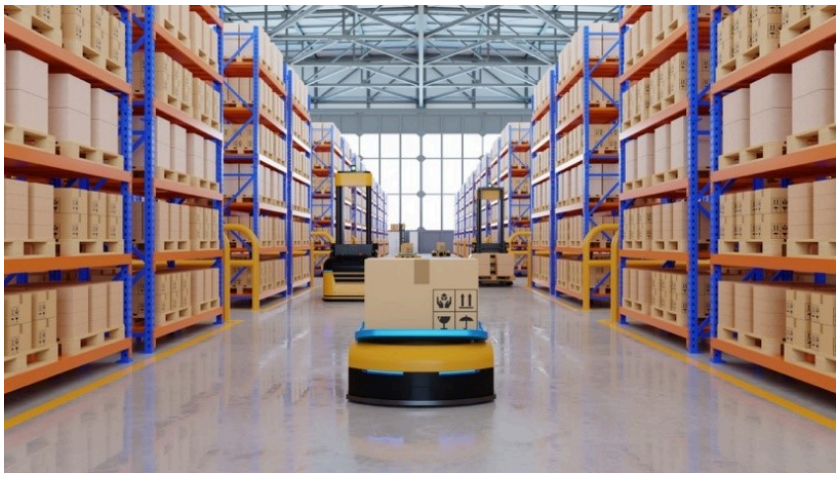

Figure 78: Automated Warehouse Robot [32]

12. Reflection

Our original project goal was to create an automated shopping cart capable of doing groceries for clients automatically. It would roam the grocery store aisles, grab items with a mechanical arm and put them into the cart. It would navigate using some form of indoor localization and collision avoidance algorithms and have some way to verify grabbed products were the expected ones.

This was a very ambitious goal. Realizing this, we restrained the scope to accomplishing the localization, collision avoidance, and navigational aspects. These elements not only be applied to automated shopping, but to shuttle robots, robots in warehouses, etc.

To accomplish the automated navigation, having accurate indoor localization is important. Two methods were explored for this: LiDAR localization and sound localization. We pursued two different methods in the hopes that one of them would be ready for integration within the time constraints of the project. Both solutions ended up being functional, and we chose the one that would result in the best demonstration: LiDAR localization. LiDAR provided many advantages such as not needing any setup in the environment, as well as being able to provide the current rotation of the cart. Sound localization would have needed a large room with no objects blocking line of sight between the speakers and microphones. Providing angle measurements would have been more complicated, requiring two microphones on the cart.

Pursuing localization brought challenges in planning for the project. Neither method was simple to implement or guaranteed to function within the time constraints. It was only relatively late into the project that either method started showing potential. It was thus unclear what kind of demo would be possible. It was only once the LiDAR localization was able to accurately show the position as well as the rotation of the cart anywhere in 4th floor 4th block Mackenzie did we decide to hold our demo there, and finalized what information would be passed from the localization subsystem to the rest of the cart.

This was a hindrance for some time for the entire capstone group because as the localisation algorithm was being figured out, it was unclear what information could be provided to the motor subsystem to direct the cart. The object avoidance algorithm had to be tested only when the localisation algorithm worked. If there was a clearer sense of the localization actualizing, this would put a better focus onto the motors and object avoidance section of the project to, potentially scaling the project up to as mentioned in Chapter 12. The localization aspect does not depend on the size of the vehicle, but the mechanical side depends on this.

To accomplish the collision avoidance, both ultrasonic sensors and the LiDAR were explored. Ultimately, only the LiDAR was used. The LiDAR was deemed enough to detect collisions at the cart's speed.

If we were to do this project again, our initial goals would align more with the localization and navigational aspect we ended with. Our project would have been better titled as "Autonomous Shuttle", with the environment being the entirety of the Mackenzie buildings 4th floor, across all the blocks.

Another aspect we would do differently is the integration. Near the end of the term, we had separate sections of our project working: motors and indoor localization. However, integrating these two things together took more time and effort than what was initially considered. Thus, if we were to do the project again, we would focus on performing the integration sooner. By doing so, it would allow us more time to debug errors and implement more features into the final project, for example using sound localization in addition to LiDAR localization.

Another difference we would make is initial choice of the cart. In the beginning of our project, we wanted to use a full-sized shopping cart that is used in grocery stores. However, we found this infeasible due to time and budget constraints. If we had started with a smaller model cart first, we could have finished the integration of the project sooner and added more features with the additional time.

As previously mentioned in Section 8.5, the use of the Arduino-Pico core caused various issues with the RFM69HCW transceiver module, as well as de-synchronization of the speakers. If we were to do the project again, we would use the official RP Pico SDK to avoid running into these problems.

As a collective, the opportunity to attempt two different localization techniques is something that was very helpful for all members that worked on this section of the project. We learnt valuable insight into localization and give an accurate assessment of what localization is. This was done due to the supervising professors; with them urging the members to delve deeper into localization and realizing that it is central towards the success of the project, it turned to some fruitful results.

13. Conclusion

Overall, the project saw many depths and intricacies between varying aspects and produced results that are promising enough to be improved to create a serviceable final product.

Indoor localization techniques were used to precisely determine the location and orientation of the robot so that it could follow a provided path within the indoor map. To achieve this feat is promising however could be improved through implementing an interactive map or real-time map form.

Collision avoidance was successfully integrated after researching various types of sensors and algorithms. The optimal sensor and algorithm worked together to provide the robot with the capability of accurately avoiding obstacles in its path. This way, our robot was able to safely drive without colliding with objects, people, or walls.

Motor control was integrated to a level that allowed for the basic testing of the localization and collision avoidance aspects of the vehicle. This means that the ability to perform approximate turns, approximate adjustments, and immediate braking were all considered. Future expansions in this area would primarily focus on scaling up the size of the vehicle to allow for mechanical and application flexibility. Improving accuracy in the turning and adjustments through higher resolution encoders and more robust wheels would also be another step forward for improving motor control.

Sound localization achieved accurate and precise results, to at least centimeter precision. In the future, this work can be expanded by installing the speakers in the ceiling, using multiple speakers and choosing the best performing ones for the ToA/TDoA algorithm, using sounds inaudible to the human ear, using the official RP Pico SDK, and fully integrating sound localization into the cart.

References

[1] "What is an Ultrasonic Sensor?." [Online]. Available: https://www.fierceelectronics.com/sensors/what-ultrasonic-sensor

[2] "SSZTA28 Technical article | TI.com." [Online]. Available: https://www.ti.com/document-viewer/lit/html/SSZTA28

[3] B. J. Smoot, "The Basics of Ultrasonic Sensors | CUI Devices." [Online]. Available: https://www.cuidevices.com/blog/the-basics-of-ultrasonic-sensors

[4] SLAMTEC Global Network, "SlamTec RPLIDAR A1 - SLAMTEC Global Network." [Online]. Available: https://www.slamtec.ai/product/slamtec-rplidar-a1/

[5] J. Vogel, "Tech explained: Ackermann steering geometry." [Online]. Available: https://www.racecar-engineering.com/articles/tech-explained-ackermann-steering-geometry/

[6] Z. Li, J. Li, and S. Yang, "The study on Differential Steering Control of In-wheel Motor Vehicle Based on Double Closed Loop System," *Energy procedia*, vol. 152, pp. 586–592, Oct. 2018, doi: 10.1016/j.egypro.2018.09.215.

[7] X. Wu, M. Xu, and L. Wang, "Differential speed steering control for four-wheel independent driving electric vehicle." [Online]. Available: https://www.semanticscholar.org/paper/Differential-speed-steering-control-for-four-wheel-Wu-Xu/ebafd717228deb00cc8f78efc682eb0fef1755f8

[8] Shalom Education, "Electric Motors." [Online]. Available: https://www.shalom-education.com/courses/gcse-physics/lessons/magnetism-and-electromagnetism/topic/electric-motors/

[9] W. Yeadon and A. Yeadon, "Handbook of Small Electric Motors." McGraw-Hill, 2001.

[10] K. Morgan, "Torque and Angular Momentum in Circular Motion," 2002, [Online]. Available: https://www.physnet.org/modules/pdf_modules/m34.pdf

[11] Raspberry Pi Ltd., "Raspberry Pi Documentation - Raspberry Pi Pico and Pico W." [Online]. Available: https://www.raspberrypi.com/documentation/microcontrollers/raspberry-pi-pico.html

[12] Robocraze, "What is Motor Driver." [Online]. Available: https://robocraze.com/blogs/post/what-is-motor-driver

[13] Ø. N. Dahl, "What is an H-Bridge?." [Online]. Available: https://www.build-electronic-circuits.com/h-bridge/

[14] HandsOn Tech, "BTS7960 Motor Driver User Guide." Accessed: Apr. 09, 2024. [Online]. Available: http://www.handsontec.com/dataspecs/module/BTS7960%20Motor%20Driver.pdf

[15] T. Smy, "MOSFET transistors." Accessed: Apr. 09, 2024. [Online]. Available: http://doe.carleton.ca/~tjs/lectures/snew20b.pdf

[16] US Digital, “What's the difference between optical, magnetic and capacitive encoders?.” [Online]. Available: https://www.usdigital.com/blog/difference-between-optical-magnetic-and-capacitive-encoders/

[17] NI, “The PID controller & theory explained.” Accessed: Apr. 09, 2024. [Online]. Available: https://www.ni.com/en/shop/labview/pid-theory-explained.html

[18] Wikipedia contributors, “Proportional–integral–derivative controller.” [Online]. Available: https://en.wikipedia.org/wiki/Proportional%E2%80%93integral%E2%80%93derivative_controller

[19] Amazon.ca, “Mecanum Wheel 4wd Metal Robot car Chassis Remote Control Learning Kit for Arduino Raspberry Pie Microbit with 4pcs DC Encoder Motor, DIY Steam AGV ROS AI Move Education Platform Robotic Model.” [Online]. Available: https://www.amazon.ca/Learning-Raspberry-Microbit-Education-Platform/dp/B09KLGKWNG

[20] Wikipedia contributors, “Deming regression.” [Online]. Available: https://en.wikipedia.org/wiki/Deming_regression

[21] “TensorFlow Machine Learning Cookbook.” [Online]. Available: https://subscription.packtpub.com/book/data/9781786462169/3/ch03lvl1sec32/implementing-deming-regression

[22] H. Obeidat, W. S. Shuaieb, O. Obeidat, and R. A. Abd-Alhameed, “A review of indoor localization techniques and wireless technologies,” *Wireless personal communications*, vol. 119, no. 1, pp. 289–327, 2021, doi: 10.1007/s11277-021-08209-5.

[23] M. Richards, *Fundamentals of Radar Signal Processing, Second Edition*. McGraw Hill LLC, 2013. [Online]. Available: https://www.mheducation.com/highered/product/fundamentals-radar-signal-processing-second-edition-richards/9780071798334.html

[24] B. Fang, “Simple solutions for hyperbolic and related position fixes,” *IEEE Transactions on Aerospace and Electronic Systems*, vol. 26, no. 5, pp. 748–753, 1990, doi: 10.1109/7.102710.

[25] F. Zhang, H. Li, Y. Ding, S.-H. Yang, and L. Yang, “Dilution of precision for time difference of arrival with station deployment,” *IET Signal Processing*, vol. 15, no. 6, pp. 353–364, 2021, doi: https://doi.org/10.1049/sil2.12036.

[26] Raspberry Pi Ltd., “RP2040 Datasheet: A microcontroller by Raspberry Pi.” [Online]. Available: https://datasheets.raspberrypi.com/rp2040/rp2040-datasheet.pdf

[27] HOPERF Electronic, “RFM69HCW ISM TRANSCEIVER MODULE.” [Online]. Available: https://cdn.sparkfun.com/datasheets/Wireless/General/RFM69HCW-V1.1.pdf

[28] Prodigy Technovations, “I2C vs SPI.” [Online]. Available: https://www.prodigytechno.com/i2c-vs-spi

[29] Pico Technology Ltd., “I²C - serial protocol decoding.” [Online]. Available: https://www.picotech.com/library/oscilloscopes/serial-protocol-decoding-i2c

[30] Eat This Not That, “These are the most High-Risk times to visit the grocery store.” [Online]. Available: https://www.eatthis.com/high-risk-times-grocery-store/

[31] Pymnts, “Uber-Backed robotics startup predicts autonomous food delivery will be in all major cities within five years.” [Online]. Available: https://www.pymnts.com/news/delivery/2021/robotics-startup-predicts-autonomous-food-delivery-will-be-in-all-major-cities-within-five-years/

[32] J. Daleo, “Your guide to the wild world of warehouse robots,” Apr. 2022, [Online]. Available: https://www.freightwaves.com/news/your-guide-to-the-wild-world-of-warehouse-robots

Appendix A: Code

The source code can also be found at https://github.com/autocart-capstone/autocart

robot-demo/Socket_approx_program.m

```
%{
This is a Mapping file for getting the current points on the cart
Gets the LIDAR from the pi then processes it and send back the
approximate
location of the cart
%}
load('ahmed.mat');
figure(1);
%figure('KeyPressFcn',@myKeyPressFcn);
plot(structures{1}(:,1), structures{1}(:,2), 'k'); % Plot walls
hold on
plot(structures{2}(:,1), structures{2}(:,2), 'k'); % Plot walls
xlim([-5 35]);
ylim([-5 30]);
set(gca, 'ButtonDownFcn', @plotClickCallback);
set(gcf, 'KeyPressFcn', @ResetMAP);

%disp(['You clicked at X:', num2str(x_clicked), ', Y:',
num2str(y_clicked)]);
X_meas = measures(1,:,:);
Y_meas = measures(2,:,:);
D_meas = sqrt(X_meas.^2 + Y_meas.^2);

last_pos = nan(1,2);
t = tcpclient('localization-pi.duckdns.org', 8003, "Timeout", 100000);
a = [0.0,-1.0,0.0,1.0,0.0];
r = a*0.2;
b = [1.0,-1.0,0.0,-1.0,1.0];
pgon = polyshape(a,b);
pgon = scale(pgon,0.3);
global Reset;
Reset=false;
angle = 0;
x_new =0;
y_new = 0;
last_pos = nan(1,2);
while true
    size = read(t,1, "int32");
    %disp(size);

    floats = read(t,size,'single');
    floats = reshape(floats, 2, []);
    theta = floats(1,:);
    distance = floats(2,:);
    % if h ~= 0
    %     delete h;
    %  end
    hold on
    [x, y, angle,x_new,y_new] =
position(theta,distance,angle,test_pos,measures,walls,D_meas,last_pos,x_new,y_new);
    if Reset==false
        last_pos=[x y];
    else
        last_pos=nan(1,2);
        Reset = false;
    end
    if exist('n','var')
        delete(n);
    end
```

```
    if exist('d','var')
        delete(d);
    end
    phi = -90+angle;
    z = rotate(pgon,phi);
    h = plot(translate(z,x,y));
    c = plot(x_new,y_new,'r.');
    %h = plot(translate(rotate(pgon, x, y,-90+angle));
    %disp(angle);
    write(t,[sprintf('%g %g %g',x,y,angle) newline]);
    hold off;
    n=h;
    d=c;
    %write(t,[x y])


end

% Nested callback function definition
function [x_clicked y_clicked] = plotClickCallback(src, ~)
    % Get the coordinates of the mouse click within the axes
    clickCoords = get(src, 'CurrentPoint'); % CurrentPoint returns
[x,y] of the click
    x_clicked = clickCoords(1, 1); % X coordinate of the click
    y_clicked = clickCoords(1, 2); % Y coordinate of the click

end

function ResetMAP(~, ~)
    global Reset;
    Reset = true;
end

% filename = "test425.txt";
% xoff = 24.63;
% yoff = 20.98;
% angle = 89.5;

%filename = "19thjantest2.txt";
%xoff = 24.63;
%yoff =  7.98;
%angle = 0;

function [xa,ya, angle, x_new,y_new] = position(theta, distances,angle,
test_pos,measures,walls,D_meas,last_pos,x_new,y_new)
% Read data file, and skip header and blank line at end
    xoff = 24.63;
    yoff = 22.50;

    % Extract angle, distance, and quality
    theta_raw = theta;
    distance = distances;
    distance = distance/1000;
    theta_fixed = mod((360 - theta_raw), 360);

    % Sort theta values
    [theta_fixed, idx] = sort(theta_fixed);
    distance = distance(idx);

    angle_list = 0:10:359;
    real_data = zeros(2,36);
    counts = zeros(1, 36);
    mean_distance = zeros(1, 36);
    fidx=1;
    pidx_near =1;
    pidx=1;
    for nx = 1:length(angle_list)
```

```
        current_angle = angle_list(nx);
        theta_range = [mod(current_angle-1,360),
mod(current_angle+1,360)];
        if current_angle == 0
            idx = (theta_fixed > theta_range(1) | theta_fixed <
theta_range(2));
        else
            idx = (theta_fixed > theta_range(1) & theta_fixed <
theta_range(2));
        end

        counts(nx) = sum(idx);
        mean_distance(nx) = mean(distance(idx));
        if (counts(nx) == 0)
            mean_distance(nx) = 0.0;
        end
    end

    minmetric = 1000000.*ones(1, 36);
    minidx = zeros(1, 36);
    if isnan(last_pos)
        for na = 1:length(angle_list)
            new_counts = circshift(counts, na-1);
            new_mean_distance = circshift(mean_distance, na-1);
            %metric = squeeze(sum(((X_meas - X_coord).^2 + (Y_meas -
Y_coord).^2).*new_counts));
            metric = squeeze(sum(abs(D_meas -
new_mean_distance).*new_counts));
            [minmetric(na),minidx(na)] = min(metric);

        end
        [~,fidx] = min(minmetric); % find best orientation
        pidx = minidx(fidx);  % find index of best nearby test position

    else
        test_pos_distance = sum((test_pos - last_pos).^2, 2);

        % Find the indices of the test positions that are near the
robot
        near_test_pos_idx = find(test_pos_distance < 0.7);
        % Discard all the non-near test positions
        D_meas_near = D_meas(:,:,near_test_pos_idx);
        %angle_list = 0:10:angle_list(fidx);
        %minmetric = zeros(1, 36);

        idx_angle= round(angle/10);
        nx = idx_angle - 9;
        %minidx = zeros(1, 36);
        for na = nx:(nx+18)
            new_counts = circshift(counts, na-1);
            new_mean_distance = circshift(mean_distance, na-1);
            %metric = squeeze(sum(((X_meas - X_coord).^2 + (Y_meas -
Y_coord).^2).*new_counts));
            metric = squeeze(sum(abs(D_meas_near -
new_mean_distance).*new_counts));
            if na < 1
                %disp("HELLO");
                [minmetric(na+36),minidx(na+36)] = min(metric);
            elseif na > 36
                [minmetric(na-36),minidx(na-36)] = min(metric);
            else
                [minmetric(na),minidx(na)] = min(metric);
            end

        end
        [minima,fidx] = min(minmetric); % find best orientation
        minima
        % disp(minima);
```

```
        %disp(fidx);
        pidx_near = minidx(fidx);  % find index of best nearby test
position
        %disp(pidx_near);
        pidx = near_test_pos_idx(pidx_near); % find corresponding test
position


        % Attempt to find a position within 1 metre of the last
    end
    xa = test_pos(pidx,1);
    ya = test_pos(pidx,2);

    precise_angle = angle_list(fidx) + (-5:.5:5);
    counts_pr = zeros(36, length(precise_angle));
    mean_distance_pr = zeros(36, length(precise_angle));

    for i =1:length(precise_angle)
        current_angle = precise_angle(i);
         for k= 1:length(angle_list)
             angle = angle_list(k);
             look_angle = angle - current_angle;

            new_theta_range = [mod(look_angle-1,360),
mod(look_angle+1,360)];
            if abs(look_angle) <= 1
                idx3 = (theta_fixed > new_theta_range(1) | theta_fixed
< new_theta_range(2));
            else
                idx3 = (theta_fixed > new_theta_range(1) & theta_fixed
< new_theta_range(2));
            end
            counts_pr(k,i) = sum(idx3);
            mean_distance_pr(k,i) = mean(distance(idx3));
            if (counts_pr(k,i) == 0)
                mean_distance_pr(k,i) = 0.0;
            end
         end
    end

    precise_cos = cosd(precise_angle);
    precise_sin = sind(precise_angle);
    new_D_meas = D_meas(:,pidx);
    metric_pr = squeeze(sum((new_D_meas-
mean_distance_pr).^2.*counts_pr))
    [minmetric_pr,minidx_pr] = min(metric_pr);
    angle = precise_angle(minidx_pr)
    %dodraw = true;
    x_new = xa + distance.*cosd(theta_fixed+angle);
    y_new = ya + distance.*sind(theta_fixed+angle);

end
```

robot-demo/simplesocket_v2.py

```
"""
Pseudocode:
Setup c++ socket connection to the C++ SDK
Read data coming from t)e SDK whenever current angle is less than
previous angle:
Store it in buffer(dynamic buffer) (1 revolution has 1200 samples)
Make sure matlab is done processing and ready to grab new data values
Send buffer contents (data: angle, distance) to matlab socket
connection

"""

import socket
import struct
import os
import subprocess
import time
import math
from smbus2 import SMBus

MATLAB_PORT = 8008
if "MATLAB_PORT" in os.environ:
    MATLAB_PORT = int(os.environ["MATLAB_PORT"])
print(f"will listen for matlab on port: {MATLAB_PORT}")

SDK_PORT = 8080
if "SDK_PORT" in os.environ:
    SDK_PORT = int(os.environ["SDK_PORT"])


class SimpleSocketRpi:
    def __init__(self):
        self.s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.m = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.buf = []

    def connect_socket(self):
        self.m.bind(("0.0.0.0", MATLAB_PORT))
        self.m.listen(1)

        print("starting to listen to matlab")
        conn, addr = self.m.accept()
        print("Connected by", addr)
        self.matlab_conn = conn
        self.matlab_conn.setblocking(False)

        self.s.bind(("127.0.0.1", SDK_PORT))
        self.s.listen(1)
        print("starting to listen to sdk")

        print("executing sdk")
        self.sdk_subp = subprocess.Popen(
            [
                "../rplidar_sdk-master/output/Linux/Release/
ultra_simple",
                "--channel",
                "--serial",
                "/dev/ttyUSB0",
                "115200",
            ],
            stdout=subprocess.DEVNULL,
        )
        # os.system("../rplidar_sdk-master/output/Linux/Release/
ultra_simple  --channel --serial /dev/ttyUSB0 115200")
        conn, addr = self.s.accept()
```

```
        print("Connected by", addr)
        self.sdk_conn = conn

    def send_data_to_matlab(self):
        self.matlab_conn.sendall(struct.pack("I", len(self.buf)))
        self.matlab_conn.sendall(struct.pack(f"{len(self.buf)}f",
*self.buf))

    def receive_data_from_matlab(self):
        try:
            return self.matlab_conn.recv(1024)
        except Exception:
            return b""

    def receive_data_from_lidar(self):
        d = self.sdk_conn.recv(8)  # reads up to 8 bytes of data
        if not d:
            print("Recevied empty string")
        # angle, distance = struct.unpack('ff', d)
        return d

    def save_data_to_buffer(self, data: tuple):
        self.buf += data

    def get_buffer(self):
        return self.buf

    def reset_buffer(self):
        self.buf = []


destinations = [
    (25, 18.5),
    (20, 18.5),
    (15, 18.5),
    (10, 18.5),  # TR -> TL
    (5, 18.5),
    (5, 14.1),
    (5, 9.7),
    (5, 5.3),  # TL -> BL
    (5, 1),
    (10, 1),
    (15, 1),
    (20, 1),  # BL -> BR
    (25, 1),
    (25, 5.3),
    (25, 9.7),
    (25, 14.1),  # BR -> TR
]


channel = 1
address = 0x12

bus = SMBus(channel)

# data_to_send = (0,0,3) #FORWARD
# data = bytes(data_to_send) #SEND
# bus.write_i2c_block_data(address, 0, list(data))

PICO_CMD_STOP = 0
PICO_CMD_TURN_LEFT = 1
PICO_CMD_TURN_RIGHT = 2
PICO_CMD_FWD = 3
PICO_CMD_BWD = 4

PICO_ANGLE_PADDING = list(int.to_bytes(0, length=2))
```

```
def got_position_from_matlab(cart_x, cart_y, cart_angle):
    dest_x, dest_y = destinations[0]
    if math.sqrt((cart_x - dest_x) ** 2 + (cart_y - dest_y) ** 2) < 1:
        print("-- Reached point of interest")
        destinations.append(
            destinations.pop(0)
        )  # put reached destination at the end of the list

    dest_x, dest_y = destinations[0]

    line_to_dest = dest_x - cart_x, dest_y - cart_y

    angle_to_dest_deg = (
        math.atan2(line_to_dest[1], line_to_dest[0]) / math.pi * 180.0
    )  # [-180, 180]
    angle_to_dest_deg %= 360.0  # [0, 360]

    angle_to_turn = int(cart_angle - angle_to_dest_deg) % 360  # [0,
360]

    print(
        "angle_to_dest:",
        angle_to_dest_deg,
        "cart_angle:",
        cart_angle,
        "angle_to_turn:",
        angle_to_turn,
    )

    cmd, angle_bytes = None, None

    if angle_to_turn > 60.0 and angle_to_turn < 180.0:
        cmd = PICO_CMD_TURN_RIGHT
        angle_bytes = PICO_ANGLE_PADDING
        print("PIVOT right")
    elif angle_to_turn < 300.0 and angle_to_turn > 180.0:
        cmd = PICO_CMD_TURN_LEFT
        print("PIVOT left")
        angle_bytes = PICO_ANGLE_PADDING
    else:
        print(f"FWD, angle = {angle_to_turn}")
        cmd = PICO_CMD_FWD
        angle_bytes = list(int.to_bytes(angle_to_turn, length=2))

    data = angle_bytes + [cmd]
    bus.write_i2c_block_data(address, 0, data)


def main():
    ss = SimpleSocketRpi()
    ss.connect_socket()
    curr_angle = 0
    prev_angle = curr_angle
    matlab_ready = True
    collision = False
    matlab_sent_timestamp = time.time()

    while True:
        # Receive from SDK
        data = ss.receive_data_from_lidar()
        data = struct.unpack("ff", data)
        curr_angle = data[0]  # Assuming first elem is angle
        ss.save_data_to_buffer(data)

        # Receive X,Y from MATLAB
        if collision   False:
            matlab_pos = ss.receive_data_from_matlab()
```

```
        # Process X,Y
        if matlab_pos and collision   False:
            matlab_ready = True
            a = matlab_pos.decode().strip().split()
            x = float(a[0])
            y = float(a[1])
            angle = float(a[2])
            print(f"received (x,y,angle): ({x},{y},{angle})")
            print(
                f"took {time.time() - matlab_sent_timestamp} seconds to
get position from Matlab"
            )
            got_position_from_matlab(x, y, angle)

        # Finish one revolution
        if curr_angle < prev_angle:
            if matlab_ready and collision   False:
                # Send buffer to matlab socket
                # print(ss.get_buffer())
                ss.send_data_to_matlab()
                print("sent data to Matlab")
                matlab_sent_timestamp = time.time()
                matlab_ready = False

            # Check for Collision
            collision = False
            for i in range(1, len(ss.buf), 2):
                print(f"Hit: {ss.buf[i]}")
                if ((ss.buf[i - 1] > 320) or (ss.buf[i - 1] < 40)) and
(
                    ss.buf[i] < 600
                ):
                    collision = True
                    print(
                        f"Collision avoided at Distance = {ss.buf[i]},
Angle = {ss.buf[i-1]}"
                    )
                    bus.write_i2c_block_data(
                        address, 0, PICO_ANGLE_PADDING +
[PICO_CMD_STOP]
                    )  # Send Stop
                    break

            # Reset buffer - always do this after finishing 1 rev
            ss.reset_buffer()
            # Save first data from the new rev
            ss.save_data_to_buffer(data)

        # If we are not done with one rev yet, keep saving data
(angle,dist) to buffer
        # Update prev_angle
        prev_angle = curr_angle


main()
```

robot-demo/pico-motors-development/calculations.h

```
#ifndef CALCULATIONS_H
#define CALCULATIONS_H

/* Cart Measurements
    length middle of wheel: 30.48 cm
    Width middle of wheel: 21.59 cm
 */

#define WHEEL_DIAM 9.7f   // cm
#define PIVOT_DIAM 38.1f  // cm

float calculate_pulses_for_angle(float angle);

float calculate_RPM(int pulses_elapsed);

#endif
```

robot-demo/pico-motors-development/calculations.ino

```
#include "calculations.h"
#include "pin_config.h"
#include <math.h>

static const float wheel_circ = WHEEL_DIAM * PI;
static const float pivot_circ = PIVOT_DIAM * PI;

static const float period = 0.10;

/* Method to calucalate the required encoder pulses to turn a given
angle */
float calculate_pulses_for_angle(float angle) {
  float pulses_per_full_pivot = PULSES_PER_REV * (pivot_circ /
wheel_circ);
  float pulses_for_theta = (angle / 360) * (pulses_per_full_pivot);

  return pulses_for_theta;
}

/* Method to calculate the current RPM of a wheel given the pulses over
a period of 0.1s */
float calculate_RPM(int pulses_elapsed) {
  float RPM = (pulses_elapsed * 60) / (period * PULSES_PER_REV);
  return RPM;
}
```

robot-demo/pico-motors-development/encoder_interrupts.h

```
#ifndef ENCODER_INTERRUPTS_H
#define ENCODER_INTERRUPTS_H

#define PULSES_PER_REV 90.0f

extern volatile int FL_turn_pulses;
extern volatile int BL_turn_pulses;
extern volatile int FR_turn_pulses;
extern volatile int BR_turn_pulses;

extern volatile int FL_speed_pulses;
extern volatile int BL_speed_pulses;
extern volatile int FR_speed_pulses;
extern volatile int BR_speed_pulses;
```

```
extern float FL_mtr_RPM;
extern float BL_mtr_RPM;
extern float FR_mtr_RPM;
extern float BR_mtr_RPM;

void init_encoders();
void init_RPM_timer();
void reset_encoders();

float getMotorRPM(int mtr_index);
bool check_all_motor_RPM(int RPM);

int getAvgPulsesLeft();
int getAvgPulsesRight();

#endif
```

robot-demo/pico-motors-development/encoder_interrupts.ino

```
#include "encoder_interrupts.h"
#include "calculations.h"
#include "pin_config.h"
// Requires installation of MBED_RPI_PICO_TimerInterrupt library (tools
-> Manage Libraries)
// https://github.com/khoih-prog/MBED_RPI_PICO_TimerInterrupt?tab=
readme-ov-file#use-arduino-library-manager
#include "MBED_RPi_Pico_ISR_Timer.h"
#include "MBED_RPi_Pico_TimerInterrupt.h"

// Period for timer in usec (0.1s)
#define TIMER_PERIOD 100000L

/* variables to track encoder pulses elapsed for turning */
volatile int FL_turn_pulses = 0;
volatile int BL_turn_pulses = 0;
volatile int FR_turn_pulses = 0;
volatile int BR_turn_pulses = 0;

/* variables to track pulses over a period to track RPM */
volatile int FL_speed_pulses = 0;
volatile int BL_speed_pulses = 0;
volatile int FR_speed_pulses = 0;
volatile int BR_speed_pulses = 0;

// FL, BL, FR, BR
float motor_RPM[NUM_MOTORS] = { 0, 0, 0, 0 };

void init_encoders() {

  pinMode(ENCODER_FL, INPUT);
  pinMode(ENCODER_BL, INPUT);
  pinMode(ENCODER_FR, INPUT);
  pinMode(ENCODER_BR, INPUT);

  digitalWrite(ENCODER_FL, INPUT_PULLDOWN);
  digitalWrite(ENCODER_BL, INPUT_PULLDOWN);
  digitalWrite(ENCODER_FR, INPUT_PULLDOWN);
  digitalWrite(ENCODER_BR, INPUT_PULLDOWN);

  /* Attach interrutps to encoder input pins */
  attachInterrupt(digitalPinToInterrupt(ENCODER_FL),
FL_encoder_IRQHandler, RISING);
  attachInterrupt(digitalPinToInterrupt(ENCODER_BL),
BL_encoder_IRQHandler, RISING);
  attachInterrupt(digitalPinToInterrupt(ENCODER_FR),
FR_encoder_IRQHandler, RISING);
  attachInterrupt(digitalPinToInterrupt(ENCODER_BR),
```

```
BR_encoder_IRQHandler, RISING);
}

void init_RPM_timer() {
  MBED_RPI_PICO_Timer ITimer(0);
  // interval (in us), callback is ISR
  ITimer.attachInterruptInterval(TIMER_PERIOD, TimerHandler);
}

void reset_encoders() {
  FL_turn_pulses = 0;
  BL_turn_pulses = 0;
  FR_turn_pulses = 0;
  BR_turn_pulses = 0;
}

void FL_encoder_IRQHandler() {
  FL_turn_pulses++;
  FL_speed_pulses++;
}

void BL_encoder_IRQHandler() {
  BL_turn_pulses++;
  BL_speed_pulses++;
}

void FR_encoder_IRQHandler() {
  FR_turn_pulses++;
  FR_speed_pulses++;
}

void BR_encoder_IRQHandler() {
  BR_turn_pulses++;
  BR_speed_pulses++;
}

float getMotorRPM(int mtr_index) {
  return motor_RPM[mtr_index];
}

bool check_all_motor_RPM(int RPM) {
  for (int i = 1; i < NUM_MOTORS; i++) {
    if (getMotorRPM(i) != RPM) {
      return false;
    }
  }
  return true;
}

int getAvgPulsesLeft() {
  // BL_turn_pulses replaced with FL_turn_pulses since motor encoder is
broken
  return (BL_turn_pulses + BL_turn_pulses) / 2;
}

int getAvgPulsesRight() {
  return (FR_turn_pulses + BR_turn_pulses) / 2;
}

// Never use Serial.print inside this mbed ISR. Will hang the system
void TimerHandler(uint alarm_num) {
  // Always call this for MBED RP2040 before processing ISR
  TIMER_ISR_START(alarm_num);

  motor_RPM[0] = calculate_RPM(FL_speed_pulses);
  motor_RPM[1] = calculate_RPM(BL_speed_pulses);
  motor_RPM[2] = calculate_RPM(FR_speed_pulses);
  motor_RPM[3] = calculate_RPM(BR_speed_pulses);
```

```
  FL_speed_pulses = 0;
  BL_speed_pulses = 0;
  FR_speed_pulses = 0;
  BR_speed_pulses = 0;

  // Always call this for MBED RP2040 after processing ISR
  TIMER_ISR_END(alarm_num);
}
```

robot-demo/pico-motors-development/i2c_setup.h

```
#ifndef I2C_SETUP_H
#define I2C_SETUP_H

void init_i2c();

#define I2C_ADDRESS (0x12)

#define MSB_ANGLE_INDEX (1)
#define LSB_ANGLE_INDEX (2)
#define STATE_INDEX (3)  //Refer to pin_config.h for the numbering

uint32_t get_turning_angle();

void set_turning_angle(uint32_t angle);

#endif
```

robot-demo/pico-motors-development/i2c_setup.ino

```
#include <Wire.h>
#include "i2c_setup.h"
#include "pin_config.h"

const byte PICO_I2C_SDA = 0;
const byte PICO_I2C_SCL = 1;

uint32_t received_angle;

//Initially, point of interest is expected to be directly ahead, so 0
degrees (where right is 90 degrees and left is 270).
uint32_t previous_angle = 0;

//Define
arduino::MbedI2C mywire(PICO_I2C_SDA, PICO_I2C_SCL);

void init_i2c() {
  mywire.begin(I2C_ADDRESS);       // Initialize I2C communication with
address 0x12
  mywire.onReceive(receiveEvent);  // Register receiveEvent function
}

// Variables to store received tuple elements
int received_data[4];

// Function to receive data
void receiveEvent(int bytes) {
  int index = 0;
  while (mywire.available()) {
    // Read received byte
    int c = mywire.read();
    Serial.println(c);
    //Get 3 bytes
    // Interface: First two bytes are the angle, third byte is the
```

```
state we want to be in.
    received_data[index++] = c;
    if (index   4) {
      Serial.println("-------------");
      processMessage(received_data);
      index = 0;
    }
  }
  // Process received data
}

void processMessage(int* data) {
  // Print received tuple elements
  // The angle to the point of interest.

  received_angle = (data[MSB_ANGLE_INDEX] << 8) |
(data[LSB_ANGLE_INDEX]);
  States state = (States)data[STATE_INDEX];
  Serial.print("Recieved Angle: ");
  Serial.println(received_angle);

  Serial.print("Recieved State: ");
  Serial.println(state);

  if (received_angle > 3 && received_angle < 357) {
    // Angle has changed significantly, set state to adjusting.
    if ((getState()   PIVOT_LEFT || getState()   PIVOT_RIGHT) &&
state != STOP) {
      // Do Nothing
    } else {
      Serial.println("changed state to ADJUST ");
      set_turning_angle(received_angle);
      setState(ADJUST);
    }
  } else if (state <= BACKWARD) {
    if ((getState()   PIVOT_LEFT || getState()   PIVOT_RIGHT) &&
state != STOP) {
      Serial.println("currently pivoting");
      // Do Nothing
    } else {
      Serial.print("changed state to  ");
      Serial.println(state);
      setState(state);
    }
  } else {
    Serial.print("Invalid state input: ");
    Serial.println(state);
  }
}

uint32_t get_turning_angle() {
  return received_angle;
}

//FOR DEBUGGING PURPOSES
void set_turning_angle(uint32_t angle) {
  received_angle = angle;
}
```

robot-demo/pico-motors-development/main.ino

```
#include "pin_config.h"
#include "i2c_setup.h"
#include "encoder_interrupts.h"

/*  Problems :
      - Pluses for rotation are not correct in practice, need to
```

```
increase them for more accurate turns

      - for some reason, when we keep getting signals from the pi, the
code breaks where the wheels dont
        turn at all, and seems to constantly go back into the stopped
state (but dont actually as seen in debugging)
        can replicate it by using pi to send signals, and holding down
an arrow key.
*/

bool stateChange = true;

static int turn_pulses = 0;   // var to track pulses during a turn
static int pivot_pulses = 0;  // var to track pulses during a pivot

/* Variables for PID Controller */
static long prevT = 0;
static int vt = 0;  // target velocity

static float velFilt[4] = { 0, 0, 0, 0 };
static float velPrev[4] = { 0, 0, 0, 0 };
static float eintegral[4] = { 0, 0, 0, 0 };
static float uPrevious[4] = { 0, 0, 0, 0 };
static int adjusted_PWMs[4] = { 0, 0, 0, 0 };


/* PID Controller definition, used to regulate operation speed of
motors individually */
void PID_controller() {
  long currT = micros();
  float deltaT = ((float)(currT - prevT)) / 1.0e6;
  prevT = currT;

  //  Apply controller to each motor
  for (int i = 1; i < 4; i++) {
    // Get current velocity in RPM (udpates every second on RPM calc
interrupt)
    noInterrupts();  // for syncronization
    float vel = getMotorRPM(i);
    interrupts();

    // Low-pass filter (25 Hz cutoff)
    velFilt[i] = 0.854 * velFilt[i] + 0.0728 * vel + 0.0728 *
velPrev[i];
    velPrev[i] = vel;

    /* -------- Proportional Component -------- */
    float kp = 0.7;             // proportional constant, needs to be
tuned. increasing provides more power
    float e = vt - velFilt[i];  // error

    /* -------- Integral Component -------- */
    float ki = 2;                                 // Integral constant.
increasing provides more power
    eintegral[i] = eintegral[i] + e * deltaT;  // Update integral with
difference

    float u = (kp * e) + (ki * eintegral[i]);  // Control signal
    uPrevious[i] = u;

    int adjusted_PWM = (int)u;
    // u is positive, meaning we need to speed up
    if (adjusted_PWM > 0) {
      if (adjusted_PWM > 255) adjusted_PWM = 255;
    } else {  // u is negative, meaning we need to slow down
      adjusted_PWM = uPrevious[i] - u;
    }
```

```
    adjusted_PWMs[i] = adjusted_PWM;
  }

  for (int i = 1; i < NUM_MOTORS; i++) {
    set_pwm_duty_cycle(PWM_FWD[i], adjusted_PWMs[i]);
    set_pwm_duty_cycle(PWM_BWD[i], adjusted_PWMs[i]);

    // Encoder for FL motor broken, set to the same speed as BL motor
    if (i   1) {
      set_pwm_duty_cycle(PWM_FWD[i - 1], adjusted_PWMs[i]);
      set_pwm_duty_cycle(PWM_BWD[i - 1], adjusted_PWMs[i]);
    }
  }
}

void setup() {
  Serial.begin(115200);
  init_i2c();
  init_encoders();
  init_RPM_timer();
}

void loop() {

  if (Serial.available() > 0) {
    char command = Serial.read();
    handleSerialCommand(command);
  }

  switch (getState()) {
    case STOP:
      stateChange = false;

      drive_all_motors(0);
      stop_motors();
      break;

    case PIVOT_LEFT:
      stateChange = false;

      pivot_pulses = (pivot_theta(90) * 2);
      pivot_left();
      drive_all_motors(TURNING_SPEED);
      if (getAvgPulsesLeft() < pivot_pulses && getAvgPulsesRight() <
pivot_pulses) {
        // Continue pivoting
      } else {
        setState(FORWARD);
      }
      break;

    case PIVOT_RIGHT:
      stateChange = false;

      pivot_pulses = (pivot_theta(90) * 2);
      pivot_right();
      drive_all_motors(TURNING_SPEED);
      if (getAvgPulsesLeft() < pivot_pulses && getAvgPulsesRight() <
pivot_pulses) {
        // Continue pivoting
      } else {
        setState(FORWARD);
      }
      break;

    case FORWARD:
      if (stateChange) {
        stateChange = false;
```

```
      }
      drive_forwards();
      control_left_motors(MOVING_SPEED_LEFT);
      control_right_motors(MOVING_SPEED_RIGHT);
      break;

    case BACKWARD:
      if (stateChange) {
        stateChange = false;
      }
      drive_backwards();
      control_left_motors(MOVING_SPEED_LEFT);
      control_right_motors(MOVING_SPEED_RIGHT);
      break;

    case ADJUST:
      if (stateChange) {
        stateChange = false;
        control_left_motors(MOVING_SPEED_LEFT);
        control_right_motors(MOVING_SPEED_RIGHT);
        turn_pulses = turn_theta(get_turning_angle());
      }
      // On-the-fly adjustment with received angle.
      // Might need to move this to set state method for integration so
we dont keep reading new updates
      if (fabs(getAvgPulsesLeft() - getAvgPulsesRight()) < turn_pulses)
{
        // Continue turning
      } else {
        setState(FORWARD);
      }
      break;
  }
}

/* Method to set target RPM for PID Controller */
void setTarget(int target) {
  vt = target;
}

void handleSerialCommand(char command) {
  switch (command) {
    case '1':
      setTarget(0);
      break;

    case '2':
      setTarget(60);
      break;

    case '3':
      //Adjust in the right direction 30 degrees.
      Serial.println("SETTING ANGLE");
      set_turning_angle(60);
      break;

    case '4':
      //Adjust in the left direction 30 degrees.
      set_turning_angle(300);
      break;
      // overshoots (30 deg)
    case '5':
      setState(PIVOT_LEFT);
      Serial.println("Set state to PIVOT_LEFT");
      break;

      // undershoots (30)
    case '6':
```

```
      setState(PIVOT_RIGHT);
      Serial.println("Set state to PIVOT_RIGHT");
      break;

    case '7':
      setState(FORWARD);
      drive_forwards();
      Serial.println("Set state to FORWARD");
      break;

    case '8':
      setState(BACKWARD);
      drive_backwards();
      Serial.println("Set state to BACKWARD");
      break;

    case '9':
      setState(ADJUST);
      Serial.println("Set state to ADJUST");
      break;

    case '0':
      setState(STOP);
      Serial.println("Set state to STOP");
      stop_motors();
      break;

    default:
      Serial.println("Invalid command. Available commands: 0, 1, 2, 3,
4, ... 9");
      break;
  }
}
```

robot-demo/pico-motors-development/pin_config.h

```
#ifndef PIN_CONFIG_H
#define PIN_CONFIG_H

typedef enum {
  STOP = 0,
  PIVOT_LEFT,
  PIVOT_RIGHT,
  FORWARD,
  BACKWARD,
  ADJUST
} States;

States state = STOP;

// FWD of the board is USB connection

// PWM-Drive
// Board Left Side Forwards
#define PWM_FWD_FL 6
#define PWM_FWD_BL 7
// Board Left Side Backwards
#define PWM_BWD_FL 5
#define PWM_BWD_BL 13

// Board Right Side Forwards
#define PWM_FWD_FR 27
#define PWM_FWD_BR 26
// Board Right Side Backwards
#define PWM_BWD_FR 28
#define PWM_BWD_BR 18
```

```
// Encoder Pins
// Board Left Side
#define ENCODER_FL (10)
#define ENCODER_BL (11)
// Board Right Side
#define ENCODER_FR (21)
#define ENCODER_BR (20)

#define PWM_FREQ 50
#define NUM_MOTORS 4

const int encoders[4] = { ENCODER_FL, ENCODER_BL, ENCODER_FR,
ENCODER_BR };
const int PWM_FWD[4] = { PWM_FWD_FL, PWM_FWD_BL, PWM_FWD_FR,
PWM_FWD_BR };
const int PWM_BWD[4] = { PWM_BWD_FL, PWM_BWD_BL, PWM_BWD_FR,
PWM_BWD_BR };

// PWM signal to send motors (0-255)
static const int MOVING_SPEED = 100;
static const int MOVING_SPEED_LEFT = 135;
static const int MOVING_SPEED_RIGHT = 100;
static const int TURNING_SPEED = 120;

extern bool stateChange;

void setup_pwm();

void drive_all_motors(uint8_t duty_cycle);

void set_pwm_duty_cycle(unsigned int pwm_pin, unsigned int duty_cycle);

void drive_forwards();

void drive_backwards();

void drive_left();

void drive_right();

void stop_motors();

void pivot_left();

void pivot_right();

int pivot_theta(float angle);

int turn_theta(float angle);

void setState(States newState);

States getState();

#endif
```

robot-demo/pico-motors-development/pin_config.ino

```
#include "pin_config.h"
#include "calculations.h"

#include <cstring>

#define MAX_DISABLED_PINS 4

//Struct to store values of duty cycle.
static struct duty_cycles {
```

```
  int FL, BL;  // Front and Back left
  int FR, BR;  // Front and Back right
} duty_cycles;

static int disabled_gpio_pins[MAX_DISABLED_PINS];

//Temp Function for storing PWM
void check_and_set_pin(unsigned int pwm_pin, unsigned int duty_cycle) {
  switch (pwm_pin) {
    case PWM_BWD_FL:
    case PWM_FWD_FL:
      duty_cycles.FL = duty_cycle;
      break;
    case PWM_BWD_BL:
    case PWM_FWD_BL:
      duty_cycles.BL = duty_cycle;
      break;
    case PWM_BWD_FR:
    case PWM_FWD_FR:
      duty_cycles.FR = duty_cycle;
      break;
    case PWM_BWD_BR:
    case PWM_FWD_BR:
      duty_cycles.BR = duty_cycle;
      break;
  }
}

/* Method to determine if provided PWM pin is disabled for direction
control */
bool check_pin_disabled(unsigned int pin) {
  for (int i = 0; i < MAX_DISABLED_PINS; i++) {
    if (disabled_gpio_pins[i]   pin) {
      return true;
    }
  }
  return false;
}

/* The set PWM of all motors */
void drive_all_motors(uint8_t duty_cycle) {
  for (int i = 0; i < NUM_MOTORS; i++) {
    set_pwm_duty_cycle(PWM_FWD[i], duty_cycle);
    set_pwm_duty_cycle(PWM_BWD[i], duty_cycle);
  }
}

/* This method to contol PWM */
void set_pwm_duty_cycle(unsigned int pwm_pin, unsigned int duty_cycle)
{
  // Check to see if the pin is disabled
  if (check_pin_disabled(pwm_pin)) {
    // Write PWM of 0 to disabled pins to disable them on BTS7960
controller
    analogWrite(pwm_pin, 0);
  } else {
    analogWrite(pwm_pin, duty_cycle);  //set duty cycle to passed in
value
  }
  check_and_set_pin(pwm_pin, duty_cycle);
}

/* Methods to control direction of motors with BTS7960 controller */

void drive_forwards() {
  int pins_to_disable[] = { PWM_BWD_FL, PWM_BWD_BL, PWM_BWD_FR,
PWM_BWD_BR };
  memcpy(disabled_gpio_pins, pins_to_disable, sizeof(pins_to_disable));
```

```
}

void drive_backwards() {
  int pins_to_disable[] = { PWM_FWD_FL, PWM_FWD_BL, PWM_FWD_FR,
PWM_FWD_BR };
  memcpy(disabled_gpio_pins, pins_to_disable, sizeof(pins_to_disable));
}

void pivot_left() {
  int pins_to_disable[] = { PWM_FWD_FL, PWM_FWD_BL, PWM_BWD_FR,
PWM_BWD_BR };
  memcpy(disabled_gpio_pins, pins_to_disable, sizeof(pins_to_disable));
}

void pivot_right() {
  int pins_to_disable[] = { PWM_BWD_FL, PWM_BWD_BL, PWM_FWD_FR,
PWM_FWD_BR };
  memcpy(disabled_gpio_pins, pins_to_disable, sizeof(pins_to_disable));
}

void stop_motors() {
  for (int i = 0; i < 4; i++) {
    analogWrite(PWM_FWD[i], 0);
    analogWrite(PWM_BWD[i], 0);
  }
}

/*  Method to control motors to pivot for a given angle
    returns pulses required for pivot
 */
int pivot_theta(float angle) {
  bool turning_right = false;

  if (angle > 180) {
    turning_right = true;
    angle = 360 - angle;
  }

  if (turning_right) {
    pivot_right();
  } else {
    pivot_left();
  }

  return calculate_pulses_for_angle(angle);
}

/*  Method to control motors to turn a given angle
    returns pulses required for turn
  */
int turn_theta(float angle) {
  bool turning_left = false;

  if (angle > 180) {
    turning_left = true;
    angle = 360 - angle;
  }
  float pulses = calculate_pulses_for_angle(angle);
  float RPM_factor = (pulses / PULSES_PER_REV) * 3;

  if (turning_left) {
    control_right_motors(duty_cycles.FR + (duty_cycles.FR *
RPM_factor));
  } else {
    control_left_motors(duty_cycles.FL + (duty_cycles.FL *
RPM_factor));
  }
  return pulses;
```

```
}

void control_left_motors(float duty_cycle) {
  set_pwm_duty_cycle(PWM_FWD_FL, duty_cycle);
  set_pwm_duty_cycle(PWM_FWD_BL, duty_cycle);
}

void control_right_motors(float duty_cycle) {
  set_pwm_duty_cycle(PWM_FWD_FR, duty_cycle);
  set_pwm_duty_cycle(PWM_FWD_BR, duty_cycle);
}

// Getter function for the 'state' variable
States getState() {
  return state;
}

// Setter function for the 'state' variable
void setState(States newState) {
  // reset the encoders tracking turning on every state change
  reset_encoders();
  drive_all_motors(MOVING_SPEED);
  stateChange = true;
  state = newState;
}
```

robot-demo/ultra_simple/Makefile

```
#/*
# * Copyright (C) 2014  RoboPeak
# * Copyright (C) 2014 - 2018 Shanghai Slamtec Co., Ltd.
# *
# * This program is free software: you can redistribute it and/or
modify
# * it under the terms of the GNU General Public License as published
by
# * the Free Software Foundation, either version 3 of the License, or
# * (at your option) any later version.
# *
# * This program is distributed in the hope that it will be useful,
# * but WITHOUT ANY WARRANTY; without even the implied warranty of
# * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# * GNU General Public License for more details.
# *
# * You should have received a copy of the GNU General Public License
# * along with this program.  If not, see <http://www.gnu.org/licenses/
>.
# *
# */
#
HOME_TREE := ../../

MODULE_NAME := $(notdir $(CURDIR))

include $(HOME_TREE)/mak_def.inc

CXXSRC += main.cpp
CXXSRC += socket.cpp

C_INCLUDES += -I$(CURDIR)/../../sdk/include -I$(CURDIR)/../../sdk/src -
I$(CURDIR)

EXTRA_OBJ :=
LD_LIBS += -lstdc++ -lpthread

all: build_app

include $(HOME_TREE)/mak_common.inc

clean: clean_app
```

robot-demo/ultra_simple/main.cpp

```
++
/*
 *  SLAMTEC LIDAR
 *  Ultra Simple Data Grabber Demo App
 *
 *  Copyright (c) 2009 - 2014 RoboPeak Team
 *  http://www.robopeak.com
 *  Copyright (c) 2014 - 2020 Shanghai Slamtec Co., Ltd.
 *  http://www.slamtec.com
 *
 */
/*
 * This program is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program.  If not, see <http://www.gnu.org/licenses/
>.
 *
 */

#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
#include <string.h>

#include "sl_lidar.h"
#include "sl_lidar_driver.h"

//Socket Implementation
#include <iostream>
#include "socket.hpp"
using namespace std;


#ifndef _countof
#define _countof(_Array) (int)(sizeof(_Array) / sizeof(_Array[0]))
#endif

#ifdef _WIN32
#include <Windows.h>
#define delay(x)   ::Sleep(x)
#else
#include <unistd.h>
static inline void delay(sl_word_size_t ms){
    while (ms>=1000){
        usleep(1000*1000);
        ms-=1000;
    };
    if (ms!=0)
        usleep(ms*1000);
}
#endif

using namespace sl;

void print_usage(int argc, const char * argv[])
{
```

```
    printf("Usage:\n"
           " For serial channel\n %s --channel --serial <com port>
[baudrate]\n"
           " The baudrate used by different models is as follows:\n"
           "  A1(115200),A2M7(256000),A2M8(115200),A2M12(256000),"
           "A3(256000),S1(256000),S2(1000000),S3(1000000)\n"
       " For udp channel\n %s --channel --udp <ipaddr> [port NO.]\n"
           " The T1 default ipaddr is 192.168.11.2,and the port NO.is
8089. Please refer to the datasheet for details.\n"
           , argv[0], argv[0]);
}

bool checkSLAMTECLIDARHealth(ILidarDriver * drv)
{
    sl_result     op_result;
    sl_lidar_response_device_health_t healthinfo;

    op_result = drv->getHealth(healthinfo);
    if (SL_IS_OK(op_result)) { // the macro IS_OK is the preperred way
to judge whether the operation is succeed.
        printf("SLAMTEC Lidar health status : %d\n",
healthinfo.status);
        if (healthinfo.status   SL_LIDAR_STATUS_ERROR) {
            fprintf(stderr, "Error, slamtec lidar internal error
detected. Please reboot the device to retry.\n");
            // enable the following code if you want slamtec lidar to
be reboot by software
            // drv->reset();
            return false;
        } else {
            return true;
        }

    } else {
        fprintf(stderr, "Error, cannot retrieve the lidar health code:
%x\n", op_result);
        return false;
    }
}

bool ctrl_c_pressed;
void ctrlc(int)
{
    ctrl_c_pressed = true;
}

int main(int argc, const char * argv[]) {
    ClientSocket python_socket = createConnection("127.0.0.1",
std::getenv("SDK_PORT"));


  const char * opt_is_channel = NULL;
  const char * opt_channel = NULL;
    const char * opt_channel_param_first = NULL;
  sl_u32         opt_channel_param_second = 0;
    sl_u32         baudrateArray[2] = {115200, 256000};
    sl_result     op_result;
  int          opt_channel_type = CHANNEL_TYPE_SERIALPORT;

  bool useArgcBaudrate = false;

    IChannel* _channel;

    printf("Ultra simple LIDAR data grabber for SLAMTEC LIDAR.\n"
           "Version: %s\n", SL_LIDAR_SDK_VERSION);


  if (argc>1)
```

```
  {
    opt_is_channel = argv[1];
  }
  else
  {
    print_usage(argc, argv);
    return -1;
  }

  if(strcmp(opt_is_channel, "--channel") 0){
    opt_channel = argv[2];
    if(strcmp(opt_channel, "-s") 0||strcmp(opt_channel, "--serial") 0)
    {
      // read serial port from the command line...
      opt_channel_param_first = argv[3];// or set to a fixed value:
e.g. "com3"
      // read baud rate from the command line if specified...
      if (argc>4) opt_channel_param_second = strtoul(argv[4], NULL,
10);
      useArgcBaudrate = true;
    }
    else if(strcmp(opt_channel, "-u") 0||strcmp(opt_channel, "--udp")
0)
    {
      // read ip addr from the command line...
      opt_channel_param_first = argv[3];//or set to a fixed value: e.g.
"192.168.11.2"
      if (argc>4) opt_channel_param_second = strtoul(argv[4], NULL,
10);//e.g. "8089"
      opt_channel_type = CHANNEL_TYPE_UDP;
    }
    else
    {
      print_usage(argc, argv);
      return -1;
    }
  }
  else
  {
    print_usage(argc, argv);
        return -1;
  }

  if(opt_channel_type   CHANNEL_TYPE_SERIALPORT)
  {
    if (!opt_channel_param_first) {
#ifdef _WIN32
    // use default com port
    opt_channel_param_first = "\\\\.\\com3";
#elif __APPLE__
    opt_channel_param_first = "/dev/tty.SLAB_USBtoUART";
#else
    opt_channel_param_first = "/dev/ttyUSB0";
#endif
    }
  }


    // create the driver instance
  ILidarDriver * drv = *createLidarDriver();

    if (!drv) {
        fprintf(stderr, "insufficent memory, exit\n");
        exit(-2);
    }

    sl_lidar_response_device_info_t devinfo;
    bool connectSuccess = false;
```

```
    if(opt_channel_type   CHANNEL_TYPE_SERIALPORT){
        if(useArgcBaudrate){
            _channel =
(*createSerialPortChannel(opt_channel_param_first,
opt_channel_param_second));
            if (SL_IS_OK((drv)->connect(_channel))) {
                op_result = drv->getDeviceInfo(devinfo);

                if (SL_IS_OK(op_result))
                {
                  connectSuccess = true;
                }
                else{
                    delete drv;
          drv = NULL;
                }
            }
        }
        else{
            size_t baudRateArraySize = (sizeof(baudrateArray))/
(sizeof(baudrateArray[0]));
      for(size_t i = 0; i < baudRateArraySize; ++i)
      {
        _channel = (*createSerialPortChannel(opt_channel_param_first,
baudrateArray[i]));
                if (SL_IS_OK((drv)->connect(_channel))) {
                    op_result = drv->getDeviceInfo(devinfo);

                    if (SL_IS_OK(op_result))
                    {
                      connectSuccess = true;
                        break;
                    }
                    else{
                        delete drv;
              drv = NULL;
                    }
                }
      }
        }
    }
    else if(opt_channel_type   CHANNEL_TYPE_UDP){
        _channel = *createUdpChannel(opt_channel_param_first,
opt_channel_param_second);
        if (SL_IS_OK((drv)->connect(_channel))) {
            op_result = drv->getDeviceInfo(devinfo);

            if (SL_IS_OK(op_result))
            {
              connectSuccess = true;
            }
            else{
                delete drv;
        drv = NULL;
            }
        }
    }


    if (!connectSuccess) {
        (opt_channel_type   CHANNEL_TYPE_SERIALPORT)?
      (fprintf(stderr, "Error, cannot bind to the specified serial port
%s.\n"
        , opt_channel_param_first)):(fprintf(stderr, "Error, cannot
connect to the specified ip addr %s.\n"
        , opt_channel_param_first));
```

```
        goto on_finished;
    }

    // print out the device serial number, firmware and hardware
version number..
    printf("SLAMTEC LIDAR S/N: ");
    for (int pos = 0; pos < 16 ;++pos) {
        printf("%02X", devinfo.serialnum[pos]);
    }

    printf("\n"
            "Firmware Ver: %d.%02d\n"
            "Hardware Rev: %d\n"
            , devinfo.firmware_version>>8
            , devinfo.firmware_version & 0xFF
            , (int)devinfo.hardware_version);



    // check health...
    if (!checkSLAMTECLIDARHealth(drv)) {
        goto on_finished;
    }

    signal(SIGINT, ctrlc);

  if(opt_channel_type   CHANNEL_TYPE_SERIALPORT)
        drv->setMotorSpeed();
    // start scan...
    drv->startScan(0,1);

    // fetech result and print it out...
    while (1) {
        sl_lidar_response_measurement_node_hq_t nodes[8192];
        size_t   count = _countof(nodes);

        op_result = drv->grabScanDataHq(nodes, count);

        if (SL_IS_OK(op_result)) {
            drv->ascendScanData(nodes, count);
            for (int pos = 0; pos < (int)count ; ++pos) {
    printf("%s theta: %03.2f Dist: %08.2f Q: %d \n",
                    (nodes[pos].flag &
SL_LIDAR_RESP_HQ_FLAG_SYNCBIT) ?"S ":"  ",
                    (nodes[pos].angle_z_q14 * 90.f) / 16384.f,
                    nodes[pos].dist_mm_q2/4.0f,
                    nodes[pos].quality >>
SL_LIDAR_RESP_MEASUREMENT_QUALITY_SHIFT);
    int quality = nodes[pos].quality >>
SL_LIDAR_RESP_MEASUREMENT_QUALITY_SHIFT;
                if(quality > 0) {
                    struct values datapoint;
                    datapoint.degrees = (nodes[pos].angle_z_q14 *
90.f) / 16384.f;
                    datapoint.distance = nodes[pos].dist_mm_q2/4.0f;
                    sendValues(&python_socket, datapoint);

        printf("%s theta: %03.2f Dist: %08.2f Q: %d \n",
                    (nodes[pos].flag &
SL_LIDAR_RESP_HQ_FLAG_SYNCBIT) ?"S ":"  ",
                    (nodes[pos].angle_z_q14 * 90.f) / 16384.f,
                    nodes[pos].dist_mm_q2/4.0f,
                    nodes[pos].quality >>
SL_LIDAR_RESP_MEASUREMENT_QUALITY_SHIFT);


                }
            }
```

```
        }

        if (ctrl_c_pressed){
            break;
        }
    }

    drv->stop();
  delay(200);
  if(opt_channel_type   CHANNEL_TYPE_SERIALPORT)
        drv->setMotorSpeed(0);
    // done!
on_finished:
    if(drv) {
        delete drv;
        drv = NULL;
    }
    closeConnection(python_socket);
    return 0;
}
```

robot-demo/ultra_simple/socket.cpp

```
++
//
//  socket.cpp
//


#include "socket.hpp"

using namespace std;

ClientSocket createConnection(char* ip, char* port) {
    ClientSocket sock;

    /* Create a socket point */
    sock.sockfd = socket(AF_INET, SOCK_STREAM, 0);

    if (sock.sockfd < 0) {
        perror("ERROR opening socket");
        exit(1);
    } else if (sock.sockfd > 0) {
        cout << "SOCKET OPENED" << endl;
    }

    sock.serv_addr.sin_family = AF_INET;
    sock.serv_addr.sin_port = htons(atoi(port));
    inet_pton(AF_INET, ip, &(sock.serv_addr.sin_addr.s_addr));

    cout << "attempting to connect to server" << endl;

    int conn_success = connect(sock.sockfd, (struct
sockaddr*)&sock.serv_addr, sizeof(sock.serv_addr));

    if (conn_success < 0) {
        cout << "ERROR connecting to " << ip << ":" << port << endl;
        exit(1);
    } else {
        cout << "connection successful to " << ip << ":" << port <<
endl;
    }

    return sock;
}

int sendValues(ClientSocket* sock, struct values a) {
    int n = send(sock->sockfd, &a, sizeof(a), 0);
    return 0;
}

void closeConnection(ClientSocket sock) {
    int n = close(sock.sockfd);
    if (n   0) {
        printf("Socket has been shutdown\n");
    }
}
```

robot-demo/ultra_simple/socket.hpp

```
//
//  socket.hpp
//


#ifndef socket_hpp
#define socket_hpp

#include <stdio.h>
#include <iostream>
#include <cstring>
#include <arpa/inet.h>
#include <sys/socket.h>
#include <unistd.h>

struct values {
    float degrees;
    float distance;
};


struct ClientSocket {
    int sockfd;
    struct sockaddr_in serv_addr;
};


ClientSocket createConnection(char* ip, char* port);
int sendValues(ClientSocket* sock, struct values a);
void closeConnection(ClientSocket sock);


#endif /* socket_hpp */
```

sound-demo/pico_play_noise/sound_transmit.cpp

```
++
#include "sound_transmit.h"
#include <I2S.h>
//#include "src/sound_samples/sound_samplesA.h"
#include "src/sound_samples/sound_samplesB.h"
//#include "src/sound_samples/sound_samplesC.h"

I2S i2s_out(OUTPUT);
const int sampleRate = 50000;

size_t sound_out_ind = SOUND_SAMPLES_LEN; // silence at first
uint64_t last_interrupt_ticks = 0;        // Number of CPU cycles since
the last "onTransmit" interrupt
uint32_t samples_to_skip = 0;             // amount of samples to skip
before sending samples
int32_t divider = 1;                      // makes sound less loud

void set_sound_divider(int div)
{
  divider = div;
}

void onTransmit()
{
  last_interrupt_ticks = rp2040.getCycleCount64();
  while (true)
  {
    int32_t sample;
    if (sound_out_ind < SOUND_SAMPLES_LEN && samples_to_skip   0)
    {
      // sample = sound | sound << 16; // 16 bits for L channel, 16
bits for R channel
      sample = sound_samples[sound_out_ind] << 16; // sample is 32-bit
audio, shift 16 bit sound to most significant bits of 32-bit sample
      sample /= divider;                          // make sound less
loud
    }
    else
    {
      sample = 0;
    }

    int write_succesful = i2s_out.write(sample, false); // L channel
    if (!write_succesful)
    {
      break;
    }
    i2s_out.write(sample, false); // R channel

    if (sound_out_ind < SOUND_SAMPLES_LEN && samples_to_skip   0)
    {
      sound_out_ind += 1;
      sound_out_ind %= SOUND_SAMPLES_LEN; // Repeat sound when done
    }
    else if (samples_to_skip > 0)
    {
      samples_to_skip -= 1;
    }
  }
}

void setup_i2s_sound_out()
{
  i2s_out.setBCLK(20); // WS / LRCLK has to be 21, BCLK+1
  i2s_out.setDATA(22);
```

```
  i2s_out.setBitsPerSample(32);
  i2s_out.onTransmit(onTransmit);
  i2s_out.setBuffers(3, 0, 0); // 3 buffers (minimum), 0 buffer length
(it will set the default), 0 silence sample

  if (!i2s_out.begin(sampleRate))
  {
    Serial.println("Failed to initialize I2S out!");
    while (1)
      ; // do nothing
  }
}
void start_transmitting_sound()
{
  sound_out_ind = 0;
  uint64_t tick_diff = rp2040.getCycleCount64() - last_interrupt_ticks;
  // Rationale:
  // With the DMA triple-buffering, we always submit a certain amount
of samples ahead of time
  // Start transmitting audio after having skipped as many future
samples submission as we are currently into the buffer
  // Ex: (s means sample already submitted, e is not submitted, [] is a
buffer)
  // [s s s s] [s s s s] [s s s s] [e e e e]
  //    ^                           ^
  //    cur time                    start transmitting here
  // want to start transmitting at cur time, but empty samples (0s)
already submitted for the next few buffer.
  // In the next available buffer, delay sound transmission by 2
samples
  // This is for consistency, this way we always transmit x samples
late (x=numbuffers*numsamplesinbuffer).
  float skip = ((float)tick_diff * ((float)sampleRate /
(float)rp2040.f_cpu()));
  samples_to_skip = (uint32_t)skip;
  // Serial.printf("delaying by %i samples\n", samples_to_skip);
}

void stop_transmitting_sound()
{
  sound_out_ind = SOUND_SAMPLES_LEN; // silence at first
}

/* bool is_done_transmitting_sound()
{
  return sound_out_ind >= SOUND_SAMPLES_LEN;
} */
```

sound-demo/pico_play_noise/sound_transmit.h

```
void setup_i2s_sound_out();
void set_sound_divider(int divider);
void start_transmitting_sound();
void stop_transmitting_sound();
//bool is_done_transmitting_sound();
```

sound-demo/pico_play_noise/transceiver_speaker.cpp

```
#include <RFM69.h>
#include <SPI.h>
#include "transceiver_speaker.h"
#include "sound_transmit.h"

RFM69 radio = RFM69(CS_PIN, INTRPT_PIN);
String receivedData = "";
```

```
unsigned long clockAdjust = 0;

void setupRfm69() {
  bool init_ant_success = radio.initialize(FREQUENCY, MYNODEID,
NETWORKID);
  if (init_ant_success) {
    Serial.printf("Node %d ready\n", MYNODEID);
  } else {
    Serial.printf("Failed to initialize Antenna Node %d\n", MYNODEID);
    while (1); // do nothing
  }
  // Set bit rate to 250kbps
  radio.writeReg(0x03,0x00);
  radio.writeReg(0x04,0x80);
  radio.setHighPower(); // Always use this for RFM69HCW
}

void sendPing(int target, char* msg) {
  Serial.printf("Sending ping to node %d\n", TONODEID);
  if (USEACK) {
    bool success = radio.sendWithRetry(target, msg, strlen(msg), 0,
2); //last 2 arguments: 0 retries, 2ms wait time for ACK
     if (success) {
         Serial.printf("ACK received!\n");
     } else {
         Serial.printf("no ACK received :(\n");
     }
  } else {
    // Serial.printf("Sending without ack\n");
    // radio.send(TONODEID, "g", 1);
    // Serial.printf("Sent!\n");
  }
}

void receivePing() {
  // Serial.println("Waiting for ping from receiver (microphone)...");
  while(!radio.receiveDone()){
    // do nothing
  }

  Serial.print("Ping received from node ");
  Serial.print(radio.SENDERID, DEC);
  Serial.print(": [");

  for (byte i = 0; i < radio.DATALEN; i++) {
    Serial.printf("%c", (char)radio.DATA[i]);
    receivedData += (char)radio.DATA[i];
  }
  Serial.println("]");

  // Send an ACK if requested.
  if (radio.ACKRequested()) {
    radio.sendACK();
    Serial.printf("ACK sent\n");
  }
}

void checkBroadcast() {
  int rcvMsg = 0;

  receivePing();
  // Convert received data to unsigned long int
  char rcvbuf[receivedData.length() + 1];
  receivedData.toCharArray(rcvbuf, receivedData.length() + 1);
  rcvbuf[receivedData.length()] = '\0';
  Serial.printf("Received message %s\n", rcvbuf);
//  rcvMsg = strtoul(rcvbuf, NULL, 10);
//  Serial.printf("Received message %lu\n", rcvMsg);
```

```
  Serial.println(rcvbuf);

  if (isDigit(*rcvbuf)) {     // is the buffer recevied a digit?
      rcvMsg = atoi(rcvbuf);  // Convert string to integer
      Serial.println(rcvMsg);
      if (rcvMsg   0) {
        stop_transmitting_sound();
      }
      else {
        set_sound_divider(rcvMsg);
        start_transmitting_sound();
      }
  }
  else {
      Serial.println("INVALID MESSAGE! Message must be an integer.");
  }
  receivedData = "";
}
```

sound-demo/pico_play_noise/transceiver_speaker.h

```
#ifndef TRANSCEIVER_SPEAKER_H
#define TRANSCEIVER_SPEAKER_H

#define NETWORKID     0   // Must be the same for all nodes (0 to 255)
#define MYNODEID      2   // My node ID (0 to 255)
#define TONODEID      100   // Destination node ID (0 to 254, 255 =
broadcast)
#define FREQUENCY     RF69_915MHZ
#define ENCRYPT       false // Set to "true" to use encryption
#define ENCRYPTKEY    ""    // Use the same 16-byte key on all nodes
#define USEACK        false  // Request ACKs or not
#define CS_PIN        17    // in PICO
#define INTRPT_PIN    2     // in PICO

void setupRfm69();
void sendPing();
void receivePing();
void checkBroadcast();
#endif
```

sound-demo/pico_play_noise/pico_play_noise.ino

```
#include "sound_transmit.h"
#include "transceiver_speaker.h"

void setup() {
  pinMode(LED_BUILTIN, OUTPUT);
  rp2040.enableDoubleResetBootloader();

  Serial.begin(115200); // Changing this doesn't actually change the
serial speed
  delay(3000); // delay long enough for serial to get set up
  digitalWrite(LED_BUILTIN, 1);

  setupRfm69();
  setup_i2s_sound_out();
  checkBroadcast();
}

void loop() {
  checkBroadcast();
}
```

sound-demo/pico_record_to_serial/pico_record_to_serial.ino

```
#include "record_sound.h"
#include <cstdint>
#include <cstddef>
#include <algorithm>

const int buttonPin = 14; // For draw button
const int clearPin = 15; // For clear button
int buttonState1 = 0; // For draw button
int buttonState2 = 0; //For clear button

void setup()
{
  pinMode(LED_BUILTIN, OUTPUT);
  pinMode(buttonPin, INPUT_PULLUP); // Set up draw button as input
  pinMode(clearPin, INPUT_PULLUP); // Set up clear button as input

  Serial.begin(115200); // Changing this doesn't actually change the
serial speed
  digitalWrite(LED_BUILTIN, 1);
  while (!Serial)
    ; // delay long enough for serial to get set up
  String str = Serial.readStringUntil('\n');
  str.trim();
  if (str != "freq") {
    Serial.println("did not get freq");
    while (true);
  }
  int fs = Serial.readStringUntil('\n').toInt();
  fs = std::clamp(fs, 8000, 50000);

  setup_sound_in(fs);
}

bool first = true;
size_t buffer_ind = 0;
byte buffer[64];

void loop()
{

  if (Serial.available() > 0)
  {
    String str = Serial.readStringUntil('\n');
    str.trim();
    if (str   "freq") {
      String _freq = Serial.readStringUntil('\n'); // skip freq reading
      return;
    }

    // For draw button
    buttonState1 = digitalRead(buttonPin);
    if (buttonState1   LOW){
      Serial.println("press");
    }
    else{
      Serial.println("yyyyy");
    }

    //For clear button
    buttonState2 = digitalRead(clearPin);
    if (buttonState2   LOW){
      Serial.println("cleared");
    }
    else{
      Serial.println("xxxxxxx");
```

```
    }
    int num_samples = str.toInt();
    for (int i = 0; i < num_samples; i++)
    {
      int16_t sample = get_sample();
      buffer[buffer_ind] = (sample >> 0) & 0xFF;
      buffer_ind++;
      buffer[buffer_ind] = (sample >> 8) & 0xFF;
      buffer_ind++;
      // buffer[buffer_ind] = (sample >> 16) & 0xFF;
      // buffer_ind++;
      // buffer[buffer_ind] = (sample >> 24) & 0xFF;
      // buffer_ind++;
      if (buffer_ind   64)
      {
        Serial.write(buffer, 64);
        buffer_ind = 0;
      }
    }
    if (buffer_ind > 0)
    {
      Serial.write(buffer, buffer_ind);
      buffer_ind = 0;
    }
    Serial.flush();
  }
  else
  {
    int16_t discarded_sample = get_sample();
    (void)discarded_sample;
  }
}
```

sound-demo/pico_record_to_serial/record_sound.cpp

```
#include <I2S.h>
#include "record_sound.h"
#include <cstdint>

I2S i2s_in(INPUT);

void setup_sound_in(int32_t fs)
{
  i2s_in.setBCLK(12); // WS / LRCLK has to be 13, BCLK+1
  i2s_in.setDATA(11);
  i2s_in.setBitsPerSample(32);
  i2s_in.setBuffers(18, 0, 0); // 3 buffers (minimum), 0 buffer length
(it will set the default), 0 silence sample

  if (!i2s_in.begin(fs))
  {
    Serial.println("Failed to initialize I2S in!");
    while (1)
      ; // do nothing
  }
}

int16_t get_sample()
{
  int32_t l, r;
  i2s_in.read32(&l, &r);
  int16_t good_bits = ((l << 4) & 0xFFFF0000) >> 16;
  return good_bits;
}

int32_t get_full_sample()
{
```

```
  int32_t l, r;
  i2s_in.read32(&l, &r);
  return l;
}
```

sound-demo/pico_record_to_serial/record_sound.h

```
void setup_sound_in(int32_t fs);
int16_t get_sample();
int32_t get_full_sample();
```

sound-demo/pico_speaker_synchronizer/pico_speaker_synchronizer.ino

```
#include "transceiver_speaker_master.h"
#include "sound_transmit.h"

void setup() {
  pinMode(LED_BUILTIN, OUTPUT);
  rp2040.enableDoubleResetBootloader();

  Serial.begin(115200); // Changing this doesn't actually change the
serial speed
  delay(3000); // delay long enough for serial to get set up
  digitalWrite(LED_BUILTIN, 1);

//  Serial.printf("Started! Clock speed: %i\n", rp2040.f_cpu());
  setupRfm69();
  setup_i2s_sound_out();

  String initial_div = "128";
  sendBroadcast(initial_div);
}

void loop() {
  if (Serial.available() > 0){
    String in = Serial.readStringUntil('\n');
    in.trim();

    sendBroadcast(in);
  }
}
```

sound-demo/pico_speaker_synchronizer/transceiver_speaker_master.cpp

```
#include <RFM69.h>
#include <SPI.h>
#include "transceiver_speaker_master.h"

RFM69 radio = RFM69(CS_PIN, INTRPT_PIN);
String receivedData = "";

void setupRfm69() {
  bool initAntennaSuccess = radio.initialize(FREQUENCY, MYNODEID,
NETWORKID);
  if (initAntennaSuccess) {
    Serial.printf("Node %d ready\n", MYNODEID);
  } else {
    Serial.printf("Failed to initialize Antenna Node %d\n", MYNODEID);
    while (1); // do nothing
  }
  // Set bit rate to 250kbps
  radio.writeReg(0x03, 0x00);
  radio.writeReg(0x04, 0x80);
  radio.setHighPower(); // Always use this for RFM69HCW
```

```
}

void sendPing(int target, char* msg) {
  Serial.printf("Sending ping to node %d\n", target);
//  Serial.printf("size of message: %d", strlen(msg));
  if (USEACK) {
    bool success = radio.sendWithRetry(target, msg, strlen(msg), 0,
5); //last 2 arguments: 0 retries, 2ms wait time for ACK
    if (success) {
      Serial.printf("ACK received!\n");
    } else {
      Serial.printf("no ACK received :(\n");
      //        sendPing(target);
    }
  } else {
    //Serial.printf("Sending without ack\n");
    // radio.send(target, "m", 1); //send character "m" (length 1) to
TONODEID
    //Serial.printf("Sent!\n");
  }
}

void receivePing() {
  // Serial.println("Waiting for ping from receiver (microphone)...");
  while(!radio.receiveDone()){
    // do nothing
  }

  Serial.print("Ping received from node ");
  Serial.print(radio.SENDERID, DEC);
  Serial.print(": [");

//  Serial.println(radio.DATALEN);
  for (byte i = 0; i < radio.DATALEN; i++) {
    Serial.printf("%c", (char)radio.DATA[i]);
    receivedData += (char)radio.DATA[i];
  }
  Serial.println("]");

  // Send an ACK if requested.
  if (radio.ACKRequested()) {
    radio.sendACK();
    Serial.printf("ACK sent\n");
  }
}

void sendBroadcast(String in) {
  char buf[60];
  strcpy(buf, in.c_str());
  Serial.println(buf);
  sendPing(TOALLNODE, buf);
}
```

sound-demo/pico_speaker_synchronizer/transceiver_speaker_master.h

```
#ifndef TRANSCEIVER_SPEAKER_H
#define TRANSCEIVER_SPEAKER_H

#include <RFM69.h>

#define NETWORKID     0   // Must be the same for all nodes (0 to 255)
#define MYNODEID      100   // My node ID (0 to 255)
#define TOALLNODE     0   // Destination node ID (0 to 254, 0 =
broadcast)
#define FREQUENCY     RF69_915MHZ
#define USEACK        true // Request ACKs or not
#define CS_PIN        17    // in PICO
```

```
#define INTRPT_PIN    2     // in PICO

void setupRfm69();
void sendPing(int target, char* msg);
void receivePing();
void sendBroadcast(String in);
#endif
```

sound-demo/standalone_mic_TDoA_simpler.py

```
import numpy as np
import scipy as sp
import matplotlib.pyplot as plt
import matplotlib.animation as pltAnim
import serial
import threading
import queue
import copy

noiseA = np.load("misc/filtered_noiseA.npy")
noiseB = np.load("misc/filtered_noiseB.npy")
noiseC = np.load("misc/filtered_noiseC.npy")

A = np.array([0, 0])
B = np.array([0.82, 0])
C = np.array([0.82 / 2, 0.551])
Z_MIC_RELATIVE_TO_SPEAKERS = 0.195

assert A[0]   0 and A[1]   0
assert B[0] > 0 and B[1]   0
assert C[1] > 0


def correlate_and_find_delay(rec, noise, name):
    Nw = len(noise)
    assert Nw   len(rec)
    rec_fft = np.fft.rfft(rec)
    noise_fft_conj = np.conj(np.fft.rfft(noise))
    # print(rec_fft.shape, noise_fft_conj.shape)
    cross_corr_freq = noise_fft_conj * rec_fft
    cross_corr = np.abs(np.fft.irfft(cross_corr_freq))

    # valid_len = diff + 1
    # cross_corr = cross_corr[:valid_len]

    # print(cross_corr)
    k_max_ind = np.argmax(cross_corr)
    k_max = cross_corr[k_max_ind]
    avg = np.sum(cross_corr) / len(cross_corr)

    return k_max_ind, k_max, avg, cross_corr


def fangs_algorithm_TDoA(ta, tb, tc):
    Nw = len(noiseA)
    # The correlation wraps around, this can cause issues when 1 val is
close to index 0, and the other close to Nw
    # assume the tdoa is always less than Nw/2, if not then substract 1
val Nw, this should bring substraction back in the range
    # substracting the largest value by Nw makes it negative
    # tab = min([ta-tb, ta-tb-Nw, ta-tb+Nw], key=lambda x: abs(x))
    # tac = min([ta-tc, ta-tc-Nw, ta-tc+Nw], key=lambda x: abs(x))

    ## Fang's algorithm, gotten from the PDF (in the repo or at
https://ieeexplore.ieee.org/document/102710)
    c = 343  # speed of wave in medium, speed of sound=343 m/s
```

```
    cTa = ta * c
    cTb = tb * c
    cTc = tc * c

    b = B[0]  # also = np.linalg.norm(B)
    cx = C[0]
    cy = C[1]
    c = np.linalg.norm(C)

    Rab = cTa - cTb
    Rac = cTa - cTc
    # Rab = c*tab
    # Rac = c*tac

    # avoid division by zero, just make em real small
    if Rab   0.0:
        Rab = 1e-5
    if Rac   0.0:
        Rac = 1e-5

    # variable names correspond to those in the paper

    g = (Rac * b / Rab - cx) / cy
    h = (c**2 - Rac**2 + Rac * Rab * (1 - (b / Rab) ** 2)) / (2 * cy)

    d = -(1 - (b / Rab) ** 2 + g**2)
    e = b * (1 - (b / Rab) ** 2) - 2 * g * h
    f = (Rab**2 / 4) * (1 - (b / Rab) ** 2) ** 2 - h**2

    z = Z_MIC_RELATIVE_TO_SPEAKERS
    x = np.roots([d, e, f - z**2])  # eq 9a
    x = x[
        (abs(x.imag) < 1e-5) & (x.real >= A[0]) & (x.real <= B[0])
    ]  # ignore imaginary roots and points beyond axis limits
    y = g * x + h  # eq 13
    # print(x, y)

    guesses = np.transpose([x, y])

    # print(guesses)

    def err(g):
        # calculate what sort of time deltas would be seen with this
guess
        # compare them to the original, return the MSE
        deltaA = np.linalg.norm(g - A)
        deltaB = np.linalg.norm(g - B)
        deltaC = np.linalg.norm(g - C)
        rab = deltaA - deltaB
        rac = deltaA - deltaC
        mse = ((rab - Rab) ** 2 + (rac - Rac) ** 2) / 2
        return mse

    if len(guesses)   0:
        return None

    best_guess = min(guesses, key=err)
    return best_guess


def plot_spect(sound):
    f, t, Sxx = sp.signal.spectrogram(sound, 50000)
    plt.pcolormesh(t, f, Sxx, shading="gouraud")
    plt.ylabel("Frequency [Hz]")
    plt.xlabel("Time [sec]")
    plt.ylim((1000, 5000))
```

```
INVALID_CURSOR_POS = (-100.0, -100.0)  # off the shown map hopefully

thread_queue = queue.Queue()


class QueueMsg:
    def __init__(
        self, cross_corrA, cross_corrB, cross_corrC, positions,
cursor_position
    ) -> None:
        self.cross_corrA = cross_corrA
        self.cross_corrB = cross_corrB
        self.cross_corrC = cross_corrC
        self.positions = positions
        self.cursor_position = cursor_position


def do_plot():
    fig = plt.figure(1)
    axA = plt.subplot(231)
    axB = plt.subplot(232, sharey=axA)
    axC = plt.subplot(233, sharey=axA)

    axMap = plt.subplot(212)

    def update_plot(frame):
        if thread_queue.empty():
            return
        msg = thread_queue.get()

        axA.clear()
        axA.plot(msg.cross_corrA)
        axA.set_title("Correlation A")

        axB.clear()
        axB.plot(msg.cross_corrB)
        axB.set_title("Correlation B")
        axB.tick_params("y", labelleft=False)

        axC.clear()
        axC.plot(msg.cross_corrC)
        axC.set_title("Correlation C")
        axC.tick_params("y", labelleft=False)

        axMap.clear()
        axMap.scatter(
            [A[0], B[0], C[0]],
            [A[1], B[1], C[1]],
            label="base stations",
            marker="o",
            color="purple",
        )
        axMap.set_aspect("equal")
        axMap.set_xlim(axMap.get_xlim())
        axMap.set_ylim(axMap.get_ylim())

        for line in msg.positions:
            axMap.plot(line[0], line[1], color="blue", linewidth="2")

        axMap.scatter(
            msg.cursor_position[0],
            msg.cursor_position[1],
            color="lightblue",
        )
        fig.canvas.draw()

    anim = pltAnim.FuncAnimation(fig, update_plot,
cache_frame_data=False, interval=10)
```

```
    plt.show(block=True)


def main_task_pico():
    Nw = len(noiseA)
    positions = []
    positions.append([[], []])
    pos_idx = 0
    with serial.Serial("COM14", 115200) as ser:
        # ser.set_buffer_size(rx_size = 8192)
        ser.write(b"freq\n")
        # print(ser.readline())
        ser.write(f"{50000}\n".encode())
        # print(ser.readline())
        # num_samples = int(Fs)
        while True:
            draw_button_on = 0
            clear_button_on = 0

            ser.write(f"{Nw}\n".encode())
            # print(size)

            rcv_draw = ser.readline()  # For draw button
            rcv_draw = rcv_draw.decode("utf-8").strip()

            rcv_clear = ser.readline()  # For clear button
            rcv_clear = rcv_clear.decode("utf-8").strip()

            num_bytes = Nw * 2  # 2 bytes per sample
            b = ser.read(num_bytes)

            sound = np.frombuffer(b, dtype="<i2")

            found_delay1, max1, avg1, cross_corrA =
correlate_and_find_delay(
                sound, noiseA, "A"
            )
            found_delay2, max2, avg2, cross_corrB =
correlate_and_find_delay(
                sound, noiseB, "B"
            )
            found_delay3, max3, avg3, cross_corrC =
correlate_and_find_delay(
                sound, noiseC, "C"
            )

            ta = found_delay1 / 48000
            tb = found_delay2 / 48000
            tc = found_delay3 / 48000

            guessed_position = fangs_algorithm_TDoA(ta, tb, tc)

            # Button stuff
            if rcv_draw   "press":
                draw_button_on = True

            if rcv_clear   "cleared":
                clear_button_on = True

            if guessed_position is None:
                cursor_position = INVALID_CURSOR_POS

            else:
                cursor_position = guessed_position
                if draw_button_on:
                    # If button is pressed
                    positions[pos_idx][0].append(
                        guessed_position[0]
```

```
                    )  # Add x coord in the curr drawing list
                    positions[pos_idx][1].append(
                        guessed_position[1]
                    )  # Add y coord in the curr drawing list
                else:
                    # If button is released, add new list to positions
                    positions.append([[], []])
                    pos_idx += 1

            if clear_button_on   1:
                positions.clear()
                positions.append([[], []])
                pos_idx = 0

            if thread_queue.empty():
                thread_queue.put(
                    QueueMsg(
                        cross_corrA,
                        cross_corrB,
                        cross_corrC,
                        copy.deepcopy(positions),
                        cursor_position,
                    )
                )
            else:
                print("dropping frame")


def main_task_mic():
    Nw = len(noiseA)

    audio_buffer_queue = queue.Queue()

    def audio_callback(indata, frames, time, status):
        """This is called (from a separate thread) for each audio
block."""

        if status:
            print(status)

        length = min(Nw - audio_callback.index, frames)
        audio_callback.buffer[audio_callback.index :
audio_callback.index + length] = (
            indata[:length].flatten()
        )

        audio_callback.index += length

        if audio_callback.index >= Nw:
            audio_callback.index = 0
            if audio_buffer_queue.empty():
                audio_buffer_queue.put(audio_callback.buffer.copy())
            else:
                print("dropping sound")

    audio_callback.index = 0
    audio_callback.buffer = np.zeros(Nw)

    import sounddevice as sd

    stream = sd.InputStream(channels=1, samplerate=48000,
callback=audio_callback)
    with stream:
        positions = []
        while True:
            sound = audio_buffer_queue.get()

            found_delay1, max1, avg1, cross_corrA =
```

```
correlate_and_find_delay(
                sound, noiseA, "A"
            )
            found_delay2, max2, avg2, cross_corrB =
correlate_and_find_delay(
                sound, noiseB, "B"
            )
            found_delay3, max3, avg3, cross_corrC =
correlate_and_find_delay(
                sound, noiseC, "C"
            )

            ta = found_delay1 / 48000
            tb = found_delay2 / 48000
            tc = found_delay3 / 48000

            guessed_position = fangs_algorithm_TDoA(ta, tb, tc)

            if guessed_position is None:
                cursor_position = INVALID_CURSOR_POS

            else:
                cursor_position = guessed_position

            if thread_queue.empty():
                thread_queue.put(
                    QueueMsg(
                        cross_corrA,
                        cross_corrB,
                        cross_corrC,
                        positions.copy(),
                        cursor_position,
                    )
                )
            else:
                print("dropping frame")


t1 = threading.Thread(target=main_task_pico, daemon=True)
# t1 = threading.Thread(target=main_task_mic, daemon=True)
t1.start()

do_plot()
```

sound-demo/sound-jupyter-notebook.py

```
# %%
%pip install numpy
%pip install scipy
%pip install matplotlib
%pip install pyserial
%pip install sympy
%pip install sounddevice

# %%
Fs = 50000
Ts = 1 / Fs

# Nw = int(noise_dur_s * Fs )
Nw = 2**14
# White noise length
noise_dur_s = Nw*Ts

silence_length = int(0/343 * Fs) # 20m worth of sample travel time
silence_length_s = silence_length * Ts

full_length = Nw + silence_length
```

```
cutoffs = [1000, 5000]


# %%
import scipy
import scipy as sp
import numpy as np
import matplotlib.pyplot as plt
## Generate noise

def make_linear_chirp(f0, f1, N, Ts):
  t = np.arange(N) * Ts

  c = (f1-f0)/(N*Ts)

  phi_0 = 0
  chirp = np.sin(phi_0 + 2*np.pi*(c/2*t*t+f0*t))
  return chirp


def make_hyperbolic_chirp(f0, f1, N, Ts):
  t = np.arange(N) * Ts

  phi_0 = 0
  chirp = np.sin(
      phi_0
        + 2*np.pi*(-f0*f1*noise_dur_s/(f1-f0))
        * np.log(1-(f1-f0)/f1/noise_dur_s*t)
  )
  return chirp

full = make_linear_chirp(cutoffs[0], cutoffs[1], Nw, Ts)
half = make_linear_chirp(cutoffs[0], cutoffs[1], Nw//2, Ts)

filtered_noiseA = full
filtered_noiseB = np.flip(full)
filtered_noiseC = np.concatenate((half, np.flip(half)))
#filtered_noiseD = np.concatenate((np.flip(half), half))
#print(max(filtered_noiseA))

def plot_spect(chirp):
  f, t, Sxx = scipy.signal.spectrogram(chirp, Fs)
  plt.pcolormesh(t, f, Sxx, shading='gouraud')
  plt.ylim([500, 6000])

def plot_freq(chirp):
  # plot noise
  noise_fft = sp.fft.rfft(chirp)
  n = len(noise_fft)
  f = np.arange(n) * Fs / n / 2
  plt.plot(f, 10 * np.log10(np.abs(noise_fft)))

plt.figure()
plt.subplot(131)
plot_spect(filtered_noiseA)
plt.xlabel('Time (s)')
plt.ylabel('Frequency [Hz]')
plt.title("Chirp A")
plt.subplot(132)
plot_spect(filtered_noiseB)
plt.title("Chirp B")
plt.xlabel('Time (s)')
plt.tick_params("y", labelleft=False)
plt.subplot(133)
plot_spect(filtered_noiseC)
plt.title("Chirp C")
plt.xlabel('Time (s)')
```

```
plt.tick_params("y", labelleft=False)

filtered_noiseA = np.concatenate((filtered_noiseA,
np.zeros(silence_length)))
filtered_noiseB = np.concatenate((filtered_noiseB,
np.zeros(silence_length)))
filtered_noiseC = np.concatenate((filtered_noiseC,
np.zeros(silence_length)))

plt.figure()
plt.subplot(131)
plot_freq(filtered_noiseA)
plt.ylabel('Frequency [Hz]')
plt.xlabel('Time [sec]')
plt.subplot(132)
plot_freq(filtered_noiseB)
plt.subplot(133)
plot_freq(filtered_noiseC)



# %%
np.save("filtered_noiseA.npy", filtered_noiseA)
np.save("filtered_noiseB.npy", filtered_noiseB)
np.save("filtered_noiseC.npy", filtered_noiseC)

# %%
for (letter, noise) in [('A', filtered_noiseA), ('B', filtered_noiseB),
('C', filtered_noiseC)]:
    with open(f"../pico_play_noise/src/sound_samples/
sound_samples{letter}.h", "w") as f:
        noise = np.copy(noise)
        noise *= 2**15-1
        noise = np.asarray(noise, dtype='int16')
        hexed = [f"{sample}" for sample in noise]
        f.write("#include <stddef.h> // for size_t\n")
        f.write("#include <stdint.h> // for uint16_t\n")
        f.write("\n")
        f.write(f"#define SOUND_SAMPLES_LEN {len(hexed)}\n")
        f.write("\n")
        f.write("//These are actually signed ints \n")
        f.write("const int16_t sound_samples[SOUND_SAMPLES_LEN] = {\n")
        for window in range(0, len(hexed), 10):
            s = "    " + ", ".join(hexed[window:window+10]) + ",\n"
            f.write(s)
        f.write("};\n")


# %%
import numpy as np
import matplotlib.pyplot as plt

def correlate_and_find_delay(rec, noise, name):
    # rec_padded = np.pad(rec, (len(noise), 0), 'constant',
constant_values=0)
    # print(rec)
    rec_fft = np.fft.rfft(rec)
    diff = len(rec) - len(noise)
    assert diff   0
    noise_padded = np.pad(noise, (0, diff), "constant",
constant_values=0)
    noise_fft_conj = np.conj(np.fft.rfft(noise_padded))
    # print(rec_fft.shape, noise_fft_conj.shape)
    cross_corr_freq = noise_fft_conj * rec_fft
    cross_corr = np.abs(np.fft.irfft(cross_corr_freq))

    #valid_len = diff + 1
    #cross_corr = cross_corr[:valid_len]
```

```
    # print(cross_corr)
    plt.plot(cross_corr)
    plt.title("correlation " + name)

    k_max_ind = np.argmax(cross_corr)
    k_max = cross_corr[k_max_ind]
    avg = np.sum(cross_corr) / len(cross_corr)

    return k_max_ind, k_max, avg

# %%
import numpy as np
import matplotlib.pyplot as plt

recording_noise_1 = np.zeros(full_length)
recording_noise_2 = np.zeros(full_length)
recording_noise_3 = np.zeros(full_length)

insert_loc = 400
# recording_noise_1[insert_loc : insert_loc + Nw] += filtered_noise1
recording_noise_1 = np.roll(filtered_noiseA, insert_loc)
insert_loc2 = 1000
# recording_noise_2[insert_loc2 : insert_loc2 + Nw] += filtered_noise2
recording_noise_2 = np.roll(filtered_noiseB, insert_loc2)
insert_loc3 = 1200
# recording_noise_3[insert_loc3 : insert_loc3 + Nw] += filtered_noise3
recording_noise_3 = np.roll(filtered_noiseC, insert_loc3)

recording_noise_together = recording_noise_1 + recording_noise_2 +
recording_noise_3
# recording_noise_together = recording_noise_together /
max(recording_noise_together)

plt.figure()
ind1 = correlate_and_find_delay(recording_noise_1, filtered_noiseA,
"1")
plt.figure()
ax1 = plt.subplot(141)
ind1, max1, avg1 = correlate_and_find_delay(recording_noise_together,
filtered_noiseA, "1")
plt.subplot(142, sharey=ax1)

ind2, max2, avg2 = correlate_and_find_delay(recording_noise_together,
filtered_noiseB, "2")
plt.tick_params("y", labelleft=False)
plt.subplot(143, sharey=ax1)
ind3, max3, avg3 = correlate_and_find_delay(recording_noise_together,
filtered_noiseC, "3")
plt.tick_params("y", labelleft=False)
print("indexes:", ind1, ind2, ind3)
print("maxes:", max1, max2, max3)
print("avgs:", avg1, avg2, avg3)
print("max/avg:", max1/avg1, max2/avg2, max3/avg3)

# %%
### Time Difference of Arrival with 3 nodes

A = np.array([0, 0])
B = np.array([10, 0])
C = np.array([5, 10])

actual_position = np.array([5, 0])

delay1 = int(abs(np.linalg.norm(A - actual_position)) * Fs // 343)
delay2 = int(abs(np.linalg.norm(B - actual_position)) * Fs // 343)
delay3 = int(abs(np.linalg.norm(C - actual_position)) * Fs // 343)
```

```
recording_noise = np.zeros(full_length)
print(delay1)
#recording_noise[delay1 : delay1 + Nw] += filtered_noise1
recording_noise += np.roll(filtered_noiseA, delay1)
#recording_noise[delay2 : delay2 + Nw] += filtered_noise2
recording_noise += np.roll(filtered_noiseB, delay2)
#recording_noise[delay3 : delay3 + Nw] += filtered_noise3
recording_noise += np.roll(filtered_noiseC, delay3)


def fangs_algorithm_TDoA(ta, tb, tc):
    ## Fang's algorithm, gotten from the PDF (in the repo or at
https://ieeexplore.ieee.org/document/102710)
    assert A[0]   0 and A[1]   0
    assert B[0] > 0 and B[1]   0
    assert C[1] > 0

    c = 343  # speed of wave in medium, speed of sound=343 m/s

    cTa = ta * c
    cTb = tb * c
    cTc = tc * c

    b = B[0]  # also = np.linalg.norm(B)
    cx = C[0]
    cy = C[1]
    c = np.linalg.norm(C)

    Rab = cTa - cTb
    Rac = cTa - cTc

    #avoid division by zero, just make em real small
    if Rab   0.0:
        Rab = 1e-5
    if Rac   0.0:
        Rac = 1e-5

    # variable names correspond to those in the paper

    g = (Rac * b / Rab - cx) / cy
    h = (c**2 - Rac**2 + Rac * Rab * (1 - (b / Rab) ** 2)) / (2 * cy)

    d = -(1 - (b / Rab) ** 2 + g**2)
    e = b * (1 - (b / Rab) ** 2) - 2 * g * h
    f = (Rab**2 / 4) * (1 - (b / Rab) ** 2) ** 2 - h**2

    z = 0
    x = np.roots([d, e, f - z**2])  # eq 9a
    y = g * x + h  # eq 13
    print(x, y)

    guesses = np.transpose([x, y])
    print(guesses)

    def err(g):
        # calculate what sort of time deltas would be seen with this
guess
        # compare them to the original, return the MSE
        deltaA = np.linalg.norm(g - A)
        deltaB = np.linalg.norm(g - B)
        deltaC = np.linalg.norm(g - C)
        rab = deltaA - deltaB
        rac = deltaA - deltaC
        mse = ((rab - Rab) ** 2 + (rac - Rac) ** 2) / 2
        return mse
    errs = [err(a) for a in guesses]
    print(errs)
```

```
    best_guess = min(guesses, key=err)
    return best_guess


plt.figure()
ax1 = plt.subplot(231)
found_delay1, max1, avg1 = correlate_and_find_delay(recording_noise,
filtered_noiseA, "A")
plt.subplot(232, sharey=ax1)
found_delay2, max2, avg2 = correlate_and_find_delay(recording_noise,
filtered_noiseB, "B")
plt.tick_params('y', labelleft=False)
plt.subplot(233, sharey=ax1)
found_delay3, max3, avg3 = correlate_and_find_delay(recording_noise,
filtered_noiseC, "C")
plt.tick_params('y', labelleft=False)

ta = found_delay1 / Fs
tb = found_delay2 / Fs
tc = found_delay3 / Fs

guessed_position = fangs_algorithm_TDoA(ta, tb, tc)
plt.subplot(212)
plt.scatter([A[0], B[0], C[0]], [A[1], B[1], C[1]], label="base
stations", marker="o")
plt.scatter(guessed_position[0], guessed_position[1], label="position
guess")
plt.scatter(actual_position[0], actual_position[1], label="actual
position", marker="x")
plt.gca().set_aspect("equal")
plt.legend(loc="center left", bbox_to_anchor=(1.0, 0.5))

# %%
from sympy import plot_implicit, symbols, Eq, sqrt

x, y = symbols('x y')

A = (0,0)
B = (1,0)
C = (0.5,1)

p = (0.75, 0.15)

R_A = sqrt((p[0]-A[0])**2 + (p[1]-A[1])**2)
R_B = sqrt((p[0]-B[0])**2 + (p[1]-B[1])**2)
R_C = sqrt((p[0]-C[0])**2 + (p[1]-C[1])**2)

p1 = plot_implicit(Eq(sqrt((A[0]-x)**2 + (A[1]-y)**2), R_A), (x, -1,
2), (y, -1, 2), show=False)
p2 = plot_implicit(Eq(sqrt((B[0]-x)**2 + (B[1]-y)**2), R_B), (x, -1,
2), (y, -1, 2), show=False)
p3 = plot_implicit(Eq(sqrt((C[0]-x)**2 + (C[1]-y)**2), R_C), (x, -1,
2), (y, -1, 2), show=False)
p1.append(p2[0])
p1.append(p3[0])
p1.show()

# %%
import matplotlib.pyplot as plt


c = 343
ta = R_A/c
tb = R_B/c
tc = R_C/c

p2 = plot_implicit(Eq(sqrt((B[0]-x)**2 + (B[1]-y)**2)-sqrt((A[0]-x)**2
+ (A[1]-y)**2), c*(tb-ta)), (x, -1, 2), (y, -1, 2), show=False,
```

```
line_color="b",
                    annotations=[{'xy': (A[0], A[1]), 'text': "A", 'ha':
'center', 'va': 'center', 'color': 'green', 'size': 'large',
'backgroundcolor':'white'},
                                 {'xy': (B[0], B[1]), 'text': "B", 'ha':
'center', 'va': 'center', 'color': 'green', 'size': 'large',
'backgroundcolor':'white'},
                                 {'xy': (C[0], C[1]), 'text': "C", 'ha':
'center', 'va': 'center', 'color': 'green', 'size': 'large',
'backgroundcolor':'white'},
                                 {'xy': (0.5, 0.40),   'text': "c*(tc-
ta)", 'rotation': -18.0, 'ha': 'center', 'va': 'center', 'color':
'red'},
                                 {'xy': (0.90, 0.8), 'text': "c*(tb-
ta)", 'rotation': 60.0, 'ha': 'center', 'va': 'center', 'color':
'blue'},]
)
p3 = plot_implicit(Eq(sqrt((C[0]-x)**2 + (C[1]-y)**2)-sqrt((A[0]-x)**2
+ (A[1]-y)**2), c*(tc-ta)), (x, -1, 2), (y, -1, 2),show=False,
line_color="r")
p2.append(p3[0])

p2.show()
```

Appendix B: Progress Report

Carleton University

SYSC 4907 Engineering Project

School Year 2023-2024

Automated Shopping Cart

With the Application of Indoor Localization

Progress Report

Project Members:

Ahmed Abdelaziz (101202944)
Nadia Ahmed (101172713)
Bilal Chaudhry (101141634)
Cam Frohar (101148699)
Mohamed Kaddour (101140829)
Sebastien Marleau (101155551)
Akuei Minyang (101133928)
Kousha Motazedian (101140854)
Bren Gelyn Padlan (101148482)
Prianna Rahman (101140422)

Project Supervisors:

Dr. Ramy Gohary
Dr. Ian Marsland

Table of Contents

1 Introduction 3
2 Learning Modules 3
2.1 ADC/DAC Labs 3
2.2 Serial Communications Lab 4
2.3 Timers Lab 4
2.4 Interrupt Lab 4
3 Mini Projects 5
3.1 Sensor Fusion Team 5
3.1.1 Ultrasonic Sensor 5
3.1.2 Gyroscope – MPU6050 6
3.1.3 Accelerometer – MMA8451Q 6
3.1.4 GPS 7
3.2 Motors 8
3.2.1 Challenges 9
3.3 Indoor Localization 10
3.3.1 Acoustic Indoor Localization 10
3.3.2 Indoor LIDAR Localization 10
4 Future Iteration 12
References 13

1 Introduction

The purpose of this document is to provide a comprehensive update on our fourth-year capstone project, which involves the development of an automated shopping cart utilizing indoor localization technology. Over the initial two-month period, our team dedicated efforts to gaining a thorough understanding of the FRDM-KL25Z board. This was achieved through a series of weekly modules, each focusing on different technical aspects such as ADC/DAC, UART, Timers, and Interrupts. These foundational topics, which will be elaborated on later in this report, have been crucial in our progress.

Following this introductory phase, we transitioned into a more hands-on approach. The team was divided into three specialized sub-groups, each tasked with executing individual mini projects. These projects were strategically designed to not only apply the knowledge acquired from the earlier modules but also to contribute essential components to our final integrated system. This structured approach ensures that each segment of the project aligns seamlessly with our overarching goal of developing a fully functional user-controlled vehicle.

2 Learning Modules

To achieve successful completion of this project, the initial phase involved thorough research and experimentation. Among the various concepts explored, four topics namely ADC/DAC, serial communications, timers, and interrupts will be significant for the development of the shopping cart.

2.1 ADC/DAC Labs

In the analog interfacing laboratory, we explored the functionality of infrared (IR) sensors by employing an Analog-to-Digital Converter (ADC) and a Digital-to-Analog Converter (DAC). The primary goal of this lab was to enhance our proficiency in constructing circuits, operating an oscilloscope, and gaining a comprehensive understanding of ADC and DAC and their application. Within this experimental setting, we utilized IR LEDs, with one serving as an IR emitter and the other as an IR receiver. A circuit was assembled to enable the microcontroller to transmit a digital signal to the IR emitter, generating pulses of IR signals, and to capture and interpret the analog data received by the IR receiver.

This lab significantly improved our debugging skills as we utilized an oscilloscope to visualize the signals transmitted and received by the microcontroller from the IR LEDs. Furthermore, our competence in interpreting the microcontroller datasheet empowered us to fully grasp and effectively implement the ADC and DAC features of our microcontroller board.

The skills acquired in this laboratory are crucial for our primary project, which aims to develop a shopping cart capable of avoiding obstacles. Many obstacle detection sensors, including IR sensors, depend on proximity or distance from objects. By gaining insights into these concepts, we can leverage our understanding of proximity sensors and the conversion of data from analog to digital, and vice versa. This knowledge will prove invaluable as we determine the type of sensor to integrate into our project.

2.2 Serial Communications Lab

In this lab, we explored the universal asynchronous transmitter/receiver (UART) serial communication protocol. We utilized a USB to serial adapter in conjunction with the microcontroller to perform serial communication between the microcontroller and user input via a terminal emulation program (ie. PuTTY). To perform this lab, we configured the proper UART registers in the microcontroller with embedded C. We then connected the USB to serial adapter into the microcontroller and a PC. Through PuTTY, the user was prompted to enter a number. Communication via UART was used to send the number from the terminal emulator to the microcontroller, where its square root was taken and then returned back to the terminal emulator. Along with gaining knowledge of UART, we also increased our understanding of polling and interrupts. This lab activity was important for the development of our capstone project, as we may use this protocol for communication between microcontrollers with different attached modules, such as wireless modules. We also aim to explore more serial communication protocols as needed, including serial peripheral interface (SPI) and inter-integrated circuit (I2C).

2.3 Timers Lab

In the timer lab we investigated the PIT (Periodic Interrupt Timer) module where an interrupt would be used to generate Square, Ramp and Sine waves. We would use interrupts to dequeue from a queue of sample/data points that were assigned based on the chosen waveform to conduct a DAC (Digital to Analog Conversion). The primary focus of the lab was on the generation of the Sinusoidal wave. This DAC would be hooked up to an oscilloscope in order to confirm that the wave was generated as expected. The PIT timer interrupted periodically which would simulate a period of the wave form. Each interrupt of the timer would correspond with a step in the period, and the next value for the sinusoidal wave would be directed from the queue to the DAC. The lab required all members of the team to get a strong and intricate understanding of the DAC (See section on DAC) and PIT diagrams done through the Kinetis KL25Z manual. This laboratory required a surface level understanding of the fundamentals of operating an oscilloscope. The laboratory utilized a key data structure in queues in order to neatly process sample points.

2.4 Interrupt Lab

This lab aimed to learn about interrupts and how they worked with our microcontroller. The lab consisted of a button and an RGB light, it was a very straightforward setup as the RGB light was already on the board. In the code, we had the LEDs used for the RGB light set up as output GPIO pins, allowing us to use registers on the board to modify the pin values (e.g. change the color of the RBG light). For the switch, we had it set up with a pull-up resistor and enabled interrupts for the port we were using (in this case Port D). The interrupt handler (whenever the button is pressed) would first check to ensure that the button was the cause of the interrupt. If it was, it would increment a variable called count. This variable is used for changing the color of the RGB light. If the least significant bit was set to 1, the red LED would turn on, if the second least significant bit was set to 1, the green LED would turn on, and finally if the 4 least significant bit was set to 1, the blue LED would turn.

3 Mini Projects

We decided to work on dedicated mini projects to build our skills in areas relevant to our capstone. These groups include sensors, motors, and indoor localization.

3.1 Sensor Fusion Team

Within the sensor fusion team, we've been actively exploring and applying diverse methodologies to automate the forthcoming design of a shopping cart scheduled for the next semester. Our research journey involved acquiring proficiency in deploying various sensors, including ultrasonic sensors, gyroscopes, accelerometers, and GPS. By using these sensors, our goal is to enable the shopping cart to constantly monitor its environment through an ultrasonic sensor, regulate its speed using an accelerometer, determine its precise location via GPS, and accurately measure its rotation with a gyroscope. This section will be divided into the different sensors we have decided to research and attempted to implement, along with an assessment of the performance of each sensor thus far.

3.1.1 Ultrasonic Sensor

As part of the collision avoidance and environmental sensing aspect of automating our shopping cart, some sort of distance sensors are required to be equipped on our shopping cart to constantly measure obstacles and moving humans. Our brainstorming involved the concept of integrating multiple ultrasonic sensors, synchronized with the shopping cart's motors, to dynamically adjust its speed in response to the proximity of surrounding obstacles.

An ultrasonic sensor works by emitting high-frequency sound waves beyond the range of human hearing. It consists of one transmitter and one receiver. The transmitter emits ultrasonic waves, which then travel through the air until they encounter an obstacle. Upon hitting a potential obstacle, the waves are reflected back towards the sensor. The receiver component of the sensor detects these reflected waves, and the sensor calculates the distance to the obstacle based on the time it takes for the waves to travel to the obstacle and back. By measuring this time delay, the ultrasonic sensor can provide accurate distance information, allowing it to detect the proximity of its surroundings.

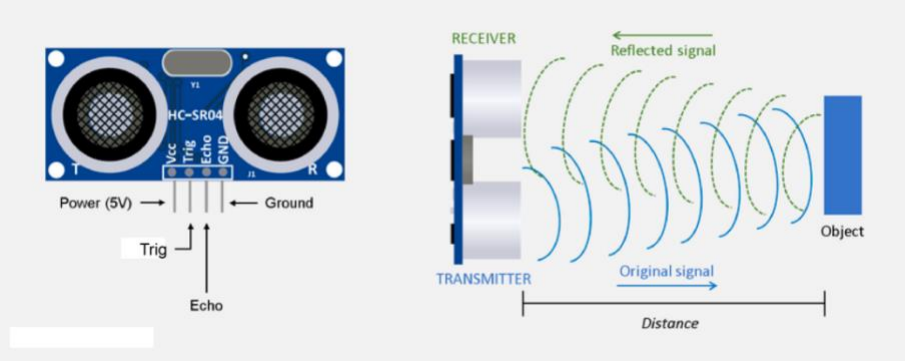

Figure 1: Ultrasonic Sensor pinout and design [1]

1 "Micro bit lesson - using the Ultrasonic module," Micro bit Lesson - Using the Ultrasonic Module " osoyoo.com, https://osoyoo.com/2018/09/18/micro-bit-lesson-using-the-ultrasonic-module/ (accessed Dec. 8, 2023).

We successfully managed to design a program that measured the pulse width of the echo (receiving) signal. We used the receiving signal's pulse width to sound a buzzer when an object approached the sensor within a 10cm distance. Of course, for now, this is just a random value chosen in order to validate the functionality of our design. In the design of our actual shopping cart, we will use the constantly measured sensor values alongside the shopping cart's current speed to determine how much we need to accelerate or decelerate, to ensure a functioning and smooth driving final product.

When brainstorming, we considered alternative distance sensors such as VL53L0X Time of Flight Sensors or GP2Y0A51SK0F Analog Distance Sensor. However, after conducting tests to evaluate the maximum range and overall performance of each sensor, it became evident that ultrasonic sensors stood out as the optimal choice for meeting our specific requirements.

Our upcoming tasks involve the integration of multiple ultrasonic sensors collaborating seamlessly. Additionally, we plan to collaborate with the Indoor Localization Team to combine ideas on incorporating ultrasonic sensors alongside the Lidar (Light Detection and Ranging) sensor. This collaborative effort aims to enhance the accuracy of the overall environmental sensing requirements of the shopping cart.

3.1.2 Gyroscope – MPU6050

The integration of the MPU6050 gyroscope into our autonomous shopping cart project was a strategic decision driven by the device's ability to precisely track orientation and rotation. The gyroscope component of the MPU6050 is instrumental in providing real-time data on the cart's angular velocity across its x, y, and z axes. This information is crucial for maintaining accurate directional control, a key aspect for an autonomous vehicle navigating through turns and avoiding obstacles. Additionally, the gyroscope plays a vital role in ensuring the stability and balance of the cart, especially critical when traversing uneven terrain or executing sharp maneuvers.

In terms of program implementation, we plan to utilize the gyroscope data to continuously adjust the cart's orientation and balance, integrating it with other sensor inputs, such as those from the accelerometer and ultrasonic sensors. Moving forward, our goal is to be able to control the cart's navigation capabilities in complex indoor environments, defining the exact angle of rotation we want our cart to turn at.

3.1.3 Accelerometer – MMA8451Q

The accelerometer built into the FRDM-KL25Z board, specifically the MMA8451, plays a pivotal role in our project. We chose to utilize this accelerometer due to it being conveniently built into the board, and for its ability to provide real-time data on acceleration and inclination, contributing significantly to speed regulation and stability control. The MMA8451 accelerometer measures acceleration in three axes (X, Y, Z). This allows us to detect both linear motion and orientation. By analyzing changes in acceleration, we can determine if the cart is speeding up, slowing down, or tilting, which is crucial for both speed control and ensuring the cart remains upright and stable during operation.

Free fall detection involves monitoring the acceleration data across three axes (X, Y, and Z). To understand the MMA8451 better, we made a mini fall detection device that detects when

an object has fallen from a surface. In free fall, the only force acting on the object is gravity, which ideally results in acceleration readings that approximate the acceleration due to gravity (approximately 9.81 m/s^2) when subtracted from the static gravitational pull. This state is characterized by near-zero effective acceleration. The core of our free fall detection algorithm involves continuously sampling acceleration data and comparing it against a predefined threshold. This threshold is set slightly above zero g to account for noise and minor fluctuations in the sensor readings. We plan to implement a moving average filter to smooth out the data, reducing the likelihood of false triggers due to transient vibrations or abrupt movements.

We also found that the MPU6050 has built-in features for motion detection, including configurable interrupts. We want to program these interrupts to trigger when the accelerometer readings fall below our free fall threshold for a specified duration, which we determined through empirical testing. This approach ensures that the system only reacts to sustained free fall conditions, rather than brief, inconsequential dips in acceleration.

3.1.4 GPS

As part of achieving precise localization for the automated shopping cart, our team opted to acquire an Adafruit Ultimate GPS module so that we could test its accuracy and overall performance. We believe that an automated shopping cart should possess the capability to accurately determine its location. This is essential for tasks such as identifying the endpoint of a designated path or monitoring its movement in various directions.

The Adafruit Ultimate GPS module operates by receiving signals from GPS satellites, processing the data, and providing accurate location information through a serial interface such as I2C or UART. The module receives signals from a network of orbiting satellites and processes the data to determine its precise location and speed [2]. A module like this can help the shopping cart not only know its precise location, but also gain access to its current speed, which can work side-by-side with the distance sensors to smoothly accelerate and decelerate based on its surroundings.

As for our progress with this module, our current focus involves implementing a solution to execute a program capable of retrieving data such as speed and precise location. Furthermore, we intend to assess its functionality indoors, recognizing the potential challenge of weaker performance for a GPS module in indoor environments.

2 How GPS works, https://www.maptoaster.com/maptoaster-topo-nz/articles/how-gps-works/how-gps-works.html (accessed Dec. 8, 2023)

3.2 Motors

The motors team's primary goal was to experiment with different motor types, motor drivers and methodologies; the results of which would be used in conjunction with localization and sensor modules to assist in automating the cart.

To operate the motor, and provide control over the rotation direction, a motor driver was used. The motor driver is an external module that can be connected with the FRDM-KL25z to operate the motor, controlling the rotation, speed and allowing the use of an external power source. This is a crucial component of the team's goal, as to bring the cart to a stop, and turn the cart, it is required that the speed of the wheels can be controlled individually.

To control the RPM of the DC motors, the microcontroller GPIO pins were configured to utilize the TPM (). This was done by manipulating the TPM module to act as an analog PWM (Pulse Width Modulation). Using an Edge-aligned PWM, it is possible to configure a signal period (using the MOD register) and specify the duty cycle and the pulse width through the CnV registers. On timer overflows, the period resets in order to generate the PWM signal. This can be used to specify the duty cycle in a percentage of the MOD register value to the CnV register value.

The motor driver that was initially selected was the L298N, dual H-bridge motor driver. An H-bridge is a circuit design that allows the directional control of the motor using two input pins. Figure x below shows the circuit diagram of the H-bridge. When S1 and S4 are closed, and S2 and S3 are open, a positive voltage is supplied across the motor and the motor spins, and if the S2 and S3 are closed, with S1 and S4 open, a negative voltage is applied, and the spin is reversed. A dual H-bridge implies that two of these mechanisms are present, allowing the individual control of two motors with the single module.

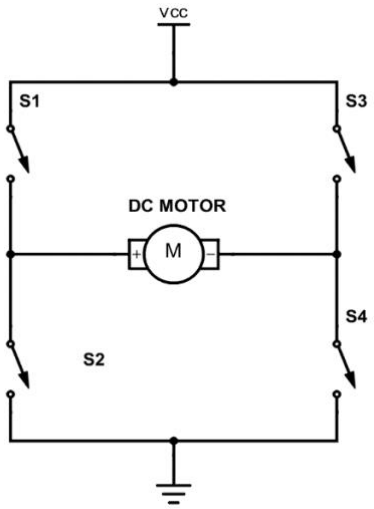

Figure 2: H-Bridge circuit diagram [3]

3 Ø. N. Dahl, "What is an H-bridge?," Build Electronic Circuits, https://www.build-electronic-circuits.com/h-bridge/ (accessed Dec. 8, 2023).

The L298N can drive DC motors up between 5V and 35V, and a peak current of 2A. This would be sufficient to power larger motors, as well as the small 3-9V motors used in the preliminary design. In the final design, two of these modules would be required, one to power each pair of motors. An external power supply of 18V was provided to the module, which should be sufficient to power two 3-9v motors.

Another dual H-bridge motor driver, the L298D, was also considered. This driver operates the same as the L298D, however it features a much more compact design, and remains capable of driving DC motors between 5V and 35V. The L298N, however, is a more advanced IC, and is capable of handling wider temperature ranges. For the preliminary design we operate smaller motors, and as such the power being supplied is lower, and therefore the L298D seems to be the optimal selection.

With the recommendations of the other mini project teams, a wheel encoder will be used in order to read the RPM of the motors as they operate. This will allow real time data to be gathered from the wheels, which can be used to determine the acceleration of the cart, which will aid the sensors team in determining the amount of braking needed to come to a stop. The localization team will also use this information to estimate the current position of the cart through odometry.

Besides the technical implementation of the motors, we also were required to make design decisions on the control of the wheels when turning, and the issue of scaling up to support different sized carts and amounts of groceries.

3.2.1 Challenges

We intended to employ the L298N H-bridge motor driver in our project; however, we encountered issues with voltage division to the motors. Notably, we observed inconsistent voltage outputs on output terminals of the H-bridge. We suspect that these discrepancies may arise from the internal configuration of the H-bridge. Further investigation into the problem led to a change in the choice of the H-bridge to L298D. The L298D allowed for the proper division of voltage and due to its size, it led to easier integration into the circuitry of the motors.

Given the choice between manually configuring the cart's wheels with a traditional axle and opting for individual steering control for each wheel, we opted for the latter This decision was driven by the desire to minimize the physical and mechanical workload required for the cart. By equipping each wheel with its own motor, we gained the ability to independently control the speed of each wheel. This individualized control, in turn, enables precise manipulation of the wheels to facilitate turning. The capability to vary the speeds of the wheels independently becomes instrumental in achieving dynamic and controlled turns for the cart.

3.3 Indoor Localization

GPS accuracy decreases when close to buildings or indoors. This may make it unusable or only accurate in the range of tens of meters. Alternatives were needed for the autonomous shopping cart. The currently explored methods include acoustic indoor localization and LIDAR localization.

3.3.1 Acoustic Indoor Localization

Sound waves travel at a rate of 330 m/s. This is much slower than both the speed of light and the processing speed of current microprocessor chips. This makes their use possible in attaining very accurate position measurements.

A demo was initially implemented using multiple laptops, connected via ethernet cables. It works as follows:

1. A transmitting laptop sends a UDP packet to the other laptops, then emits a sound byte.
2. The other laptops receive this ping and start recording audio.
3. The laptops then correlate the sound recorded to the sound byte and find the moment in time when these correlated the best.

The travel time of the sound byte can thus be estimated to be the correlation time minus some latency. The estimated distance can then be calculated. The more consistent the latency, the more accurate the measurements will be.

Issues encountered were inconsistent latencies before sending/recording audio, as well inconsistent time needed to transmit and receive the UDP packets.

Next steps:

- Using time difference of arrival, removing the need for the transmitter to send a message to the receivers when it begins transmission. The receivers will need to communicate with each other.
- Replacing the receiving laptops with microphone-enabled microcontrollers, communicating with each other via some kind of wireless transmission.
- Replacing the transmitter with a speaker-enabled microcontroller or single board computer.

3.3.2 Indoor LIDAR Localization

Light Detection and Ranging (LIDAR) is a sensing method that uses light in the form of a pulsed laser to measure distances relative to its position. We used the RPLIDAR A1M8 which scans at a frequency of 10 Hz, meaning there are 10 different revolutions within one second. The LIDAR we used returns a tuple of distance, angle and quality. The quality parameter pertains to the quality of the measurement taken. LIDAR can be very useful for indoor localization as it can create images of what it detects. Using this, we can invoke mapping of an entire movement throughout the motion of a robot where the LIDAR is mounted. An initial demo was implemented using Hector SLAM (simultaneous localization and mapping), a technology that helps create a map without the use of odometry (direction). This is used to map previous LIDAR scans in relative previous time.

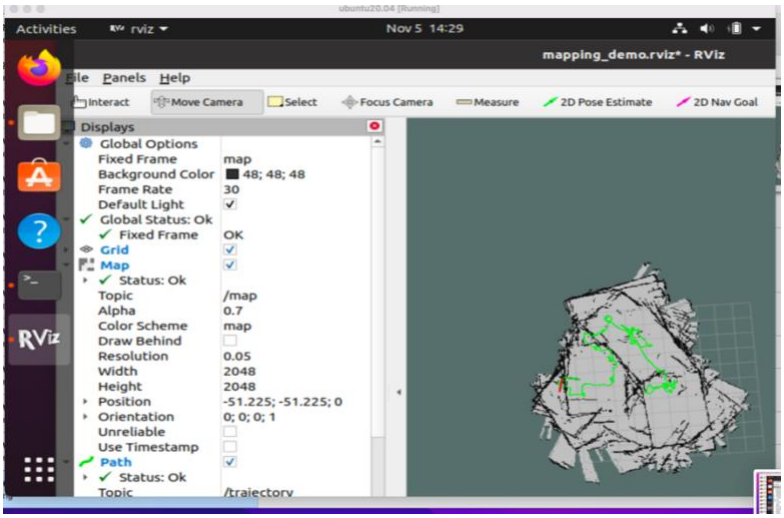

Figure 3: Mapping of Room as User Moves LIDAR Manually using Hector SLAM

Although this implementation of Hector SLAM is useful, we would like to create an implementation from scratch. LIDAR is a relatively new concept, but there are several tools to improve understanding upon the sensor. We attempted to extract raw LIDAR data with a MATLAB program and Python program through the use of a Python library. The MATLAB program was able to communicate with the LIDAR, however no data could be extracted. The Python program was more successful as data was extracted, but data extraction was slow and inconsistent.

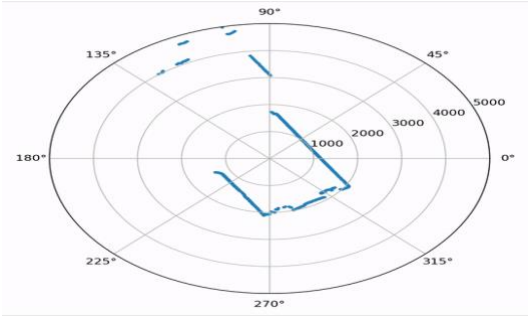

Figure 4: Mapping of LIDAR using Python Program [5]

The current end goal is to use the LIDAR to track a user's body movement and follow the user within the LIDAR range. Once LIDAR data is collected, it can be manipulated to a developer's specification. The currently planned solution is to use the RPLIDAR Software Development Kit (SDK) given by the manufacturer to work with direct configuration rather than using Python libraries or the Robot Operating System (ROS) framework, which provide pseudo-solutions for the current issue. While the current research has proven very useful, we would like to implement our own, unique solution.

4 Future Iteration

As the mini projects have continued, we have further iterated the goals for our overall capstone project. The final goal is to have a small, autonomous cart/basket that will follow a shopper around. To accomplish this, the shopper will carry a remote or tag device. Localization will then be used between the autonomous cart and tag for accurate following. We also aim to have a “return-to-home" feature, which will enable the cart to go back to a specified position after shopping is complete. A user interface, such as a mobile or web app, may also be developed. For object detection and avoidance, a combination of ultrasonic sensors and LIDAR will be used.

References

[1] How GPS works, https://www.maptoaster.com/maptoaster-topo-nz/articles/how-gps-works/how-gps-works.html (accessed Dec. 8, 2023).

[2] Micro bit lesson - using the Ultrasonic module," Micro bit Lesson - Using the Ultrasonic Module " osoyoo.com, https://osoyoo.com/2018/09/18/micro-bit-lesson-using-the-ultrasonic-module/ (accessed Dec. 8, 2023).

[3] Ø. N. Dahl, "What is an H-bridge?," Build Electronic Circuits, https://www.build-electronic-circuits.com/h-bridge/ (accessed Dec. 8, 2023).

[4] MMA8451, 2, 3Q motion and freefall detection using the, https://www.nxp.com/docs/en/application-note/AN4070.pdf (accessed Dec. 8, 2023)

[5] Hyun-je, "PyRPlidar," *GitHub*, 2019. https://github.com/Hyun-je/pyrplidar (accessed Dec. 09, 2023).