REVISING ENVIRONMENT MAPS IN REAL TIME

Yashas Salankimatt

CONCEPT OF OPERATIONS

CONCEPT OF OPERATIONS
FOR
Revising Environment Maps in Real Time

TEAM <83>

APPROVED BY:

Yashas Salankimatt	4/30/2022
Project Leader	Date
_____	
Prof. Kalafatis	Date
_____	
T/A	Date

Change Record

Rev.	Date	Originator	Approvals	Description
-	9/13/2021	Yashas Salankimatt		Draft Release
1	10/2/2021	Yashas Salankimatt		Revision 1
2	12/5/2021	Yashas Salankimatt		Revision 2
3	4/30/2022	Yashas Salankimatt		Final Report Revision

Table of Contents

List of Tables

No table of figures entries found.

List of Figures

1. Executive Summary

When mapping indoor environments that have frequent changes in the environment, like whenever the state of doors is changed or if furniture has moved, existing robotic systems struggle since they most commonly build a static map of their surroundings that is not able to recognize its location whenever only a few things are changed in the surroundings. As robots continue to be used alongside humans, robots and robotic systems should be able to understand where in their environment they are without nearly as much previous knowledge and update their understanding of their surroundings as they explore. This research attempts to solve these problems and make a robot be able to localize itself in a previously known map and then dynamically update that map without the need for remapping the environment completely, providing itself the most up to date environment map for navigation purposes. This research yields a system that allows for the accurate revision of a previously generated ground truth environment map while being able to localize a robot with no previous knowledge of its location within the environment. This research also yields an implementation of this system on a physical robot with a LIDAR sensor, as well as a simulation of this system and a comparison of this method with other methods of localization and map revision of small, day-to-day changes to human indoor environments.

2. Introduction

Commonly, small indoor robotic systems cannot properly perceive their environments and update their perception of the environment in real-time. This research attempts to allow these robotic systems to be able to update their previous knowledge of any indoor environment instead of having to reacquire the knowledge from scratch.

2.1. Background

As robots start to get cheaper and more affordable, the range of things that robots will be used for and the environments that they will traverse will continue to grow more and more varied. As these robots change over from environments like factory floors and warehouses to places like buildings and apartments, the algorithms that these robots use will need to be updated so that the robots will be able to properly perceive their environment and respond accordingly (Zlatanova et al., 2013).

Currently, most indoor mapping systems will take some form of map of an indoor space, whether it be point cloud data or just a 2D occupancy grid of the space, and just store it within the robot. Hence, the robot must keep track of where it powered off in the space in order to keep its bearings. Many, if not all these systems are also not able to easily update the previously created map with new information of the space, like when furniture is moved, doors are opened, etc. These problems lead to robots that currently cannot navigate human spaces very naturally at all (Zlatanova et al., 2013).

As discussed above, current robotic systems cannot autonomously navigate human environments intelligently at all as their perception of the space is most often flawed and not up to date since human systems by nature, have a lot of change on a day to day or hour to hour basis. To fix this perception, a robotic perception system must be able to take a ground truth initial map of an environment and then be able to update the environment with any changes that occur within the system (Khoshelham & Zlatanova, 2016).

2.2. Overview

To do this, the robot kidnapping system must be solved intelligently within this project, wherein a robot will not know where it is if it is taken to a random location and turned on (Kim et al., 2007). Once the robot kidnapping problem is solved and the robot can know where it is on a random power up, the robot must also be able to intelligently identify when it is in the same environment as before, but just with a few changes like doors being in different positions or furniture being moved around. This will involve intelligently identifying landmarks that don't change between runs, like walls, large tables, cabinets, etc.

Once the robotics system is able to adapt to these simple changes in its surroundings, this work can be applied to robotics of many different uses. Obviously single agent robotic systems can benefit from this by being able to path find more intelligently through a human environment, but multi agent systems can also benefit greatly from this work since if one agent updates the environment map in a certain region, another agent could never have

visited that region before, but still know how to navigate the most up to date map of this new environment intelligently.

2.3. Referenced Documents and Standards

About Ros. ROS.org. (n.d.). Retrieved from https://www.ros.org/about-ros/.

Institute of Electrical and Electronics Engineers. (2021, February 26). *IEEE 802.11-2020 - IEEE standard for Information Technology--Telecommunications and information exchange between systems - local and metropolitan area networks--specific requirements - part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications*. IEEE SA - The IEEE Standards Association - Home. Retrieved from https://standards.ieee.org/standard/802_11-2020.html.

Khoshelham, K., & Elberink, S. O. (2012). Accuracy and resolution of Kinect depth data for indoor mapping applications. *Sensors*, *12*(2), 1437–1454. https://doi.org/10.3390/s120201437

Khoshelham, K., & Zlatanova, S. (2016). Sensors for indoor mapping and navigation. *Sensors*, *16*(5), 655. https://doi.org/10.3390/s16050655

Kim, H.-D., Seo, S.-W., Jang, I.-hun, & Sim, K.-B. (2007). SLAM of mobile robot in the indoor environment with digital magnetic compass and ultrasonic sensors. *2007 International Conference on Control, Automation and Systems*. https://doi.org/10.1109/iccas.2007.4406885

Kim, S., Cheong, H., Park, J.-H., & Park, S.-K. (2009). Human augmented mapping for indoor environments using a stereo camera. *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*. https://doi.org/10.1109/iros.2009.5354128

Lu, X., Yi, Q., Jiangang, X., Yueguang, Z., Yuhu, W., & Zhenbo, W. (2009). A study on topographic map revision and virtual scene creation by integrating photograph, video and high resolution remote sensing imagery with virtual reality technology. *2009 International Forum on Information Technology and Applications*. https://doi.org/10.1109/ifita.2009.521

Why Gazebo? gazebo. (n.d.). Retrieved from http://gazebosim.org/.

Zlatanova, S., Sithole, G., Nakagawa, M., & Zhu, Q. (2013). Problems in indoor mapping and modelling. *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, *XL-4/W4*, 63–68. https://doi.org/10.5194/isprsarchives-xl-4-w4-63-2013

3. Operating Concept

3.1. Scope

The proposed systems in this project will provide a method to localize a robotic system indoors, even when parts of the indoor environment have changed compared to the time at which the initial map of the indoor environment was generated. The system in this project will be engineering to take small changes to the scale of moving around small pieces of furniture like chairs, stools, etc., as well as changes like doors being in a different state than when they were initially mapped. This algorithms in the system will not need Wi-Fi to function properly but the system itself will need Wi-Fi to send commands and receive data from the physical robotic system. This system will only use internal sensors to perform all of this functionality like Lidar, IR sensors, cameras, etc. and will not need any external locating sensors like GPS. This system will also most likely on work on 2D maps like 2D occupancy grids as environment maps instead of 3D versions of maps like 3D point clouds or other alternatives.

In addition to performing this localization, it will also update the indoor environment map with the most up to date data of the environment state in real time or near real time. This project will develop this system in software first, then implement it onto a small robotic system and test the success of this method in a small simulated indoor test environment as compared to other methods of acquiring updated environment data.

3.2. Operational Description and Constraints

To use this system, a ground truth map of the indoor environment must first be known, which is one operational constraint of this system. To obtain this ground truth map of the environment, the user can use a computer operated remote interface to move the robotic platform around and as the robotic platform moves around the indoor environment, it generates a map of the environment from its sensors and its movement data.

The robot can then be powered off and moved to a different location and/or have parts of the environment changed, like changing the state of a door or moving some furniture, and it will be able to localize itself within the previous map of the environment. The user will be able to see the robot's perception of where it is in the previously generated map and see how it adjusts as it gathers more data about the landmarks that it is near. As the user navigates around this new environment using the computer interface, the robot will localize itself better. Once the robot has reached a certain level of confidence that its localization is correct, it will then update the environment map with the new changes to the environment that it perceives so that the most up to data information about the environment is stored in the system.

3.3. System Description

To accomplish the goals set forth by this project, Unity or the Gazebo simulator will be used at first to simulate the robotics environment and allow for rapid iteration of the algorithms that I will be developing. In the simulation software, I will first build out a few environments in the

3D space where there are doors and pieces of furniture that can be moved around, allowing for a dynamic, human-like environment. I will then create a robotic mapping system within the software wherein a single agent robotic system will traverse the environment and map it thoroughly a few times, with no changes in the environment, in order to build the ground truth map of the environment that will be updated.

The second subsystem that I will then have to build out is to implement the functionality on the simulated robot to be turned on in a random part of the environment and for it to localize where it is within the environment, thus solving the robot kidnapping problem within this project and providing a base for any changes to the ground truth environment. This will most likely be done using previously researched solutions wherein a robotic system will identify big landmarks such as walls, cabinets, etc., and localize its location and direction based on the landmarks.

The third and final subsystem that I will have to build out in the simulator is the create the algorithm or system to take the new mapping data that is conflicting with the older mapping data and update the older map with the new data about where certain furniture has moved, where certain doors are closed or opened, etc. This work will build on top of the landmark identification on the previous system and most likely classify anything that is not a landmark as an object whose position can be updated for the purposes of this capstone project. This approach is a bit naïve, but intelligently identifying which objects can be moved intelligently and in what ways they can be moved is beyond the scope of this project.

The next phase of this project is the then implement this system in the real world and test it against other methods of mapping the environment and responding to changes.

To do this, a small robot with a LIDAR scanner or some other mapping system will be built and placed within a custom created miniature environment, mimicking a house layout, with doors and furniture. The previously mentioned algorithms that worked within the simulation will then be imported over to the small robot and once the small robot can fulfill all of the same functionality as in the simulation, my method will be tested against reacting to changes by simply remapping the entire environment, a naïve approach that is still used by many robotic systems today.

3.3.1. Figure 1: System Level Flow Chart

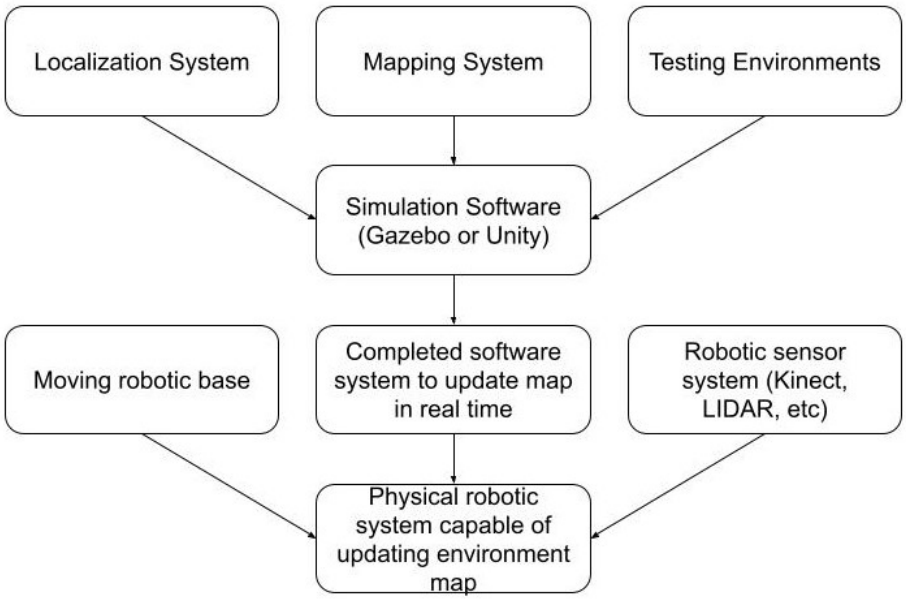

Figure 1: System Level Flow Chart

3.4. Modes of Operations

Initializing

This mode occurs whenever the robot is in a new environment and the user in control of the robot tells it to be in a mapping mode while the user is navigating around the environment. Here, the sensor data and movement data will be combined to generate a ground truth map of the environment that it is in.

Localizing

This mode occurs anytime the robot is initially powered on and it is not in the initializing mode. In this mode, the robot will recognize landmarks of the environment that it is in, even if substantial parts of the environment might have been changed. Using these recognized landmarks, the robot will be able to localize itself within the previously generated map of the indoor environment.

Updating

This mode occurs after the robot has been in the localizing mode for a while and it reaches a certain level of confidence in the results of its localization system. After this level of confidence is reached, the robot will then continue to localize itself within the environment, but it will also start updating the environment map with the real time changes that it sees in the environment.

3.5. Users

The target audience for this system is roboticists who will implement this system as a part of another larger system, possibly one with integrated indoor navigation modes, swarm robotics systems, etc. As such, a familiarity with robotics systems will be expected for installation and operation of this system, and a deeper understanding of algorithms and programming languages will be required to properly understand this system to integrate it. These users will be testing this system through the provided graphical user interface system on a computer.

3.6. Support

Instructions on how to install, set up, and run the purely software side of this project will be included in a user manual of sorts. Well documented code and a paper explaining the algorithms and process behind the system will also be provided. A paper explaining the process behind the integration of this system into an actual small robotic system will also be provided to show one possible implementation of this system to provide insight on how the software system can be integrated with other robotic systems.

4. Scenario(s)

4.1. Indoor Single Agent Navigation

In indoor systems where there is only one robotic agent, this system can help the single agent keep track of the changes to the environment, even as it navigates to another part of the indoor environment. For example, if there were a security guard robot for an apartment complex that would make the rounds every set amount of time, this system would allow for the robot to track many of the changes to the apartment such as opened doors, large pieces of trash or furniture that are in the hallway, etc, all while continuing to localize itself properly. In this use case, this would allow for tracking of activity and ensuring cleanliness and safety of the apartment complex.

4.2. Multi Agent Indoor Applications

In indoor systems where there are many robotic agents traversing the same indoor environment, this system can really shine and help all of the robotic agents as a swarm have the most up to date information about the environment to more efficiently and safely navigate the indoor environment. For example, if there were robotic hospital assistant robots that would go to certain rooms and give patients medication or check on the patients, there would be tens if not hundreds of robots per hospital performing these rounds. In a scenario like this, as one robot goes to a certain room, as it passes all the other rooms on its way to its destination room, it can update the state of the environment map as well as updating the map of the destination room, allowing other robotic agents that have not seen the changes to the environment to know about them before they even start navigation.

5. Analysis

5.1. Summary of Proposed Improvements

- Inherently solves the robot kidnapping problem wherein a robot might not know where in an environment it is if it is transported to an unknown location within the environment
- Adapts better to the robot kidnapping problem since it can localize itself even with changes in the environment.
- Instead of just perceiving the environment and simply adjusting to changes in the environment, it updates the environment map itself.

5.2. Disadvantages and Limitations

- Is limited in scope since it does not allow for autonomous gathering of additional data to improve fidelity of environment map.
- Will most likely only work on a 2D occupancy grid version of an environment map and not a 3D version using a point cloud or alternatives.
- Localization will be slower since it will take longer to latch onto landmarks and gain certainty in location.

5.3. Alternatives

Describe/list any alternative solutions and what any trade-offs may be to contrast your proposed project to the alternative.

- Localizing, but not updating map
    - Involves finding out where it is in an environment, does not update the environment map, leading to increasingly out of data perceptions of the environment as changes take place.
- Remapping the entire space autonomously
    - Involves localizing itself in an environment, but if it fails or if it succeeds but there are many changes that need updating, this system will trigger autonomous navigation of the indoor environment and remap the entire indoor space. This leads to the most up to date map but is very time and energy inefficient.

5.4. Impact

This system will increase the efficiency of current indoor robotic mapping systems since it allows for real time updating of the environment map instead of more energy inefficient operations. However, this type of indoor robotics is still much more energy inefficient than a system that relies on a distributed system of sensors built into the indoor space and if this type of system is widely adopted in the future, it may cause major environmental problems when compared to systems that are more specialized for certain use cases.

This project has great possibility for positive impact for public health since robotic systems like this could be used as assistants for doctors and nursing homes for elderly people. This system would allow large numbers of robotic agents to move about and monitor these patients and elderly people, give them medicine, and help them have more autonomy from their caretakers. However, there are safety and welfare considerations that must be taken into account since these types of robotic systems can be wrong sometimes and make mistakes and when working around patients and elderly people, these mistakes can mean a great deal of injury or emotional stress and terror.

This type of robotic system also has the possibility for great cultural and social impact since robotic systems that can perceive the environment well are the stuff of science fiction and it is what we have been working towards for a long time as a society now. This project takes a small step towards making some of that possible. When this does become possible, humanity will become more and more reliant on robotic systems to do their rote tasks for them, treating them like butlers and assistants in the physical world, much like there are many software assistants and helpers.

As for ethical concerns, there are none that arise from the base system itself, but in particular applications there may be room for concern due to the fact that this system allows for updating the perception of an indoor space much more easily and without notice when compared to other robotic systems. This may come with some concerns about invasion of privacy and data concerns.

REVISING ENVIRONMENT MAPS IN REAL TIME

Yashas Salankimatt

FUNCTIONAL SYSTEM REQUIREMENTS

REVISION – 2
30 April 2022

FUNCTIONAL SYSTEM REQUIREMENTS
FOR
Revising Environment Maps in Real Time

Team <83>

PREPARED BY:

Yashas Salankimatt 12/5/2021
Author Date

APPROVED BY:

Yashas Salankimatt 4/30/2022
Project Leader Date

_____
John Lusher, P.E. Date

_____
T/A Date

Change Record

Rev.	Date	Originator	Approvals	Description
-	10/3/2021	Yashas Salankimatt		Draft Release
1	12/5/2021	Yashas Salankimatt		Revision 1
2	4/20/2022	Yashas Salankimatt		Final Draft Revision

Table of Contents

List of Tables

List of Figures

1. Introduction

1.1. Purpose and Scope

As robots continue to be used indoors, robotic systems need to become more intelligent in their perception of indoor environments to improve upon navigation. Currently, many indoor navigation systems work on simple object avoidance rather than accounting for objects in the navigation plan itself and other systems that do map out their environment still have a difficult time keeping the map of their surroundings accurate due to the dynamic nature of indoor systems like doors, chairs, furniture, and other items moving around. With this system, robotic systems will be able to localize themselves within a ground truth map of the environment with no previous knowledge about their location and then update the ground truth map of the environment to improve navigation outcomes in the future and to have a more accurate perception of the surroundings.

This system will account for changes in indoor environments like small pieces of furniture (chairs, stools, coffee tables, etc.) being moved, as well as changes like doors being in different states than when they were originally mapped. This software system itself will not need Wi-Fi to perform any of its functionality, but this project will require Wi-Fi communication through ROS whenever the software system is implemented on a robot to offload the heavy duty computation off of the Raspberry Pi running on the physical robot. This system will only use internal sensors to perform all this functionality like LIDAR, IR sensors, cameras, etc. and will not need any external locating sensors like GPS. This system shall also most likely work on 2D maps like 2D occupancy grids as the environment maps instead of the 3D versions of maps like 3D point clouds or other alternatives.

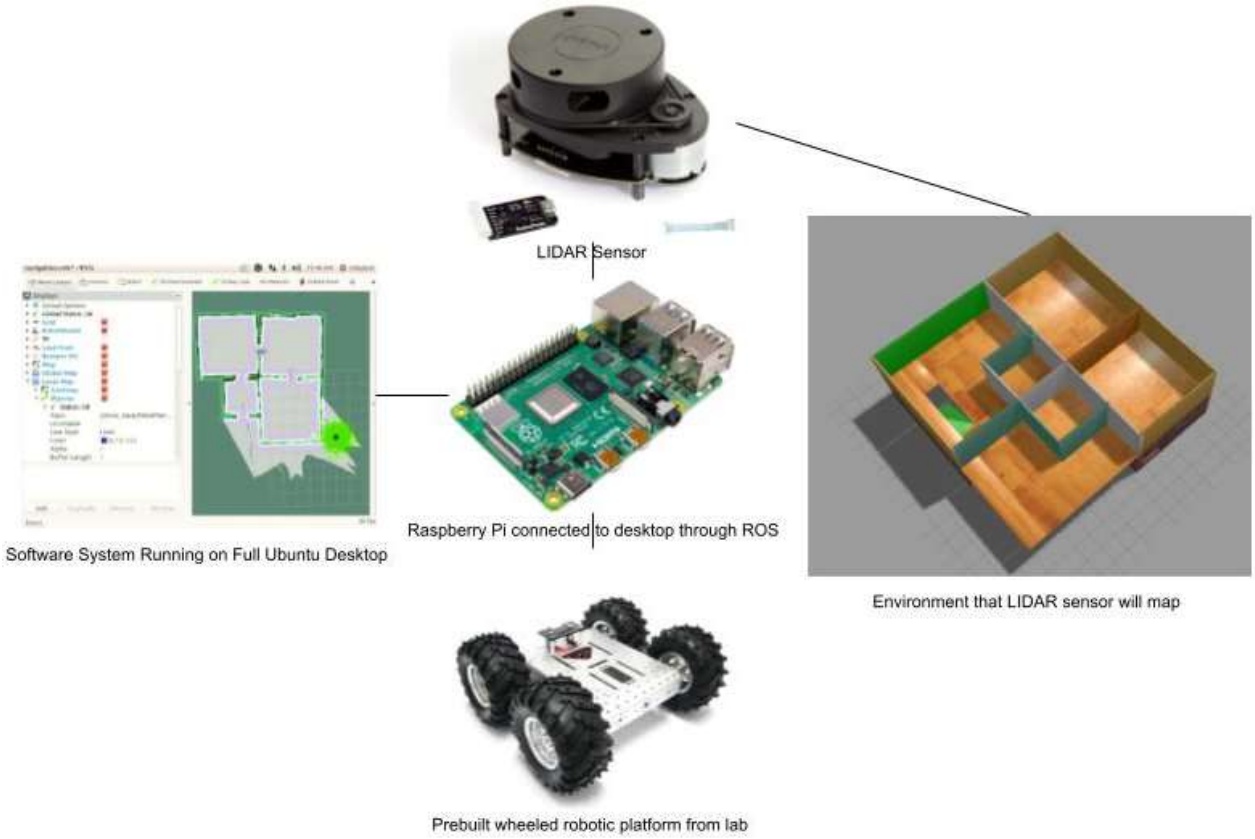

Figure 1. Your Project Conceptual Image

1.2. Responsibility and Change Authority

The sole member and team leader, Yashas Salankimatt, is responsible for verifying that all requirements of the project are met. Changes to the requirements, specifications, or scope fo the project can only be made with the approval of the team leader and the sponsor, Professor Stavros Kalafatis.

2. Applicable and Reference Documents

2.1. Applicable Documents

The following documents, of the exact issue and revision shown, form a part of this specification to the extent specified herein.

Document Number	Revision/Release Date	Document Title
IEEE 802.11	2020	IEEE Standard for Information Technology-Telecommunications and Information Exchange between Systems – Local and Metropolitan Area Networks- Specific Requirements
IEEE 1873-2015	2015	IEEE Standard for Robot Map Data Representation for Navigation
IEEE 1872.2-2021	2021	IEEE Approved Draft Standard for Autonomous Robotics (AuR) Ontology

Table 1: Applicable Documents

2.2. Reference Documents

The following documents are reference documents utilized in the development of this specification. These documents do not form a part of this specification and are not controlled by their reference herein.

Document Number	Revision/Release Date	Document Title
IEEE 1872.2-2021	2021	IEEE Approved Draft Standard for Autonomous Robotics (AuR) Ontology

Table 2: Reference Documents

2.3. Order of Precedence

In the event of a conflict between the text of this specification and an applicable document cited herein, the text of this specification takes precedence without any exceptions.

All specifications, standards, exhibits, drawings or other documents that are invoked as “applicable” in this specification are incorporated as cited. All documents that are referred to within an applicable report are considered to be for guidance and information only, except ICDs that have their relevant documents considered to be incorporated as cited.

3. Requirements

3.1. System Definition

This project describes a complete system to reliably locate a robotic agent within an indoor environment, even if the indoor environment has changed slightly. It also allows for the robotic agent to move around and gather more data and the system will allow the new data to override the old indoor map data. This system has 5 main subsystems: Localization, Mapping, Testing Environments, moving robotic base, and sensor systems. These subsystems are shown in the block diagram below. The localization system, mapping system and testing environments are connected to the simulation system through ROS software buses (nodes, topics, etc.). The physical robot, including motors and sensors are connected to the software system through a microcontroller on the robot that will be running ROS that will send over the sensor data and receive instructions on how the robot should move and what it should do. This way the robot does not have to perform any heavy duty computation on the onboard microcontroller.

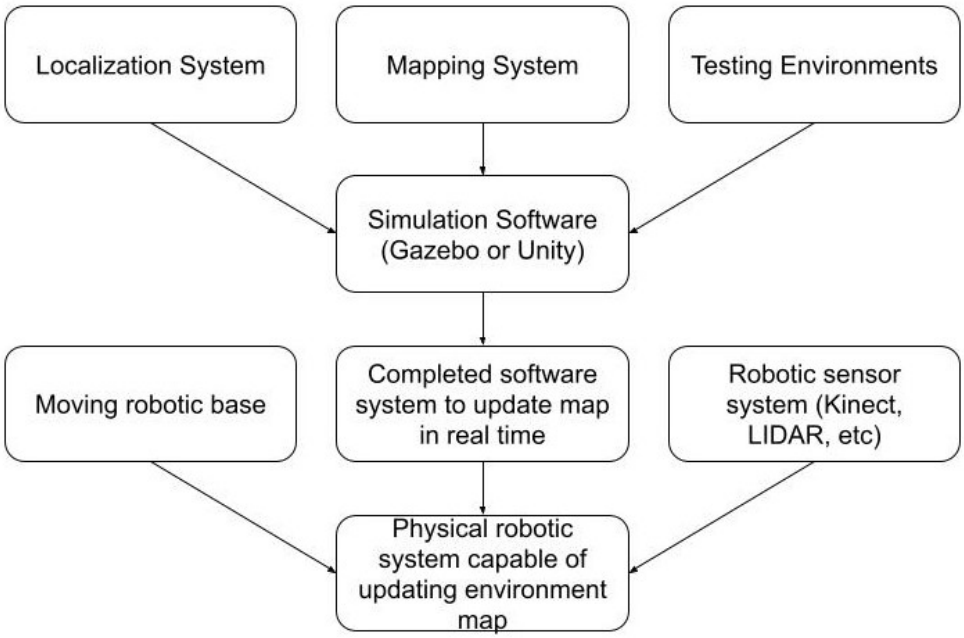

Figure 2. Block Diagram of System

The localization system will consist of the software subsystem that will take in the sensor input data as well as the previously accepted map of the surroundings and will then localize the robotic agent within the environment, even if certain small things have been changed about the environment, as noted before in the scope. The mapping system will consist of using the sensor input data mostly to first generate a map of a given indoor space and accepting that as the ground truth. Once that is done, during the primary use of the robot, this subsystem will also use the localization output as input as well as the sensor data and previous map in order to update the previous map with any new data about the environment that it is seeing. The final software subsystem is the testing environments. Since this system will need to be iterated upon many times, the initial development of the software system will

take place within the Gazebo simulation software, using ROS as a core part of it. As such, in order to test the previous two systems for proper functionality, there needs to be a robotic agent and environments for the robotic agent to traverse within the simulation software. This subsystem entails creating the robotic agent as well as the environments for the robotic agent to map and traverse.

The two physical systems are the moving robotic base and the sensor systems. The moving robotic base subsystem consists of using a previously built four wheeled robotic system from the capstone lab and using a Raspberry Pi and any additional hardware required to control the motors and make the robot base move around. This subsystem will also entail tracking the robotic agent using either encoders or timers for the motors so that the software system can have an accurate pose estimate for the robotic base. The sensor systems subsystem consists of attaching the LIDAR sensor to the moving robotic base as well as wiring it up to the Raspberry Pi. It also consists of setting up an instance of ROS on the Raspberry Pi to interface with the LIDAR sensor and convey the data back to the desktop computer doing the majority of the computing.

3.2. Characteristics

3.2.1. Functional / Performance Requirements

3.2.1.1. LIDAR Scanning Frequency

The LIDAR sensor shall scan at a minimum frequency of 5 Hz and a maximum frequency of 10 Hz.

> *Rationale: The more frequently data comes in from the LIDAR sensor, the more accurate the system can be, but if there is too much data, the computer will not be able to parse through all of the data and the network might not be able to communicate that much data in real time.*

3.2.1.2. LIDAR Scanning Range

The detection range of the LIDAR sensors used shall be a minimum of 5 meters.

> *Rationale: The LIDAR sensor must be able to gather data about its indoor environment and many larger rooms extend as far as 5-8 meters. As such, a detection range of at least 5 meters will allow the system to gather enough data to localize itself with respect to local landmarks within the indoor space.*

3.2.1.3. LIDAR Distance Accuracy

The LIDAR sensor shall have a distance reading accuracy of at least +/- 5 cm at a 5 meter distance and at least +/- 2 cm below a 2 meter distance.

> *Rationale: Getting consistent data is key to be able to localize the robotic agent properly and update the previous map, as such, these accuracy specifications should allow for enough resolution that the robotic agent should be able to pick up landmarks such as walls and large pieces of furniture properly for localization.*

3.2.1.4. Battery Operating Time

The battery operating time of the physical robotic agent shall be at least 15 minutes of driving time.

> *Rationale: The physical robotic agent needs to drive around the indoor space in order to create the indoor map and for testing purposes. As such, a battery operating time at 15 minutes with the speed specification listed below should supply more than enough time to conduct at least one testing run per charge.*

3.2.1.5. Time to Localization

The software system shall be able to localize the robotic agent within the previous map within 45 seconds of the software system being initialized.

> *Rationale: The robotic agent must be given time to move around slightly to acquire more data about its surroundings as well as identify landmarks in the environment to localize itself with certainty. As such, 45 seconds should be enough for this process to conclude in a confident localization result. This number was ascertained through trials that showed that 45 seconds was a maximum bound for the robot to be able to move around and conclude initial localization.*

3.2.1.6. Time to Map Updating

The software system shall be able to update the map of the indoor space with any new changes from the previous map within 30 seconds of the localization system concluding.

> *Rationale: The robotic agent must also be given some time here to move around the indoor environment and gather enough data about the changes in the space in order to update the map properly so that the quality of the indoor map does not degrade through the run cycles.*

3.2.1.7. Robotic Agent Speed

The speed of the robotic agent shall be at least 0.25 meters/second.

> *Rationale: The physical robotic agent needs to drive around through the indoor space for testing purposes. As such, this speed specification, combined with the battery operating time requirement specified above should provide more than enough time to conduct at least one testing run per charge.*

3.2.1.8. Communications Requirements

The physical robotic agent will communicate with the desktop computer doing most of the computing through a local area network, with the physical robotic agent being connected to the local area network through Wi-Fi.

> *Rationale: Communication from the robotic agent to the desktop computer happens through Wi-Fi, enabled and facilitated by ROS, running on each side.*

3.2.1.9. Magnitude of Change

The system developed shall be able to remap objects that have moved up to six feet from their original location shown in the previous robot map.

Rationale: Many human scale changes happen in ranges that are from 0 to 6 feet of change, like a door being opened or furniture being moved around slightly in day-to-day or hour-to-hour occurrences. Targeting this range of changes to remap allows for remapping most of the targeted human scale changes.

3.2.2. Physical Characteristics

3.2.2.1. Mass

The mass of the physical robotic agent shall be less than or equal to 25 kilograms.

Rationale: This allows for the physical robotic agent to be picked up and carried easily by one person to facilitate testing purposes.

3.2.2.2. Volume Envelope

The volume envelope of the physical robotic agent shall be less than or equal to 18 inches in height, 18 inches in width, and 18 inches in length.

Rationale: This is for the same reason as the mass requirement specified in 3.2.2.1.

3.2.3. Electrical Characteristics

3.2.3.1. Inputs

a. The presence or absence of any combination of the input signals in accordance with ICD specifications applied in any sequence shall not damage the software or physical robotic system, reduce its life expectancy, or cause any malfunction, either when the unit is powered or when it is not.

b. No sequence of command shall damage the software or physical robot system, reduce its life expectancy, or cause any malfunction.

Rationale: By design, should limit the chance of damage or malfunction by user/technician error.

3.2.3.2. LIDAR Input

The LIDAR sensor data shall serve as an input to the Raspberry Pi on the physical robot system and will be connected to the Raspberry Pi through a USB cable and a USB-serial converter dongle.

Rationale: This allows for proper power delivery to the LIDAR sensor through the converter dongle and provides useful debugging functionality since the primary scope of this project is the software system and not the actual construction of the physical robot.

3.2.3.3. Outputs

3.2.3.3.1 Process Output

The software system shall include an interface for viewing the output of the localization and mapping subsystems as well as the current state/mode the system is in.

Rationale: This is the primary view through which the user will see what the system is doing and where the robotic agent thinks it is in the indoor space.

3.2.3.3.2 Diagnostic Output

The software system shall include a diagnostic interface for error logging and fine-grained debugging.

Rationale: Provides the ability to control things for debugging manually and a way to view/download debugging data generated during runtime.

3.2.3.3.3 Raw LIDAR Output

The software system shall also provide an interface to show the raw LIDAR output for diagnostic purposes and manual sensor occlusion detection.

Rationale: Allows for debugging and diagnostics graphically without needing to interpret the runtime debugging data.

3.2.4. Environmental Requirements

The physical robotic system and the software system shall be run and stored in climate controlled indoor environments with further specifications provided below.

Rationale: The scope of this project is to map indoor systems, hence this requirement.

3.2.4.1. Terrain

The physical robotic system shall be run on smooth, indoor terrain such as wood, tile, or carpet and shall not be run on any terrain with particulates or dust or any terrain with drops or bumps of any kind.

Rationale: The scope of this project is to map indoor systems, hence this requirement.

3.2.4.2. External Contamination

The physical robotic system shall be run in indoor environments with no or very low levels of particulate matter in the air or on the ground and may not withstand dusty indoor environments since the LIDAR sensor and motors will not be enclosed in any protective casing.

Rationale: The scope of this project is to map indoor systems, hence this requirement.

3.2.5. Failure Propagation

3.2.5.1. LIDAR Sensor Error

The software system will check for proper operation of the LIDAR sensor by checking for occlusion of the sensor throughout runtime as well as checking for erroneous data points from distances that are further than the sensor's range.

> *Rationale: This will allow the system to know if something is blocking the LIDAR sensor or if the LIDAR sensor is trying to pick up and accept data that is not accurate.*

3.2.5.2. Motor Error

The physical robot system will check for proper operation of the motors through the use of position feedback, allowing for proper pose information as well as checking if both the motors are working properly.

> *Rationale: This will allow the system to ensure that the motors are actually spinning whenever the command to turn the motors is sent.*

3.2.5.3. Communication Error

The communication system will be checked for communication errors by verifying the data is arriving at the proper intervals, and operation of the system will be ceased to prevent errors or problems if this data arrival rate is too low on either side.

> *Rationale: This will ensure that the system is more impervious to flaky Wi-Fi connections and will stop the program if the connection drops out for a small duration of time.*

4. Support Requirements

The system will require a laptop with a CPU with at least 4 cores, running Ubuntu 20.04. This is because many ROS nodes make use of multiple cores and this will allow for the software systems to run at the speed required to also run the custom code and systems that will be created in this project. The system will include the physical robotic system with the LIDAR equipped robotic agent running ROS. Issues in the field will be resolved by a robust support manual detailing use of the program as well as detailing the source code behind the program. A paper explaining the algorithms and processes behind the system will be provided as well as a paper explaining the process behind the integration of the software system into the physical robotic system to show one possible implementation of the software system to provide insight on how the software system can be integrated with other robotic systems.

Appendix A: Acronyms and Abbreviations

GPS	Global Positioning System
GUI	Graphical User Interface
Hz	Hertz
ICD	Interface Control Document
TBD	To Be Determined
USB	Universal Serial Bus
LIDAR	Light Detection and Ranging
ROS	Robot Operating System

Appendix B: Definition of Terms

N/A

REVISING ENVIRONMENT MAPS IN REAL TIME

Yashas Salankimatt

INTERFACE CONTROL DOCUMENT

REVISION – 2
30 April 2022

INTERFACE CONTROL DOCUMENT
FOR
Revising Environment Maps in Real Time

PREPARED BY:

Yashas Salankimatt 4/30/2022
Author Date

APPROVED BY:

Yashas Salankimatt 4/30/2022
Project Leader Date

_____
John Lusher II, P.E. Date

_____
T/A Date

Change Record

Rev.	Date	Originator	Approvals	Description
-	10/3/2021	Yashas Salankimatt		Draft Release
1	12/5/2021	Yashas Salankimatt		Revision 1
2	4/30/2022	Yashas Salankimatt		Final Draft Revision

Table of Contents

List of Tables

No table of figures entries found.

List of Figures

No table of figures entries found.

1. Overview

This Interface Control Document (ICD) for this project will provide more detail on how the subsystems in the Concept of Operations Report and the Functional System Requirements Document will be produced and linked to each other. This document will provide a physical description of the physical robotic agent that will be used in this project as well as the electrical interface of the robotic agent with the motors, LIDAR sensor, position feedback devices, etc. This document will also document how the physical robotic agent communicates with the desktop computer running the visualization and compute software.

2. References and Definitions

2.1. References

Please refer to sections 2.1 and 2.2 of the Function System Requirements document.

2.2. Definitions

GPS	Global Positioning System
GUI	Graphical User Interface
Hz	Hertz
ICD	Interface Control Document
TBD	To Be Determined
USB	Universal Serial Bus
LIDAR	Light Detection and Ranging
ROS	Robot Operating System
TTL	Transistor-Transistor Login
UART	Universally Asynchronous Receiver/Transmitter
SPI	Serial Peripheral Interface
SSH	Secure Shell
GPIO	General Purpose Input/Output

3. Physical Interface

The Husarion ROSBOT 2.0 will be used for this project as a prebuilt robotic agent. As such, this prebuilt robot follows all of the specifications needed for this project shown below. Since the primary focus of this project is software based, this robot will be used as built and will not be modified or physically interfaced with in any custom way.

3.1. Weight

The mass of the physical robotic agent, including motors, sensors, microcontroller, and battery shall be less than or equal to 25 kilograms.

3.2. Dimensions

The volume envelope of the physical robotic agent shall be less than or equal to 18 inches in height, 18 inches in width, and 18 inches in length.

The LIDAR sensor shall be no greater than 6 inches in width, 6 inches in length, and 3 inches in height.

3.3. Mounting

Since a prebuilt robotic agent will be used for this project, there will be no custom mounting necessary for this project as the microcontroller, motors, motor drivers, batter, and LIDAR sensor have already been mounted in place on the ROSBOT 2.0.

4. Thermal Interface

The microcontroller will use heatsinks in combination with a small fan attached to the main processor unit as well as the memory modules to dissipate heat and sustain performance. The motors in the system as well as any motor controllers and batteries are designed to be run with passive cooling and will not need any thermal interfacing.

5. Electrical Interface

5.1. Primary Input Power

The power for the robotic agent will be provided by a lithium polymer battery cell internal to the ROSbot 2.0. This battery cell will most likely provide 11.1 Volt power and will be recharged with an external recharging station. This battery based power source will already be interfaced with the motor controllers since this project will be using a prebuilt robot.

5.2. Signal Interfaces

The microcontroller most likely interfaces with the motors through a PWM signal to control their speed and a direction pin to control their spin direction. It will also most likely interface with the position feedback devices using one of its General Purpose Input/Output (GPIO) pins and a standard communication protocol like SPI or UART, depending on the type of position feedback device. The specification for this is unknown since Husarion does not provide information about how their robot interfaces its custom microcontroller with the motor controllers and the motor.

The LIDAR sensor requires 5 Volt power and 3V TTL serial logic. Since this part of the project is not the primary purpose of the research, a USB-serial converter dongle will be used to provide clean power and communication.

5.3. User Control Interface

The user will control these systems from the desktop computer running Ubuntu. The GUI will be generated by ROS and the Raspberry Pi status can be checked and modified using SSH.

6. Communications / Device Interface Protocols

6.1. Wireless Communications (WiFi)

The single board computer that will be used on the physical robotic agent is equipped with a Wi-Fi module supporting the 802.11ac standard. This connection will be used to connect to a local area network (LAN) that the desktop computer also shares to run ROS on both machines and transmit and receive data between the two machines.

6.2. Host Device

The on-board computer/microcontroller will interface with the LIDAR sensor through its USB port through which a USB-serial converter dongle will be plugged in to provide an easy interface with the RPi to interpret the LIDAR data.

6.3. Simulation Interface

During the simulation and development phase of the software side of the project, the ROS library will be heavily used in conjunction with the Gazebo simulation software in lieu of using an actual robotic agent for testing. Maps of real world environments will be built in the Gazebo simulation software through the provided tools and a model of the physical robotic agent and the sensors will be created in Gazebo to mimic the real world testing scenario as closely as possible. The primary algorithm development will then happen in ROS with the simulation in Gazebo. Once the development is done, since the development and physical robot both use ROS, porting the software system to a physical robot should be much easier.

7. Execution and Validation Plan

7.1. Execution Plan- ECEN 403

Task	Date Due	Status
ConOps Report Completed and Submitted	9/13/2021	Done
FSR Completed and Submitted	10/4/2021	Done
ICD Completed and Submitted	10/4/2021	Done
Midterm Presentation Completed and Submitted	10/6/2021	Done
Complete all ROS Basic Tutorials on ROS Website	10/11/2021	Done
Complete all Gazebo Sim Basic Tutorials on Gazebo website	10/11/2021	Done
Create testing environments in Gazebo Simulator	10/15/2021	Done
Finalize robotic agent, determine dimensions and sensors	10/15/2021	Done
Finalize LIDAR Sensor Model/Make and Order/Acquire	10/15/2021	Done
Model robotic agent movement systems in ROS/Gazebo, ensuring that the robot can be moved around with the keyboard or a program	10/20/2021	Done
Model LIDAR sensor in ROS/Gazebo, ensuring that it can see the virtual environment with the same specificity as the actual LIDAR sensor	10/20/2021	Done
Complete pseudocode of localization system	10/29/2021	Done
Complete pseudocode of mapping system	10/29/2021	Done
Finish first implementation of localization system	11/5/2021	Done
Finish revised/finished localization system	11/19/2021	Done
Finish first implementation of mapping system	11/12/2021	Done
Finish revised/finished mapping system	11/19/2021	Done
Comparison tests in Gazebo between my system and existing systems run and data is gathered	11/26/2021	Done
Final presentation Completed and Submitted	12/1/2021	Done

7.2. Execution Plan- ECEN 404

7.3. Validation Plan

Paragraph #	Test Name	Success Criteria	Methodology	Status
3.2.1.1	LIDAR Scanning Frequency	Can scan environment at minimum frequency of 5 Hz	Self test and check of single point data update periods	VALIDATED
3.2.1.2	LIDAR Scanning Range	Detection range is a minimum of 5 meters	Object place outside of 5 meter range and whether it detects or not is observed	VALIDATED
3.2.1.3	LIDAR Distance Accuracy	Error within 5cm at 5m and 2cm at 2m distances	ROSbot distance from wall evaluated at these two distances with a tape measure and checked against LIDAR data	VALIDATED
3.2.1.4	Battery Operating Time	At least 15 minutes of driving and operating time	Driving and operating time measured with stopwatch	VALIDATED
3.2.1.5	Time to Localization	Localization achieved in 45 seconds of initialization	Localization time measured from initialization with stopwatch	VALIDATED
3.2.1.6	Time to Map Updating	Map updating achieved in 30 seconds of localization concluding	Time to map update measured with stopwatch after localization completed	VALIDATED
3.2.1.7	Robotic Agent Speed	Can move at least 0.25 m/s	ROSbot will be made to move known distance and timed, from there, actual velocity will be determined	VALIDATED
3.2.1.8	Communication Requirements	Should send data and receive commands/computations with Wi-Fi	Host computer will check if receiving data from ROSbot using Wi-Fi through the LAN	VALIDATED
3.2.1.9	Magnitude of Change	Should be able to remap with changes of up to 6 feet from previous location	ROSbot will map environment and then object in environment will be moved 2-6 ft and ROSbot should be able to remap and erase old data and add new location	VALIDATED
3.2.2.1	Mass	Should be less than or equal to 25kg	Will be weighed using a scale	VALIDATED
3.2.2.2	Volume Envelope	Should be 18 inches at maximum in each dimension	Will be measured using a tape measure	VALIDATED
3.2.4.1	Terrain	Should run without problem on smooth, indoor terrain	Will be run on hardwood and carpet flooring and proper operation without error will be observed for	VALIDATED
3.2.4.2	External Contamination	Should run without problem in indoor environments with no or low levels of particulate matter in air or on the ground	Will be run in environments without particulate matter contamination and operation without error will be observed for	VALIDATED
3.2.5.1	LIDAR Sensor Error	Is able to identify if something is blocking the LIDAR sensor	An object will be made to block the LIDAR sensor and whether this system will detect the occlusion or not will be noted	VALIDATED
3.2.5.2	Motor Error	Is able to identify stalled or improperly running motors	Motor will be manually slowed and stalled by hand and system detection of this stall will be noted	VALIDATED
3.2.5.3	Communication Error	Is able to identify communication errors and disregard periods of time with spotty connections with host computer	Midway through operation, the connection with host computer will be severed, and each side must recognize this and throw the proper error	VALIDATED

REVISING ENVIRONMENT MAPS IN REAL TIME

Yashas Salankimatt

SUBSYSTEMS REPORT

SUBSYSTEMS REPORT
FOR
Revising Environment Maps in Real Time

TEAM <83>

APPROVED BY:

Yashas Salankimatt 4/30/2022
Project Leader Date

_____
Prof. Kalafatis Date

_____
T/A Date

Change Record

Rev.	Date	Originator	Approvals	Description
-	12/5/2021	Yashas Salankimatt		Draft Release
1	4/30/2022	Yashas Salankimatt		Final Report Revision

Table of Contents

List of Tables

List of Figures

1. Introduction

This project is about a system that is able to map out an indoor system with a 2D LIDAR sensor mounted to a batter powered moving robotic base, and then remap the environment and localize itself within the environment even whenever there are small changes to the environment like chairs, stools, coffee tables, and doors moving. This is since these are the changes that most often occur in human systems and by being able to track and perceive these changes well, a whole suite of monitoring tools for buildings for security as well as for medical purposes becomes available.

To monitor these micro changes, we need a few subsystems that have been built out in this project. First, we need the testing environment itself. Since this is a software system that needs to be iterated many times, simulation is key, and simulation allows for rapid testing and iteration of the software. As such, the testing environment itself needs to be built, including the base environment as well as the furniture to populate the environment. Then robotic agent model must also be constructed so that the robot can move around the environment and have a simulated LIDAR sensor that can roughly represent the real thing. Once these are built, the mapping system includes the system to create a complete map of the indoor environment as a base and includes the system to map out any changes to the environment. The remapping part of this subsystem involves reading the ground truth and the map of the changes and producing a map with the old data removed and only the changes internalized so that a complete remapping of the environment is not necessary. The final system is the localization system since the robot also needs to know where it is despite these micro, human changes, a landmark based approach to localization must by implemented and tested. More details about these subsystems are provided in the rest of the report. A more graphical representation of the system and its subsystems is shown below.

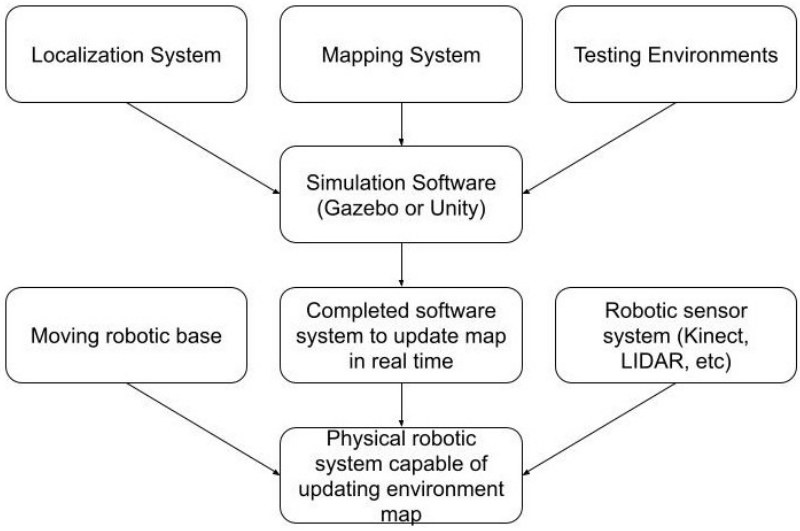

Figure 1. Block Diagram of System

2. Simulation Subsystem Report

2.1. *Introduction*

To create a robotic software system, testing is key, but iteration times in the real world are long, and running untested code on physical robots can be quite dangerous. In order to address both of these problems, this software project was tested in a simulation environment. This meant that iteration times were much faster, the raw code could be tested in isolation from real world confounding factors, and any dangerous factors were avoided during the development of the system.

There are a few main parts of the simulation subsystem. There is the testing environment itself, which encompasses the actual bare indoor environment itself as well as the furniture to populate it that will be moved around to simulate changes that would happen on a day-to-day basis with human use.

There is also the robotic agent model, which has to simulate the prebuilt robot that will be used in the physical world as well as its movement speed, LIDAR sensor specifications, etc. This robotic agent must also be integrated with ROS, since the agent in simulation will need to interact with the code and systems created, just like the physical robot that will be used in this project, the Husarion ROSBOT 2.0.

Finally, there is the ROS Robotic System setup. ROS, or the Robot Operating System, is a framework that is used for many robotics uses and can handle a great deal of modularity when it comes to robotics uses. This is what allows all of the subsystem to exist and work together, otherwise, integration manually would be a nightmare. As such, when developing with ROS, the setup and installation of the library itself as well as the setup of the project and the development environment, including scripts, launch files, node structures, etc. is very important.

2.2. *Testing Environments*

When creating the testing environment special care had to be made in order to make sure that indoor environments were being properly represented in terms of size as well as the types of objects that humans interact with and move on a daily/hourly basis.

As such, the two environments that were built in this project were based on apartments that I had lived in. The bare environment was created using the floor plans of the apartments and the furniture models were created using common dimensions of furniture found on furniture sale online stores.

Special attention to detail needed to be had for the furniture because since the robot was very low the ground, its 2D LIDAR sensor would only be able to pick up the bottom parts of the furniture and it is not feasible to turn a 2D LIDAR sensor to capture 3D data.

2.2.1. Validation

The environments created are shown below. The top image is of environment 1, without any furniture populated, and the bottom image is of environment 2, with furniture populated.

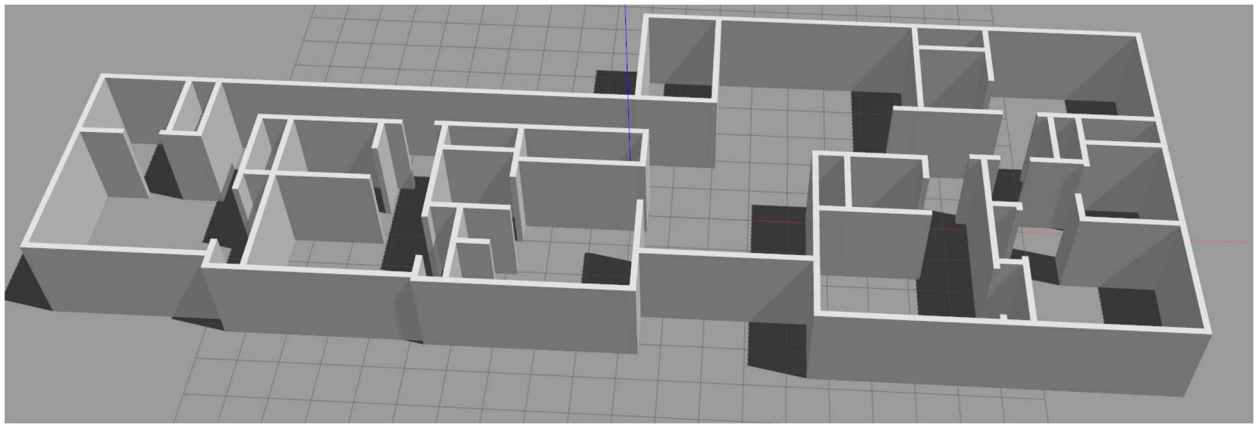

Figure 2. Environment 1 Bare Model, no furniture populated

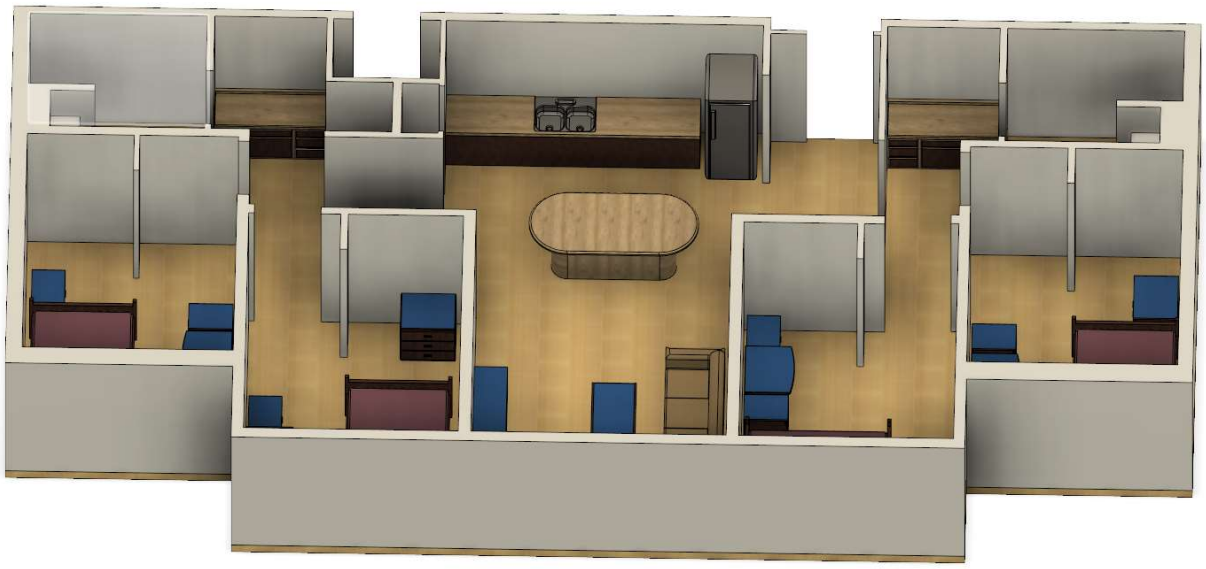

Figure 3. Environment 3 Model with furniture populated

2.3. *Robotic Agent Model Construction*

The next stage of preparing the simulation environment was the create a version of the actual physical robot itself in the simulation environment. In this project, the physical robot that was assigned to be used was the Husarion ROSBOT 2.0, shown below in Figure 4. This robot has an RPLIDAR on it and 4 wheels with skid steering. As such, I needed to both create the physical robot itself in the simulation environment as well as define its movement and the correct parameters for the LIDAR sensor so that it would most accurately represent the way the LIDAR sensor would capture data in the real world. Below, are the 3D models of the various parts of the robot that were used to make up the final robot in the simulation software.

Figure 4. Husarion ROSBOT 2.0 Actual Robot

2.3.1. Validation

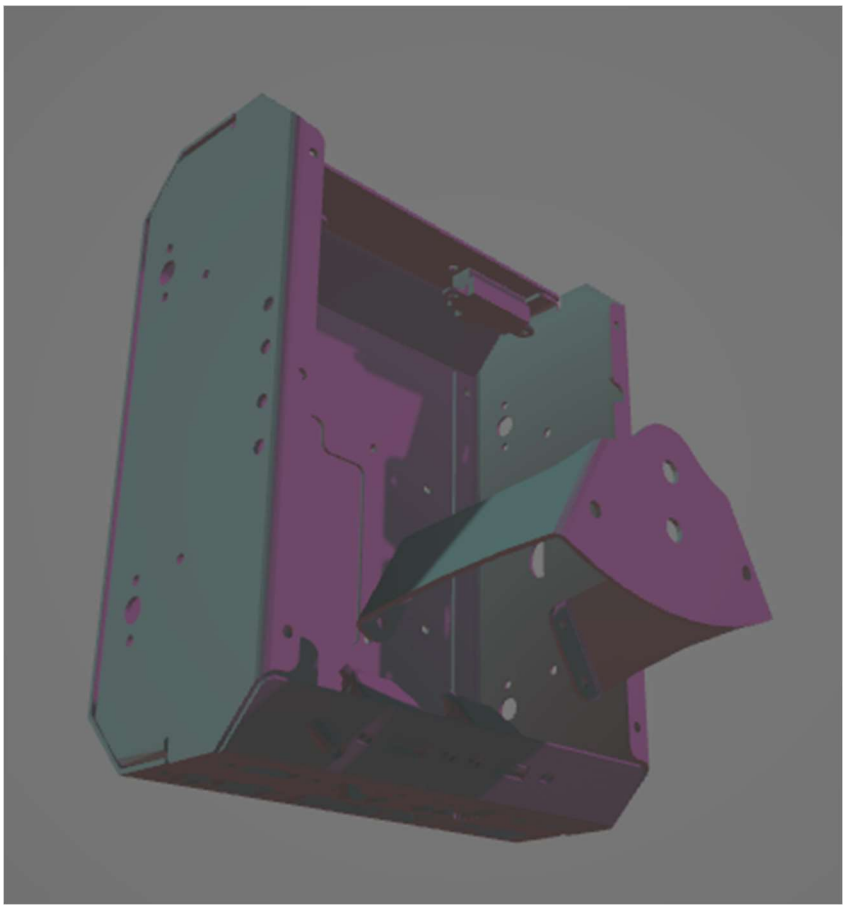

Figure 5. ROSBOT main lower body piece

Figure 6. ROSBOT main body upper plate piece

Figure 7. ROSBOT RPLIDAR 3D Model

2.4. *ROS Robotic System Setup*

The final and most important part of the simulation system that needed to be built out was the ROS robotic system, including putting together all of the 3D models of the testing environment, the robotic agent, tying in sensor data, etc. To get into the ways that ROS does all of this would be beyond the scope of this report, but the basic idea is that ROS uses nodes and topics to communicate between different devices. For example, there could be a mapping node that subscribes to a topic from the LIDAR node, and the mapping node could then publish a topic with its resulting data for any further nodes to use. The setup and interplay of these nodes is controlled by Linux shell script files that have to be sourced at launch, as well as launch files that sequentially launch other launch files or nodes that then use world files (files that contain information about the environment that needs to be opened) and/or URDF (Universal Robot Description file) files (these provide details on the physical robotic agent like the specifications of the LIDAR sensor or the way that the robot will move/interact with other robot pieces). There are hundreds of these files that were created for this project and this interplay proved to be difficult to manage as the complexity of this project went up. These files can be found in the link in the appendix. Examples of one launch file (of which there are hundreds that call each other) as well as a snippet of a URDF file is provided below.

```
<gazebo reference="range_rr">
  <sensor type="ray" name="range_rr">
    <pose>0 0 0 0 0 0</pose>
    <update_rate>5</update_rate>
    <ray>
      <scan>
        <horizontal>
          <samples>5</samples>
          <resolution>1.0</resolution>
          <min_angle>-0.05</min_angle>
          <max_angle>0.05</max_angle>
        </horizontal>
        <vertical>
          <samples>5</samples>
          <resolution>1</resolution>
          <min_angle>-0.05</min_angle>
          <max_angle>0.05</max_angle>
        </vertical>
      </scan>
      <range>
        <min>0.01</min>
        <max>0.9</max>
        <resolution>0.01</resolution>
      </range>
    </ray>
    <plugin filename="libgazebo_ros_range.so" name="gazebo_ros_range">
      <gaussianNoise>0.005</gaussianNoise>
      <alwaysOn>true</alwaysOn>
      <updateRate>5</updateRate>
      <topicName>range/rr</topicName>
      <frameName>range_rr</frameName>
      <fov>0.1</fov>
      <radiation>ultrasound</radiation>
    </plugin>
  </sensor>
</gazebo>
```

Figure 8. Snippet of the URDF file that describes the LIDAR sensor

```xml
<?xml version="1.0" encoding="UTF-8"?>
<launch>

  <arg name="world" default="empty"/>
  <arg name="paused" default="false"/>
  <arg name="use_sim_time" default="true"/>
  <arg name="gui" default="true"/>
  <arg name="headless" default="false"/>
  <arg name="debug" default="false"/>

  <include file="$(find gazebo_ros)/launch/empty_world.launch">
    <arg name="world_name" value="$(find rosbot_gazebo)/worlds/whitecreek_apartment.world"/>
    <arg name="paused" value="$(arg paused)"/>
    <arg name="use_sim_time" value="$(arg use_sim_time)"/>
    <arg name="gui" value="$(arg gui)"/>
    <arg name="headless" value="$(arg headless)"/>
    <arg name="debug" value="$(arg debug)"/>
  </include>

</launch>
```

Figure 9. Lowest level launch file that launches the apartment model created in Gazebo, the simulation software, there are many launch files above this, organized in a tree structure, each one calling more launch files or processes/nodes

2.5. *Conclusion*

The simulation environment was crucial to be built out properly in order to keep the iteration time low and make sure that the time to port over the software from simulation to the real world would be as low as possible. I believe that from the validation results above as well as the demo videos that can be found in the appendix that this goal was achieved.

3. Mapping Subsystem Report

3.1. *Introduction*

The first of two main software subsystems involved in this project was the mapping subsystem. The mapping subsystem is responsible for creating a complete map of the indoor environment with user-controlled robot movement and being able to save it to a file to be processed and/or used later as a base. The mapping subsystem is also responsible for mapping out any micro, human scale changes that may have happened to the environment as described earlier and then also saving that to a file to be processed.

The mapping subsystem then takes the two files, one of the ground truth data of the environment, and the other of the ground truth data in addition to the changes that the environment may have witnessed and filtering out the old data, keeping the landmark data from the ground truth map and the new furniture position data from the new map. A demonstration of this can be seen in the link in the Appendix.

3.2. *LIDAR Based SLAM Environment Mapping*

For the robot to be able to map the base environment, it must be able to both move around the environment as well as collect data points using its LIDAR sensor and put it all together. This is a process known as SLAM (Simultaneous Localization and Mapping). In order to be able to do this, a ROS node was created to be able to control the robot with a keyboard, adjusting its linear and angular speed as well as being able to move it forward, backward, left and right in the ways that the actual physical robot would be able to, using skid steering.

Once this was done, a launch file was created that would launch a series of nodes, the most important ones being: the RViz node/program, that would visualize the data as we received it and mapped it, the Gazebo simulation software loaded with a model of our choosing with the robot model loaded in there waiting for movement instructions and publishing its LIDAR data, the teleoperation node to control the robot, and a custom configured SLAM node called gmapping that would be able to take into account the robot’s pose, movement and laser LIDAR data and put it all together to create a map.

As validation, shown below is an example of the mapping system, as well as a fully completed map.

3.2.1. Validation

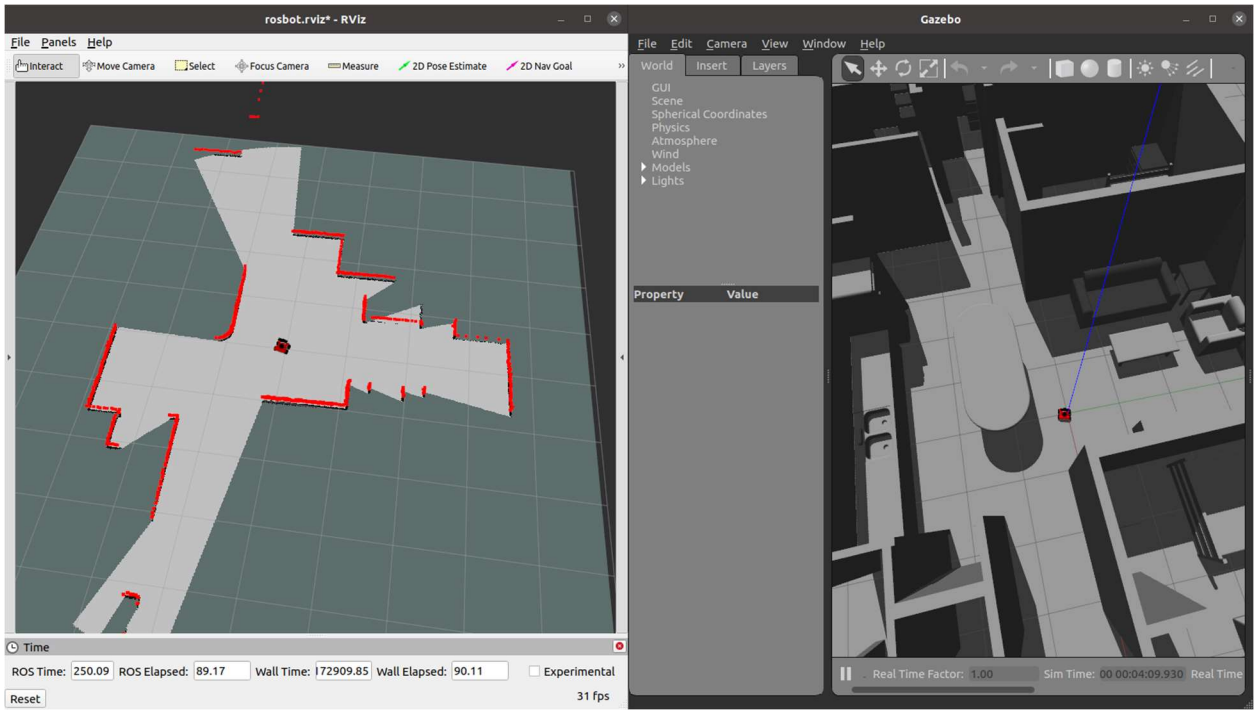

Figure 10. Mapping system shown with RViz on left and Gazebo on right

Figure 11. Completed map- result of mapping system and teleoperation of robot

Specification	Requirements	Results (average)	Status
SLAM Map Update Time	< 1s	.66s (25 runs)	PASS
Acceptable Map Error	< 5 cm	3.6 cm (10 points)	PASS

Table 1. Mapping System Base Validation Data

3.3. *Generated Map Saving and Format*

The generated map is then saved using another ROS launch file that launches many other nodes, but the most important of which is a node called map_server, with the map saver sub-process called. When this node is called whenever the mapping system is still running, it will take all of the generated and saved occupancy map data and write it to a file in the PGM format.

This format is essentially just an image file, but grayscale and provides all of the values of the image pixel by pixel from 0-255 with gray being unmapped points, black being mapped and occupied points, and white being mapped and unoccupied points. It write this data in a binary format with a PGM version of P5. The importance of these factors will become clear during the explanation of the remapping system. The validation of this part simply involves showing the saved PGM files and since there is more processing to be done of these files in the remapping system, the validation of this part will be shown below in the validation section.

4. Localization Subsystem Report

4.1. *Introduction*

The localization subsystem is responsible for taking in ground truth map data and still being able to figure out where in the environment the robot is based on the LIDAR data, even if there are changes to the environment.

4.2. *Base Localization from Previous Environment Map*

To perform the base localization, a probabilistic localization system is implemented in ROS, where probabilistic localization is a type of localization that uses many previous robot locations in conjunction with the robot pose and movement as well the map data to figure out where in the map it is.

In this project, the probabilistic localization model that was implemented was called AMCL that is a system for a robot moving in 2D and this uses the adaptive Monte-Carlo probabilistic localization approach that uses a particle filter to track the post of the robot against the ground truth map. It is in this particle filter that the localization is made to be more change-blind and this will be talked about in section 4.4.

4.3. *2D Nav Goals*

The localization system, as described above, works by knowing many previous locations and positions of the robot in addition to the ground truth map and the LIDAR data. As such, for this system to work, I had to implement a system for the robot to be able to gather more data to make the localization work. The problem with this is that we cannot use teleoperation methods since forward will keep changing. Hence, a system where a 2D navigation goal is placed and the robot iteratively keeps trying to move towards that navigation goal was implemented, and as such, the localization algorithm would be able to acquire more data.

4.4. *Making the Localization More Change-Blind*

In order to make the localization more change-blind, it must be first made to focus more on the landmarks in an environment and much less on the smaller pieces of furniture that may be moving. As such, we use many filtering methods.

The method of beam skipping was implemented, as well as rejecting LIDAR data points that were too close or too far for reliability purposes. We also weighted points from the LIDAR that are part of longer adjacency chains, where they have close neighbors. We also account for groups and adjacency chains with more raw data points and weight those more and we discount small groups that are far away from other groups. The final result of all of this can be seen in the demo video in the appendix and some screenshots can be seen below.

4.5. *Validation*

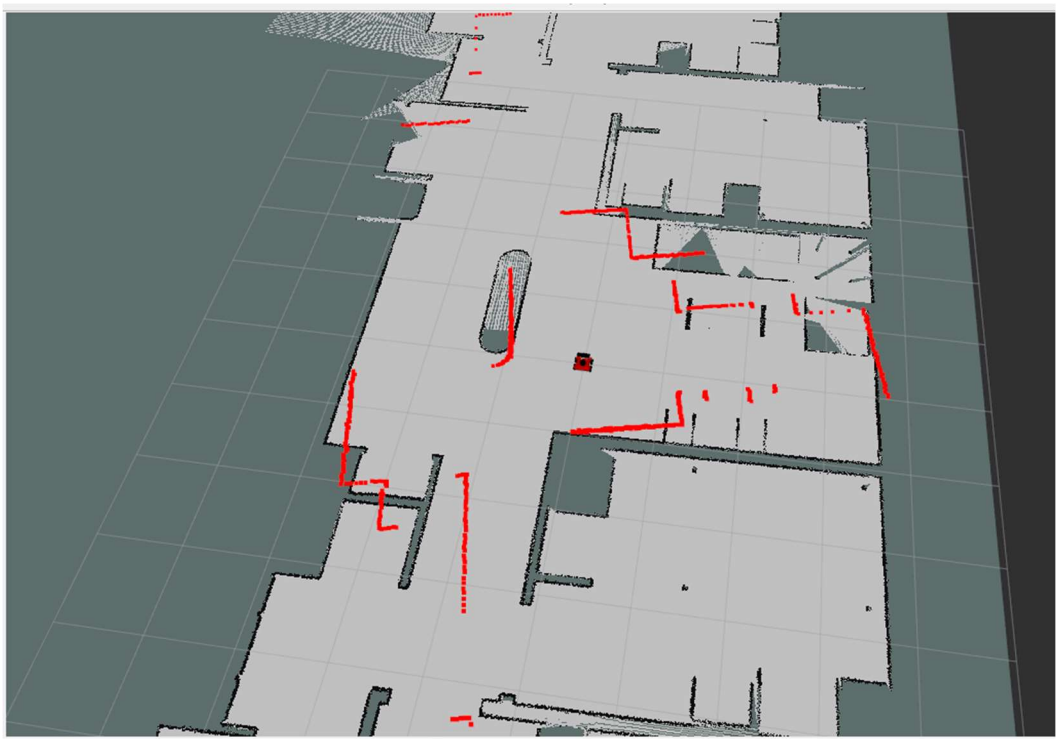

Figure 12. View of RViz on launch, pre-localization

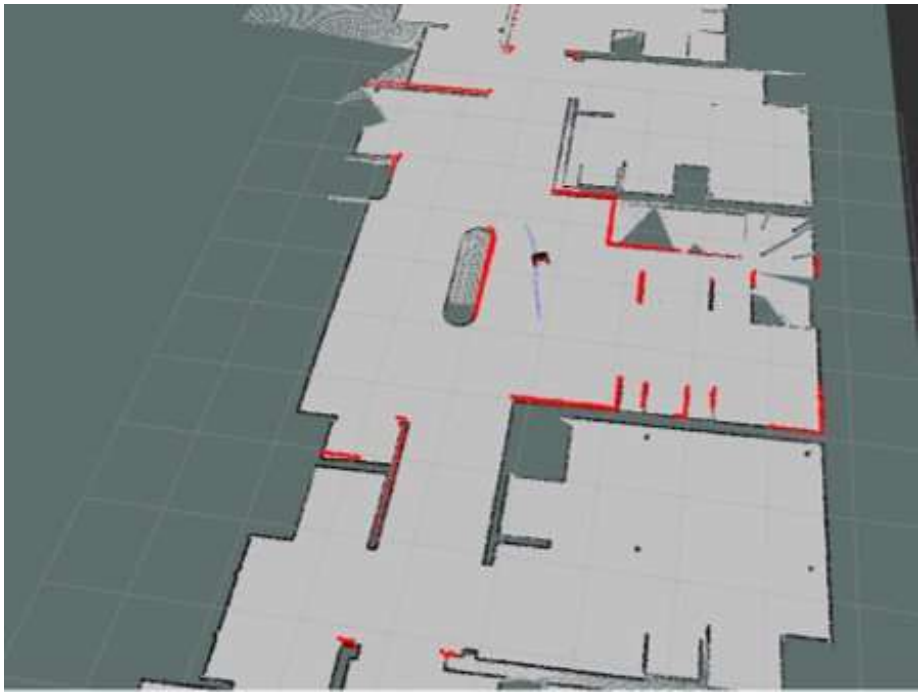

Figure 13. View of RViz after localization, this environment has no changes to it

The above post localization validation image shows an environment with no changes compared to the ground truth map. For validation of the localization system working with changes to the environment, see the video linked in the Appendix.

Shown below is the data for the base localization implementation compared to my custom implementation for various levels of changes to the environment. Each point is an average of 5 runs.

	Base Implementation	My Implementation
No Changes	13.7 s	16.8 s
Small Changes	66.9 s	25.4 s
Larger Changes	DNF	38.9 s

Table 2. Localization Subsystem Data

5. Remapping Subsystem Report

5.1. *Introduction*

The remapping system is responsible for knowing the state of the environment previously and taking in new data about the environment and saving the changes that it can see to the environment into the previously generated map, thus revising the map instead of generating a whole new one. This is done by taking in a previously generated map of the environment from the mapping subsystem as well as new, localized input from the sensors about the state of the environment with small (2-6 feet) human scale changes and integrating the two, computing the overlap and keeping the areas where there is overlap the same and making decisions about how to revise the map.

5.2. *ECEN 403 Simulation Based Remapping System*

The remapping system that was developed in ECEN 403 is responsible for taking in the ground truth data of the environment in the form of the PGM map described above, as well as the data of the changes to the system that it sees on top of the ground truth. Examples of the data in the form of the generated PGM maps is shown below.

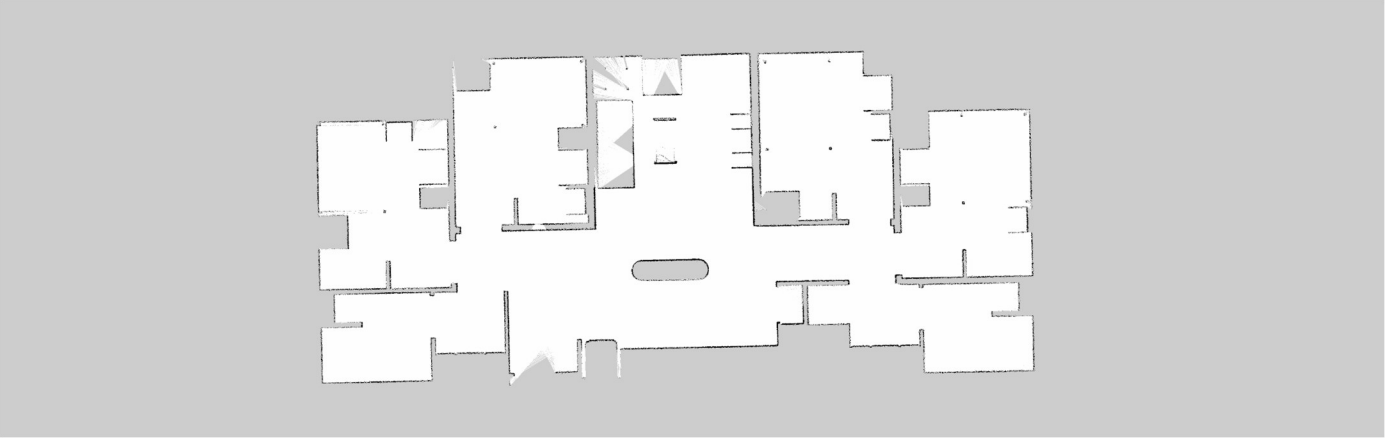

Figure 14. Ground truth map data

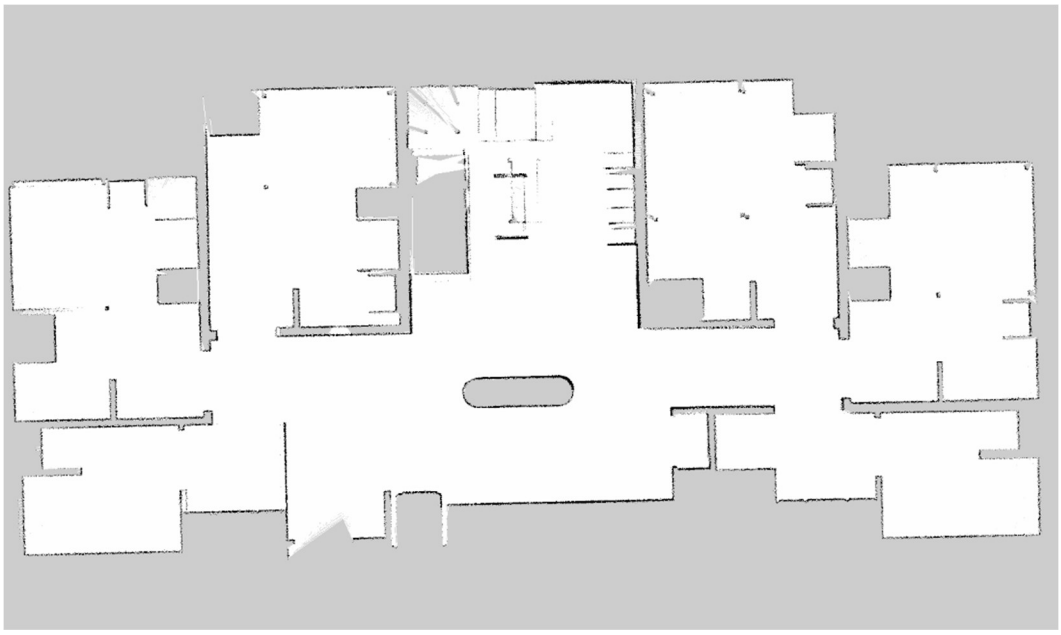

Figure 15. Pre-remapping map data

As you can see in the middle of Figure 13, there is a lot of noise since there is doubled data since the map has the old data of ground truth position of the furniture and it has the new data of where the furniture is, which is also in there. The older positions of the furniture need to be filtered out while keeping the data of the walls and the new positions of the furniture.

This processing of the files was done by first converting the PGM files from the P5 PGM format, which is in binary, to the P2 PGM format, where the data is encoded in ASCII. Once that was done to both of the input files, the sides needed to be cropped since there are large amounts of gray space that would waste computation power to the left and right as can be seen in Figure 12. This was done by scanning from the left and the right to see the first column that had more than 5 black pixels in the column and cropping everything before that. Then, once the input files had been cropped and put into 2D array representations for easier processing, the walls were first isolated from both files using percentages of gray and white that were around each pixel since walls have a particular range of percentages for black, white, and gray as well as a pretty small radial distance to an unmapped pixel. These factors were combined to determine whether each pixel was a wall and then the walls were isolated from each file. The version of the data without walls from the pre-remapping data and the ground truth was then run through a significance isolator that again, iterated through pixel by pixel to find out if the same pixels were there in both files, and if they were, to check whether similar groups of pixels existed within a certain radius. If that same group existed within that radius, then it was known that the furniture had moved and the group that was not present within the ground truth was the new position and the old group was deleted from the data. Once the wall data was added back in and the thickness of the lines was boosted, the remapping was complete and output to a file, similar to as seen below.

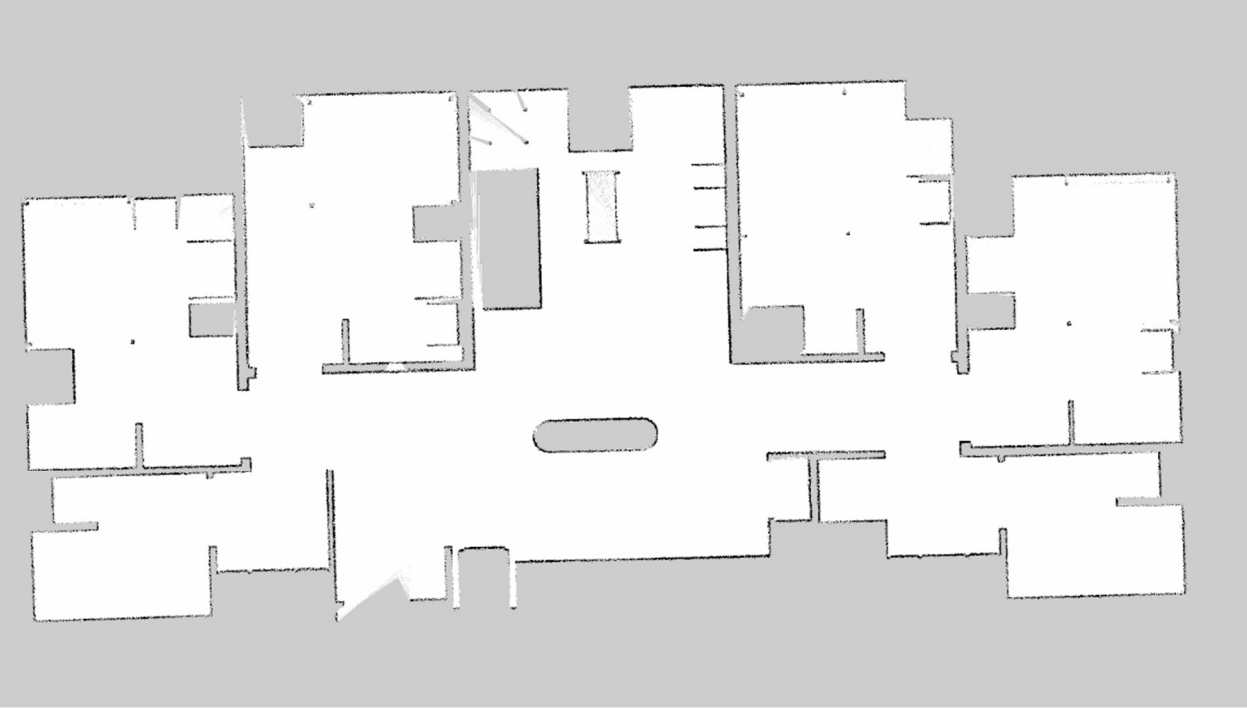

Figure 16. Post-remapping map data

The timings for how long my implementation took to reimplement the new data into the map compared to remapping the entire environment is shown below. Each number is the average of 5 experimental runs.

	Remapping Entire Environment	My Implementation
Small Changes	167.3 s	32.5 s
Larger Changes	284.8 s	56.7 s

Table 3. ECEN 403 Remapping Subsystem Data

5.3. *Problems/Limitation with ECEN 403 Remapping Approach*

This first approach developed in ECEN 403 had some key limitations that made it infeasible for use in real world use cases, especially with the lower cost LIDAR sensor (the RPLidar A3) that the ROSbot and other more affordable robotics solutions come with. The first of these limitations was that the previous approach relied on certain features of the map output to be present. Namely, it relied on near perfect mappings of the environment where everything past a wall would be unmapped and there was very little noise. However, this is not the case in real life. As can be seen in Figure 17, even in a well generated map, there is noise and erroneously mapped pixels outside of the main walls of the indoor space. This means that the first approach would already be infeasible.

The first approach also has the limitation that it only works well for small movements of the furniture and other non-wall objects, not allowing for larger, more radical movements

that occur like doors being opened and closed or furniture being moved around more than a few inches. To do this, a second remapping approach, focused on optimizing for the real world using the ROSbot as the base, was developed.

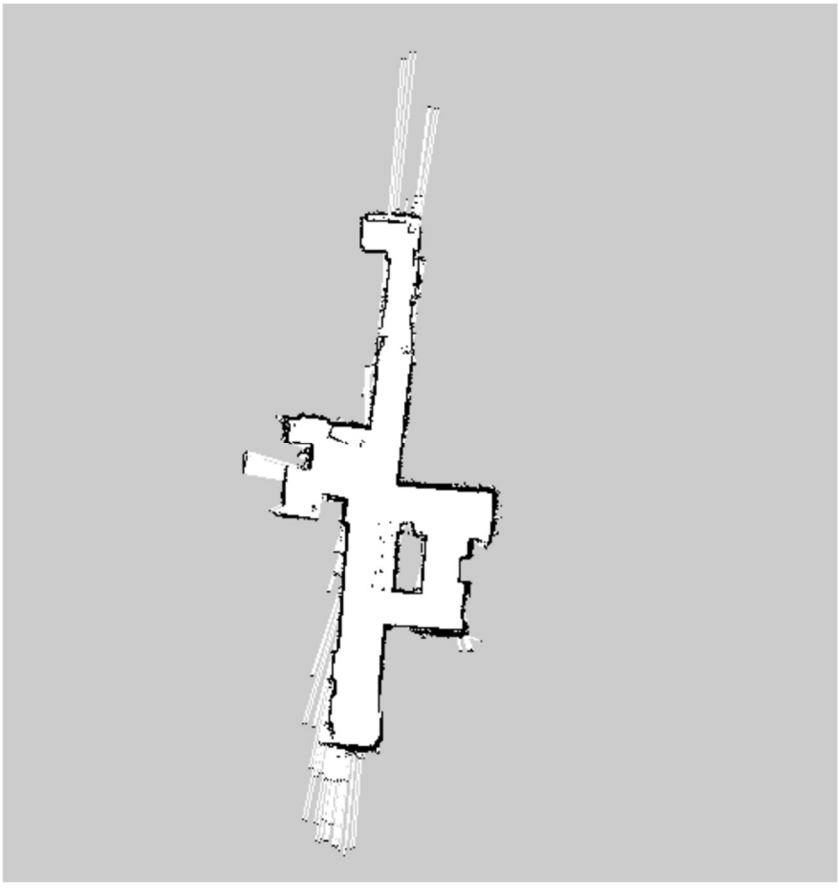

Figure 17. Initial map generated by using SLAM to map apartment space

5.4. *Improved ECEN 404 Remapping System*

The second, newer, approach fixes many of the problems of the first approach, namely that it works even with large amounts of noise on the initially generated map and that it also allows for a much greater range of changes. This is done by fusing the data from the mapping system and the localization system.

More specifically, the localization system takes in a generated environment map, as seen in Figure 17, and then localizes to where it is in the map, assuming that at least in its initial position the environment is very similar to the initial mapping. The mapping system based on SLAM that was described in section 3 is also launched. This mapping system starts to create a new map of the environment as it currently is. The remapping node then comes in and takes a window of data of a specified radius from around the robot position in both the initially generated map as well as the new SLAM map.

The remapping node then also takes in the transforms of each of these windows and carefully aligns the two windows together so that both of these windows now show the same part of the indoor space, but one with the new map and one with the data from the initial map generation.

Once these windows are aligned, the overlap between these two is calculated and once the overlap of the occupancy grid between the two windows is higher than a specifiable parameter for a specifiable period of time, the remapping node understands that the localization is complete since the localization data lines up with the new map data.
The remapping node then moves on to the map revision part of the process. In the revision process, the remapping node takes the initially generated map's window and will go through the window and for each point in the occupancy grid, change data according to the following rules:

1. If the current point in the initial map window is occupied and it can be determined that there are some occupied points in the SLAM map window around the current point, keep the point data from the initial map.
2. If there is a point that is occupied in the SLAM map window and not marked as occupied in the initial map window, revise the map to mark the current point as occupied to reflect the SLAM map window.
3. If there is a point that is occupied in the initial map window and marked as not occupied in the SLAM map window, revise the map to mark the current point as not occupied to reflect the SLAM map window.

It can be seen from the above rules that this set of rules would lead to a window of data that fuses the most up to date/relevant data points from each of the two maps, allowing for the map to only be updated where necessary by using the initial map as much as possible and still allowing for new data points to come in through the SLAM map window.

While the rules above are simple, there are many tunable parameters in the code that allow for this system to generate a proper overlap of data in a wide array of indoor environments.

5.5. *Validation*

Since this is a URS project, the validation data and conclusions about this part serve as full system validation since this remapping system integrates the results and/or work of all previous systems.

The data from the full SLAM map that was just generated can be seen in Figure 18. The SLAM map window used for remapping can be seen in Figure 19, the data from the initial map window can be seen in Figure 20, and the combined data window after they have been lined up and revised can be seen in Figure 21. Finally, the result of this fusion and revision to the initial map can be seen in Figure 22. When compared to 17, it can be seen that the new map in Figure 22 has updated data that reflects the door being open at the top of the map. All this improved performance is achieved while still maintaining similar performance numbers as the previous simulation based remapping system.

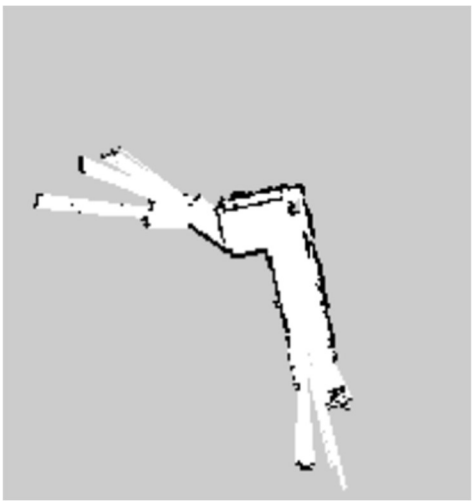

Figure 18. Full data from new SLAM instance in remapping node

Figure 19. Window selection of SLAM data for remapping node

Figure 20. Window selection of initially generated map for remapping node

Figure 21. Window of combined data, keeping data from initial map but also adding state of door being open from new mapping

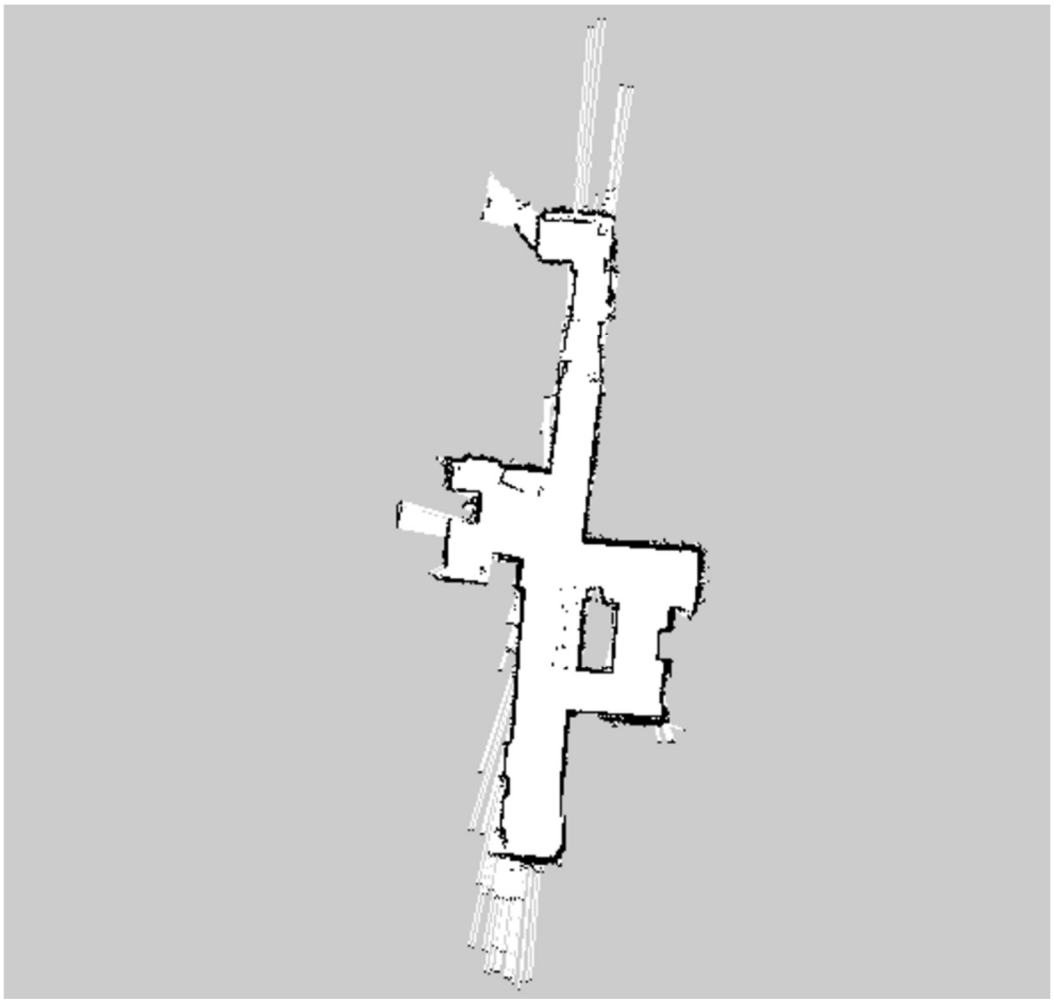

Figure 22. Revised map that shows the change of the door being opened at the top of the map as opposed to closed in Figure 17

Specification	Requirements	Results (average)	Status
Time to Localization	< 45s	38 s (25 runs)	PASS
Time to Map Revision	< 75s	58 s (25 runs)	PASS
Revised Map Error	< 5 cm	4.8 cm (10 points)	PASS
Door Revision Error	< 10 cm	5.9 cm (25 runs)	PASS
Large Object Revision Error	< 10 cm	6.4 cm (25 runs)	PASS
Small Object Revision Error	< 10 cm	9.7 cm (25 runs)	PASS

Table 4. Validation Data for ECEN 404 Remapping System

The validation results shown in Table 4 detail performance metrics for each step and functionality of the remapping system. The time to localization field details the time once the system was launched that the system would localize the robot within the environment. The time to map revision field details the time once the system was launched that the system would start revising the previous map with new data. The revised map error field specifies the error on dimensions and distances in the revised map. The door revision error specifies the difference between the amount a door was moved in real life versus the amount that the revised map showed that it had moved. The large object revision error specifies the difference between the amount a large object (6-24 inches visible to the LIDAR) was moved in real life versus the amount that the revised map showed that it had moved. The small object revision error specifies the same difference as the large object revision error, but this time for objects that have a LIDAR cross section of less than 6 inches. All the values seen in Table 4 were the result of averages from 25 runs in each environment, except for the revised map error where 10 different points across a map were sampled and the error calculated.

The system was validated in 4 different environments. The first environment was an indoor apartment as seen in Figure 17-22. The second and third environments were of my robotics lab. The fourth environment was of an apartment recreational area that was much more open than the other two environments and as a result, was not completely mapped initially.

5.6. *Conclusion*

From the validation and data from Table 4, this system works well for its targeted use case and in a variety of different indoor environments. From the data however, it can be seen that the larger an object is and the larger its LIDAR cross section, the easier it is to remap. This intuitively makes sense since those smaller objects are much more difficult to track and map and as such are even harder to remap. It was also found that in environments that are more open, and have not been mapped in their entirety, the robot was not able to traverse previously unmapped areas and add that to the map since this system was meant to revise existing maps, not to add to them. Finally, it was found that when the robot first starts, there needs to be a sufficient amount of data points that have not changed since the initial mapping. This is so that the localization part of the remapping system has something to base the rest of the remapping off of. Once the robot has localized, there can be a much larger amount of changes and the robot will internalize them and revise the map.

This system overall was able to do fulfill the design goal of revising environment maps in real time. Though computationally heavy, the remapping system that was the final result of this project that was developed in ECEN 404 can be extended to work on even greater systems and can be optimized to work in real time with an even faster refresh rate. This system of

taking localized slices of the output of different mapping and localization systems and overlaying them and integrating old and new data can be used not only with 2D LIDAR sensors as seen here, but it can also work with almost every other kind of mapping and mapping sensor including 3D LIDAR, undersea RADAR, ultrasonic sensors, etc. This system saves much computation power and time that would be used remapping entire environments and has great uses in swarm robotics, attending robotics, security robots, etc and I look forward to using the findings from this project in future works.

6. Appendix

6.1. *GitHub Repository of Project Files*

https://github.com/yashas-salankimatt/URS-Project

6.2. *Link to YouTube Demo for Mapping in Simulation*

https://www.youtube.com/watch?v=qJp8XyhPQ_o

6.3. *Link to YouTube Demo for Localization in Simulation*

https://www.youtube.com/watch?v=Tg-mSti8fpk

6.4. *Link to YouTube Demo for Remapping in Simulation*

https://www.youtube.com/watch?v=2kdwm_7KHEs

6.5. *Link to YouTube Demo for Mapping on ROSbot*

https://www.youtube.com/watch?v=sd0dRmPITRY

6.6. *Link to YouTube Demo for Localization, Remapping on ROSbot*

https://www.youtube.com/watch?v=t--rqz0HZvo