Implementation of Semantic SLAM on a Mobile Manipulator System

by

Zachary R. Francis

B.S. Electrical Engineering and Computer Science and Mechanical Engineering, MIT, 2024

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

MASTER OF ENGINEERING IN ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2025

© 2025 Zachary R. Francis. All rights reserved.

The author hereby grants to MIT a nonexclusive, worldwide, irrevocable, royalty-free license to exercise any and all rights under copyright, including to reproduce, preserve, distribute and publicly display copies of the thesis, or release the thesis under an open-access license.

Authored by: Zachary R. Francis
Department of Electrical Engineering and Computer Science
May 20, 2025

Certified by: John J. Leonard
Samuel C. Collins Professor of Mechanical Engineering, Thesis Supervisor

Accepted by: Katrina LaCurts
Chair, Master of Engineering Thesis Committee

Implementation of Semantic SLAM on a Mobile Manipulator System

by

Zachary R. Francis

Submitted to the Department of Electrical Engineering and Computer Science
on May 20, 2025 in partial fulfillment of the requirements for the degree of

MASTER OF ENGINEERING IN ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

ABSTRACT

In the field of robotics, the development of household robots capable of performing everyday tasks continues to be a major area of research and practical interest. Many domestic chores—such as picking up and moving objects from one location to another—have been successfully performed by stationary robotic manipulators paired with visual perception systems. However, accomplishing more complex, varied, and spatially distributed tasks in real-world home environments requires a mobile platform with a more human-like form factor. These tasks demand greater flexibility, spatial awareness, and interaction capabilities than fixed systems can typically provide.

This work focuses on the RBY1 robot from Rainbow Robotics, a humanoid platform designed to support advanced manipulation and mobility. A range of tools and modules were developed to enhance its functionality, including software for semantic perception, task execution, and environment interaction. This thesis provides a technical overview of these tools, highlighting their roles in collecting new datasets that can be used for semantic SLAM researech. In the future, these tools can enable the robot to operate more effectively in domestic settings, towards the ultimate goal of enabling more capable home-assistive robots.

Thesis supervisor: John J. Leonard
Title: Samuel C. Collins Professor of Mechanical Engineering

Acknowledgments

For the past 24 years, I have had the support of my family pushing me through everything that I have ever done. My father, Richard Francis, has always acted as a guiding and loving presence, pushing me forward, even if I didn't want to be. I can truly say that any accomplishment I have made in my life has been because of him. My mother, Juanitta Francis, is no different. She has seen every part of me through the years, good and bad, and has loved me regardless of how difficult it has gotten at times. She is the reason I believe in myself and the reason I can be proud of myself. My sisters, Chloe, Shannon, and Samantha, have always been examples to me of what I should strive for: Chloe, in showing me sustained academic excellence, an attention to detail, and a perspective that life is more than just the work we do; Shannon, in showing me how important it is to care for myself both physically and mentally, an open-mindedness to new perspectives and opinions, and unbelievable resilience; and Samantha, in showing me a love for life, a true expression of oneself, and being the best friend I could ever ask for. My Auntie Janet has been a third parent to me my entire life and has never failed to make me feel loved and cared for. My Auntie Michelle has always believed in me and encouraged me to be upbeat. My Auntie Sally always calls to check in on me to make sure that I am on track and doing well. My Auntie Denise, even if we haven't spoken too much, always makes me happy when I hear her voice over the phone. Grandpa Chin inspired me from a young age to connect more with my heritage and shared my interest in engineering and robotics. Grandma Chin always made me laugh and encouraged me to be a decent person. Elsa and Maria helped raise me to be the man that I am today, and I can't express how important they are to me. And to my late Grandma Francis, I miss you so much. I wish you could see me now that I have grown up and done so much. I remember all the loving words you gave me, all the nights I spent watching TV with you, all the times you called me Richard because I reminded you so much of my father. I really wish you could see it all because I know you would have loved it. I love you all so much, and I want you all to see how far I can truly go.

I would also like to thank Professor Leonard for all the patience and grace that he has given me over the course of these two semesters. He has helped me in many more ways

than just academically, and I do not believe that I would have been able to get through this without all of his support and generosity.

Jungseok Hong has helped me immensely throughout this process and is the reason that I am able to submit this. No matter how many questions I had or how far behind I had fallen, he consistently made sure that I was in a position to succeed. I wish that the beginning of my time in the lab looked more like the end, but I am still very grateful you were the constant.

There are many others in my life that I met throughout my time as an undergrad or through my travels that I haven't been able to mention. I'm sorry that I can't fully express my gratitude to all of you. Sunny, Nathan, Jesse, Rondel, Jasmine, Ivy, Uyen, Yen, Kaito, Brian, Bryan, Kimi, Lennie, and so many others have made my life better.

Thank you all.

Contents

List of Figures 9

1 Introduction 11
- 1.1 Simultaneous Localization and Mapping (SLAM) . . . 11
- 1.2 Semantic SLAM . . . 13
- 1.3 Data Association . . . 13
- 1.4 Perception Systems . . . 14
- 1.5 Pose Estimation . . . 16
- 1.6 Robotic Manipulation . . . 17
- 1.7 Inverse Kinematics . . . 18
- 1.8 The RB-Y1 . . . 19
- 1.9 Containerization . . . 20

2 Related Work 21
- 2.1 SEO-SLAM . . . 21
- 2.2 Max-mixture . . . 22

3 System Overview 23
- 3.1 Implementation on an RGB-D Camera . . . 23
- 3.2 The ROS bridge . . . 24
- 3.3 Reliable Communication . . . 24
- 3.4 Navigation and Object Avoidance . . . 26
- 3.5 Controllers . . . 28
- 3.6 Example Data Collection . . . 30

4 Summary and Future Work 31
- 4.1 Summary . . . 31
- 4.2 Future Work . . . 31

A ros1_bridge code 33

References 41

List of Figures

1.1 An example of a map generated using SLAM [3] 12
1.2 Maximum Likelihood Estimation framework applied to SLAM [8] 14
1.3 The depth image generated with the use of a Lidar system [10] 15
1.4 The output of an RGB-D camera: a traditional RGB image and a depth image [12] 16
1.5 An example of joint space control on a robotic manipulator. Desired joint positions are input and joint positions are output [16] 17
1.6 An example of cartesian space control on a robotic manipulator. A desired pose is input and joint positions are output [16] 18
1.7 The full body of the RBY1 system [18] 19

3.1 A high-level look at the communication flow. 25
3.2 The host machine and docker containers running inside it 26
3.3 An example of a cost map with the associated simulated lidar measurements 27
3.4 The actual joint of the RBY1 robot being read and imposed in simulation . . 28
3.5 A trial of RGB-D data being collected while running the RBY1 remotely . . 29
3.6 The trajectory of the RBY1 around the CSAIL Living Lab, computing using OrbSLAM3. 29

Chapter 1

Introduction

As robotic and perception systems mature further, they are able to take on not only more complex, but meaningful tasks as well. One such task is that of perceiving a grouping of objects and interacting with it with the same level of complexity and decision making as a human. Specifically, the ability to recognize the changes done to the scene, and to undo previously made motions in order to get back to the original scene in reverse sequential order. Appropriate software and hardware choices must be made in order to properly achieve this goal. The Marine Robotics Group has produced work on robust semantic SLAM systems for use in object detection and segmentation. While traditional SLAM is used to map and localize robots in an environment, semantic SLAM can be used to perceive and interpret objects in a given scene. This work will be the underlying framework that will be used and implemented on the hardware. The mobile manipulator used will be the RB-Y1 produced by Rainbow Robotics, a dual-armed system with seven degrees of freedom per arm. This work aims to analyze and document the implementation of the Semantic SLAM work done by the Marine Robotics Group onto the RB-Y1 mobile manipulator and create a semantic robotic grasping system. As a stepping stone in the process, the software will first be implemented and tested on a system consisting of an RGB-D camera and a fixed manipulator. By first understanding and debugging the implementation process on this system, the same process should be easier on the RB-Y1 mobile manipulator.

1.1 Simultaneous Localization and Mapping (SLAM)

Simultaneous Localization and Mapping (SLAM) is a fundamental computational problem in robotics, wherein a mobile robot constructs or updates a map of an unknown environment while concurrently estimating its own position within that map [1]. This dual estimation challenge arises from the fact that accurate localization requires a map, and accurate mapping

Figure 1.1: An example of a map generated using SLAM [3]

requires reliable localization.

For a mobile robot operating in three-dimensional space, the robot's pose is typically represented in the special Euclidean group $SE(3)$, which encodes both its position and orientation. Let the robot's trajectory be denoted by the pose vector $x \in SE(3)$, and let l represent the positions of observable landmarks in the environment.

The robot receives a sequence of noisy observations z, which provide indirect measurements of both its own pose and the positions of landmarks [2]. These measurements are inherently uncertain due to sensor noise and environmental ambiguity.

To handle this uncertainty, SLAM is typically formulated as a probabilistic inference problem. The objective is to compute the most likely (i.e., maximum a posteriori, or MAP) estimate of both the robot's trajectory (or collection of poses) $\hat{X}$ and the set of landmark positions $\hat{L}$, given the full set of sensor measurements Z. This leads to the MAP optimization formulation:

$$\hat{X}, \hat{L} = \arg\max_{X, L} p(X, L \mid Z) \tag{1.1}$$

This formulation seeks the joint configuration of robot poses and landmark positions that best explains the observed measurements under a probabilistic model. In practice, solving this problem often involves techniques from nonlinear optimization, Bayesian filtering (e.g., the Extended Kalman Filter), or graph-based methods such as pose graph optimization and factor graphs.

1.2 Semantic SLAM

Building on the foundation of traditional SLAM, Semantic SLAM extends the problem by not only estimating the 3D geometry of a scene but also encoding semantic information speci cally, assigning labels to objects within the environment. By integrating semantic information into the representation of landmarks, robotic systems gain a more human-interpretable understanding of their surroundings [4]. For instance, rather than relying solely on geometric coordinates, a robot can interpret and execute higher-level commands such as move to the green circle.

In Semantic SLAM, the landmark vector L includes both spatial and semantic components and can be modeled as:

$$L = SE(3) \quad C \tag{1.2}$$

where $SE(3)$ represents the geometric pose (position and orientation) of the landmark, and $C = f1; : : : ; Cg$ is a prede ned set of discrete semantic classes. Similarly, the measurement vector Z consists of both geometric and semantic observations, enabling more robust and meaningful data association.

Two key components of the Semantic SLAM pipeline are object detection, the process of actually recognizing objects a scene and classifying them, and object segmentation, the process of separating out the three dimensional representation into distinct object clusters. These steps allow the system to extract semantic labels from raw sensor data and associate them with landmarks. Deep learning-based methods such as YOLO [5] and Faster R-CNN [6] are widely used for object detection, o ering signi cant improvements in accuracy and speed over traditional computer vision techniques. For object segmentation, methods like SegNet [7] provide dense pixel-level classi cation, although classical approaches (e.g., contour detection or color thresholding) may still be e ective in structured or well-lit environments.

By combining semantic understanding with spatial awareness, Semantic SLAM enables robots to interact more intelligently with dynamic, human-centric environments.

1.3 Data Association

In the context of Simultaneous Localization and Mapping (SLAM), *data association* is the critical process of correlating newly acquired sensor measurements with previously observed landmarks or environmental features. Accurate data association ensures consistent mapping and localization by correctly identifying which observed features correspond to which elements

of the existing map. Let D denote the set of all possible data associations for measurements z at a given time, such that $D = \{d_t\}$, where each d_t represents an association at time t.

One common strategy for resolving data association is the maximum likelihood (ML) approach. In this method, the data association con guration that maximizes the likelihood of the observed measurements is selected. Formally, the objective is to identify the most probable associations and, conditioned on these, estimate the most likely robot poses and landmark positions [2].

Figure 1.2: Maximum Likelihood Estimation framework applied to SLAM [8]

While the maximum likelihood approach is conceptually straightforward and computationally e cient, it is highly sensitive to incorrect associations. A single erroneous match can propagate through the SLAM pipeline, resulting in substantial deviations between the estimated and true states. To address this vulnerability, more robust probabilistic techniques have been developed.

A prominent alternative is the probabilistic data association method, which considers multiple potential data associations simultaneously. Instead of committing to a single hypothesis, this approach marginalizes over all possible associations, weighting each by its posterior probability. This yields a more resilient estimation framework, reducing the impact of outlier associations:

$$p(X, L \mid Z) = \sum_{D} p(X, L \mid D, Z)\, p(D \mid Z) \tag{1.3}$$

Like previously stated, X represents the robot trajectory, L the map of landmarks, and Z the set of sensor measurements. The equation expresses the joint posterior over robot poses and landmark positions by marginalizing over the space of possible data association sequences D.

1.4 Perception Systems

To collect meaningful data from the environment, a robotic system must rely on a perception system capable of detecting and measuring features such as object positions or environmental

Figure 1.3: The depth image generated with the use of a Lidar system [10]

landmarks. One common approach is to use LiDAR (Light Detection and Ranging), which estimates distance by emitting laser pulses and measuring the time it takes for the re ected light to return to the sensor. This time-of- ight measurement provides accurate, real-world distance data with high precision. However, LiDAR systems are often expensive, power-intensive, and can be a ected by adverse weather conditions such as rain, fog, or snow, which degrade the quality of the returned signal [9].

As a more a ordable and compact alternative, RGB-D cameras have gained popularity in robotics and computer vision applications. These devices provide both traditional color (RGB) imagery and depth data using various depth-sensing technologies. One such technology is stereo vision, which mimics human binocular vision to infer depth by comparing two images taken from slightly di erent viewpoints.

The RGB-D camera used in this work, the ZED2i, relies on stereo vision to estimate depth. It employs two cameras mounted at a known, xed baseline distance apart. When these cameras capture an image of the same scene simultaneously, each point in the environment appears at slightly di erent positions in the left and right images a phenomenon known as disparity. By analyzing this disparity, the system can triangulate the depth of each point in the scene. The process involves nding corresponding points in both images, calculating the disparity between their positions, and applying geometric principles (based on the known camera intrinsics and baseline) to compute the depth [11].

Stereo vision is advantageous because it does not rely on active illumination (like LiDAR or structured light), making it more robust in bright outdoor environments. However, it can struggle in scenes with low texture, repetitive patterns, or poor lighting, where nding accurate correspondences between the two images becomes more di cult.

Figure 1.4: The output of an RGB-D camera: a traditional RGB image and a depth image [12]

1.5 Pose Estimation

Pose estimation refers to the process of determining the position and orientation collectively known as the 6-degree-of-freedom (6-DoF) transformation of an object relative to a given reference frame. With su cient perception data, such as visual and depth information, it is possible to estimate this transformation accurately. This estimation is fundamental for robotic systems, enabling them to perceive and interact e ectively with their environments, whether for object grasping, assembly tasks, or navigation [13].

When working with RGB images alone, machine learning techniques including convolutional neural networks (CNNs), transformer-based architectures, or keypoint detection methods can be used to estimate the relative poses of objects in the scene. However, RGB images lack explicit depth information, which can limit accuracy, particularly in cluttered or occluded environments.

To address this limitation, RGB-D cameras, as discussed in the previous section, can be employed. With known camera intrinsics, the depth map can be projected into a 3D point cloud, yielding spatial coordinates for each pixel. This 3D representation allows for more robust analysis, including segmentation of individual objects within the scene. Once segmentation is complete, each object can be isolated into its own point cloud, which can then be used for more precise pose estimation through techniques such as Iterative Closest Point (ICP) if a model of an image is already known, deep learning-based pose regressors, or template matching.

1.6 Robotic Manipulation

Robotic manipulation refers to a robot's ability to interact with objects in its environment and manipulate them to accomplish speci c tasks. In the context of this thesis, the environment consists of a tabletop workspace populated with objects, and the primary task is a pick-and-place operation. A robotic arm, or manipulator, with six degrees of freedom (DOF) is employed to grasp an object and relocate it within the coordinate space it occupies [13].

To perform such operations, the motion of the robotic arm must be carefully planned. This motion planning is typically carried out in the robot's joint space [14]. Joint space is an N-dimensional space, where N corresponds to the number of degrees of freedom of the manipulator. Each point in this space de nes a unique con guration of the robot by specifying the value of each joint angle or actuator position [15].

The joint space con guration is represented by a vector of joint variables:

$$q \; 2 \; R$$

where q contains the values of all joint positions. Given a complete joint vector q, the full pose of the robot manipulator's end-e ector and the position of each link can be determined through the process known as *forward kinematics*. As such, trajectory planning in joint space enables the robot to compute smooth and feasible motion paths from an initial con guration to a goal con guration, ensuring accurate manipulation of the target object.

Figure 1.5: An example of joint space control on a robotic manipulator. Desired joint positions are input and joint positions are output [16]

While the actual motion planning could be calculated in the joint space in order to better account for self collisions and joint limits, it is not the preferred coordinate system to use as it cannot take into account the position of objects in the real world. To better account for this limitation, a manipulator's end e ectors can be controlled in Cartesian space. By

Figure 1.6: An example of cartesian space control on a robotic manipulator. A desired pose is input and joint positions are output [16]

controlling the end e ectors transformation in this coordinate space, direct interaction with processed objects becomes possible.

1.7 Inverse Kinematics

While controlling a robotic manipulator in Cartesian space is often preferred especially for tasks involving interaction with objects in the real world it is typically more straightforward to control the robot via its joint positions. This is because joint control involves specifying the joint positions of the actuators directly. Controlling a joint position q can be as simple as commanding the actuator to rotate to a speci c angle or extend to a speci c length, provided that accurate feedback is available from joint encoders.

To enable Cartesian control, a process known as *inverse kinematics* (IK) is required. This process involves computing the set of joint angles that correspond to a desired pose (position and orientation) of the end e ector in Cartesian space. Once computed, these joint positions can then be sent as commands to the actuators of the robotic arm![17].

Although inverse kinematics is a powerful tool, it becomes signi cantly more complex when dealing with *redundant* or *over-actuated* systems those with more actuators (or degrees of actuation) than the degrees of freedom (DoF) required to describe the end-e ector pose. In such cases, multiple joint con gurations may satisfy the same desired end-e ector pose, resulting in a set of solutions rather than a unique one.

This redundancy introduces challenges. Without additional constraints or optimization, the selected joint con guration may ignore important considerations such as joint limits, previous poses, singularities, or the potential for self-collision. As a result, the robot may follow an ine cient trajectory or, in worst-case scenarios, cause damage to itself or its environment.

To mitigate these issues, constraints and optimization criteria are often incorporated into the inverse kinematics solution. These may include minimizing joint displacement, avoiding joint limit violations, or ensuring collision-free motion. By introducing such constraints, inverse kinematics can produce more practical and safe joint con gurations for real-world robotic applications.

1.8 The RB-Y1

The RB-Y1 is a dual-arm mobile manipulator developed by Rainbow Robotics as seen in Figure 1.7. Each arm features 7 degrees of freedom (DoF), including a linearly actuated gripper, while the robot's central torso or singular leg provides an additional 6 DoF of movement. The mobile base further contributes to high-speed navigation and mobility, resulting in a total of 24 DoF for the full robotic system.

Figure 1.7: The full body of the RBY1 system [18]

The robot is equipped with two onboard computers. The rst, known as the Robot PC (RPC), is dedicated to low-level control, handling all commands related to robot state information, joint actuation and real-time motor control. The second, referred to as the User PC (UPC), manages the gripper and supports any additional onboard computation or high-level tasks required by the user. The UPC is powered by an NVIDIA Jetson AGX Orin

module running Ubuntu 22.04. Due to compatibility constraints with this operating system and Jetson hardware, the Robot Operating System 2 (ROS 2) is required for all ROS-based development.

To facilitate control, Rainbow Robotics provides a software development kit (SDK) that allows external systems to send commands to the RPC. Communication between the UPC and RPC is handled via an internal Ethernet connection, ensuring reliable, low-latency command transmission. While there is an alternative method for sending commands directly to the RPC from a host computer via the RPC's onboard Wi-Fi hotspot this approach is unreliable and generally unsuitable for continuous operation. Further discussion on the limitations of this method will be presented later [19].

1.9 Containerization

A major goal of this thesis is to establish a streamlined and reproducible process for deploying Semantic SLAM, pose estimation, path-planning, and manipulation control on the RB-Y1 platform. In practice, con guring the RB-Y1 with a new host computer and integrating it with the existing SLAM frameworks presents signi cant challenges. These di culties primarily stem from compatibility issues across di erent software environments, hardware platforms, and operating system con gurations.

To address these challenges, this work employs containerization a method of packaging and running software along with all its dependencies in a self-contained, portable unit. This approach mitigates environment-speci c inconsistencies and ensures reliable deployment across di erent systems. Speci cally, Docker is used to construct a modular pipeline composed of multiple containers, each responsible for hosting and executing critical components of the robotic software stack. By encapsulating discrete parts in separate containers, the system achieves a high degree of exibility, portability, and ease of deployment.

Chapter 2

Related Work

Previous works have demonstrated the use cases and theory behind semantic SLAM, highlighting both its potential and the de ciencies that can be improved.

2.1 SEO-SLAM

While a very useful application of the of SLAM, Semantic SLAM systems often struggle with the mapping of clustered objects with similar semantic values. Typically speaking, the classi cations proved are limited to what is input by the creator, and they tend to not be exhaustive. This can lead to the erroneous grouping of separate objects into one simpli ed classi cation, such as a boot and a ip- op being classi ed together as just shoes While not a massive problem on its own, for a system whose goal is to eventually help in the domestic realm, this can have major consequences. For example, if a speci c medication is needed and the system simply recognizes all given medication as medicine, that would be a problem. Another issue arises from object detectors being limited and biased by their training set, which can lead to classi cation errors for objects that are less trained or novel.

To address these issues, Semantic Enhancement for Object SLAM (SEO-SLAM) is introduced as an improved version of semantic SLAM. By leveraging Vision Language Models [20] and Multimodal Large Language Models [21], SEO-SLAM can better generate descriptive tags for objects. This directly remedies the issue of limited classi cations. In the example given early, even if the system has never encountered a boot or a ip op before, they would each get a separate and accurate classi cation. To facilitate this process, object tagging, grounding, and segmentation models are integrated into the software pipeline.

2.2 Max-mixture

Previously, the probabilistic data association method was introduced to match novel sensor data with stored landmark data. It was discussed that instead of using a maximum likelihood approach, a weighted sum of probabilities would lead to more robust estimate. However, this approach does not preserve the Gaussian nature of the problem because the resulting belief is a weighted sum of Gaussian distributions. This makes it di cult to apply nonlinear least-squares optimization to solve the SLAM problem. To address this, the max-mixtures approach is used instead of the sum-mixtures" [2].

$$p(X; L \mid Z) = \max_{D} p(X; L \mid D; Z)\, p(D \mid Z) \tag{2.1}$$

Rather than using the weighted sum of priors, the max operator selects the most probable data associations for a given set of measurements. This improves computational e ciency and allows for better loop closures.

Recent work in the Marine Robotics Group at MIT has built on this approach to explore a variety of techniques for Semantic SLAM, including recent PhD theses by Doherty [22] and Huang [23]. This thesis helps to lay the groundwork for using semantic SLAM for mobile manipulation, using the RBY1 wheeled mobile manipulator. In the next chapter, we provide an overview of the software control system to enable semantic SLAM data collection for this robot.

Chapter 3

System Overview

The work done on this thesis has focused primarily on the setup, con guration, and integration of various systems onto the RBY1 robot for future use in the Marine Robotics Group's research. The RBY1 is a very new system, and as such, important infrastructure for development was either still underdevelopment from Rainbow Robotics or did not exist.

3.1 Implementation on an RGB-D Camera

The rst major objective of this thesis was to successfully integrate the Max-Mixtures SLAM pipeline with a standard RGB-D camera. Initial implementation was performed using an Intel RealSense D455, which provides synchronized color, depth, and inertial measurement unit (IMU) data all critical for the Max-Mixtures SLAM framework. Leveraging Intel's SDK and associated ROS packages, the integration process on standard Ubuntu systems was relatively straightforward. Actually viewing the images from the camera on Rviz, ROS's 3D visualization tool was easy. And running these images through the Max-Mixtures pipeline also proved easy,

However, signi cant challenges emerged during deployment on the RB-Y1 platform. Since wireless control and real-time communication with the robot were required throughout operation, all sensor data had to be transmitted from the RGB-D camera through the onboard UPC and relayed to a host machine via a network router. The UPC, which runs Ubuntu 22.04, and NVIDIA's JetPack 6 SDK, presented a critical compatibility issue: the RealSense D455 is not natively supported in this software environment.

While one possible solution involved running a Docker container on the UPC with a compatible kernel and middleware to interface with the RealSense camera, this approach was ultimately rejected. This solution would have worked, but, because of internal interactions between the UPC and RPC, most processes on the UPC are preferred to be done on its

native environment.

To circumvent this issue, the ZED2i depth camera was selected as an alternative. The ZED SDK provides equivalent functionality delivering synchronized RGB, depth, and IMU data and is fully compatible with the JetPack 6 environment on the UPC. This allows for only one environment to be needed on the UPC.

Once this compatibility issue was resolved, the ROS2 messages containing the color, depth, and IMU data could be created on the UPC.

3.2 The ROS bridge

Establishing reliable communication between the UPC and the host computer, while conceptually straightforward, presented a number of technical challenges. As discussed previously, the UPC uses Ubuntu 22.04 as its operating system, which o cially supports only ROS 2 distributions. In contrast, the Max-Mixtures SLAM pipeline used in this project was developed using ROS 1, leading to an incompatibility between the two systems.

To bridge this gap, the `ros1_bridge` package was used. This ROS package enables message and service exchange between ROS 1 and ROS 2 nodes by translating between the versions. For the `ros1_bridge` to function properly, both a ROS 1 and ROS 2 distribution must be installed on the same system. Ubuntu 20.04 is uniquely suitable in this task, as it is the only version that o cially supports both ROS 1 Noetic and ROS 2 Humble the respective versions used in this project. A Docker image based on Ubuntu 20.04 was therefore constructed, containing both ROS environments. This image was deployed on the host machine to facilitate seamless bridging between the two ROS ecosystems.

For the Max Mixtures pipeline to work properly, it requires rgb data, depth data, odometry data, and camera info. The speci c topics bridged can be seen in the `bridge.yaml` code in the appendix.

3.3 Reliable Communication

The UPC itself lacks wireless networking capabilities. To establish connectivity, it was physically tethered via Ethernet to a Netgear R6700 router, with the host machine connected to the same local network. To ensure that ROS 2 messages transmitted from the UPC were properly received by the host machine and subsequently by the Docker container the `ROS_DOMAIN_ID` environment variable was set identically across the UPC, the host machine, and the Docker container. This uniform setting is essential for ROS 2 nodes to communicate within the same domain.

Figure 3.1: A high-level look at the communication ow.

Additionally, to enable proper message discovery and transmission over the network, the ROS middlewarefast_dds was used.fast_dds serves as the middleware implementation for ROS 2, facilitating e cient and reliable message passing across networked nodes. To enable this, the RMW_IMPLEMENTATION environment variable was set tormw_fastdds_cpp in both the UPC and the Docker container.

Once these con gurations were in place, the communication pipeline operated as follows: ROS 2 messages generated on the UPC by the Zed2i camera were transmitted over the local network to the Docker container running on the host PC. Inside the container, the ros1_bridge translated these incoming ROS 2 messages into ROS 1 messages, which were then input to the Max-mixtures SLAM pipeline running on the host. A high level overview of this communication ow can be seen in Figure3.1.

A notable consideration in this setup was the high volume of messages produced by the ZED2i camera's ROS 2 package. By default, theros1_bridge attempts to translate all available message topics, which can result in signi cant transmission delays and render image data unusable due to excessive latency. To address this issue, a parameter le was con gured to restrict the bridge to only translate essential ROS 2 messages required for the application. This selective bridging strategy signi cantly reduced computational overhead and network load, thereby preventing frame rate drops and ensuring that image data remained viable for downstream processing. The docker containers and ow of communication occuring inside the host machine can be seen in Figure3.2

Figure 3.2: The host machine and docker containers running inside it

3.4 Navigation and Object Avoidance

One of the primary long-term objectives of this project is to enable the robot to autonomously navigate to designated locations, analyze scenes, and perform any necessary manipulation tasks. Achieving this requires a robust navigation stack capable of reliable path planning and obstacle avoidance, both in simulation and on the actual RBY1 platform.

For this purpose, the Nav2 stack was selected due to its extensive feature set and straightforward setup process. To maximize modularity and minimize system dependencies, all Nav2 code and related simulations are executed within a Docker container on the host machine. This approach not only ensures portability and ease of maintenance but also conserves the limited storage resources available on the onboard UPC. The Docker container runs the ROS 2 Jazzy distribution, chosen after encountering compatibility issues with certain package builds.

Setting up the RBY1 for simulation in Gazebo is a relatively streamlined process. Rainbow Robotics provides a URDF model of the RBY1, which served as the foundation for the simulation. This URDF was converted to SDF format for Gazebo compatibility and was further modi ed to incorporate additional sensors, including lidars that already exist on the robot and a depth camera. With these enhancements, the robot model can be placed into a

Gazebo environment.

Figure 3.3: An example of a cost map with the associated simulated lidar measurements

In the simulated environment, a cost map can be generated using the modeled sensors as seen in Figure 3.3. This cost map uses the data range and RGB-D data coming from the lidar and depth camera, respectively, to create and project an occupancy grid onto the floor surrounding the robot. Then, using both sets of data, an obstacle layer is created that can then be used as the basis for object avoidance. The footprint of the simulated robot is a circle, the simulated robot will avoid obstacle grids, allowing for safe movement. For path planning and control of the simulated robot, Nav2 also provides planning and control servers.

A mapped environment is needed for the path planning, so the real RBY1 can be manually controlled about a specified environment. Using the onboard lidar and depth camera, a cost map can be generated and returned to the simulation. The NavFn Planner planner plugin is used in order to generate the shortest path necessary between the current and specified positions. Similarly, the DWB controller is then used to generate the corresponding velocities.

The actual odometry data and robot joint states in simulation are directly read from the real robot as seen in Fig 3.4. Using a custom ROS publisher node and the RBY1 SDK, all necessary data is read in from the RPC, transmitted through the UPC and local network onto the host network and subsequent Docker container.

Figure 3.4: The actual joint of the RBY1 robot being read and imposed in simulation

3.5 Controllers

Rainbow Robotics provides a comprehensive GitHub repository featuring valuable code for controlling most of the robot's joints using the MoveIt Motion Planning Framework. However, they do not supply a controller for the mobile base, which meant that a custom controller for both manual operation and autonomous path execution had to be created.

For remote control, a simple ROS package was installed to convert keyboard inputs into ROS joy topic messages, enabling direct teleoperation from the host machine. A custom ROS subscriber node was then developed to translate these messages into speci c speed and direction commands for the mobile base. This setup allows for intuitive and responsive manual driving.

Rainbow Robotics also o ers a well-documented SDK, which provides direct access to the robot's RPC interface and supports sending a wide range of commands. This SDK was instrumental in facilitating low-level communication with the robot and integrating custom control logic.

While the same approach used for the mobile base can be extended to control the robot's arms-whether through Cartesian, joint, or optimal control-MoveIt was used directly for all non-mobile manipulation tasks due to its robust planning and execution capabilities.

MoveIt is extremely helpful as it provides self-collision detection and smooth trajectory planning. Modeled objects can also have their direct pose, in reference to the camera frame, be input into the system. By default, MoveIt uses the base of the robot as the reference frame for all cartesian control, so by simply computing the transform between the depth camera and the base, the end e ector can be directly commanded to a non colliding object grasping pose.