UNIVERSITY OF TURKU

Multi Robot Coordination for Efficient Search and Rescue through Deep Image Processing and Communication

University of Turku
Department of Computing
Master of Science (Tech) thesis

Author:
M Mahmudul Hassan

Supervisor:
Professor Dr. Jukka Heikkonen

July 2025
Turku

Master of Science (Tech) thesis

Subject: Robotics and AI
Author: M Mahmudul Hassan
Title: Multi Robot Coordination for Efficient Search and Rescue through Deep Image Processing and Communication
Supervisor: Professor Dr. Jukka Heikkonen
Number of pages: 60 pages
Date: July 2025

This thesis leveraging the capabilities of Robot Operating System (ROS 2 Humble) and Gazebo Fortress presents the design, implementation, and evaluation of decentralized multi-robot coordination in autonomous search and rescue operations utilizing the deep image processing, parallel communication, and Nav2 navigation in a semi structured environment. The core components include four TurtleBot3 Waffle robots each equipped with 2D LiDAR and RGB-D cameras deployed in a maze-like environment to mimic real-world challenges encountered in search and rescue operations. The aim of this research is to enable autonomous exploration of unknown environment, precise detection and localization of the targets which is red cylindrical structure, and efficient task allocation across these multi robots. Two localization techniques, Simultaneous Localization and Mapping (SLAM) and Adaptive Monte Carlo Localization are implemented and evaluated rigorously. While SLAM localization generates dynamic mapping in a real-time manner of an unknown environment, AMCL loads static pre-defines map from the Nav2 map server. Each localization method offers distinct advantages such as AMCL delivers faster initialization, reduces CPU usage and deterministic navigation in familiar space while SLAM provides dynamic adaptability at higher computational load with uncertain localization initially. Feature detection is obtained using a pre-trained YOLOv8 model integrated via the ultralytics library. Realtime RGB images from simulated Intel RealSense R200 cameras undergo deep learning based processing for target identification while depth images, intrinsic calibration, and odometry data are fused to correctly compute gazebo world coordinates which is obtained leveraging the quaternion-to-Euler conversions and yaw-based rotation matrices for reliable camera-frame detections to the global environment in the custom DepthToWorldConverter ROS 2 node. Target detections are shared across all robots via the /global_cylinder_detections topic. The system demonstrated accurate object detection approximate to 95%. A decentralized coordination manager evaluates each robot's shortest path only in the AMCL localization technique using the ComputePathToPose and selects the closest robot resulting in issuing NavigateToPose action. Detection and allocation states are visualized using a GUI Tkinter and monitored through rqt_graph, rviz2, and ROS 2 logs. Utilizing the ROS 2 Data Distribution Service (DDS) standard Wi-Fi shows low message latency (0.05-0.24s) while Bluetooth like environments shows significant delays (up to 1.1s) which at times impacts the coordination. Performance benchmark evaluated under both SLAM and AMCL configuration and focuses on critical metrics such as CPU load, target detection, obstacle avoidance, communication latency, localization accuracy, map coverage, task completion time, and system robustness under varying network conditions. To sum up, this thesis provides a rigorous and technically robust, scalable solution for autonomous multi-robot systems in SAR contexts. Integrating advance deep learning perception, real-time coordinate transformation, decentralized decision making, situational awareness, and resilient navigation mimicking real world deployment in SAR and other high-risk environments. The research not only demonstrates the feasibility of decentralized autonomous agents for complex operations leveraging state of the art image processing method, communication framework, localization techniques but also establishes a foundation which paves the way for a more safe and efficient SAR operations.

Key words: MRS, SAR, Situational Awareness, AMCL, SLAM, Object detection, Decentralized task allocation, Parallel Communication, MARL.

Acronyms and Abbreviations

MRS	Multi Robot Systems
SAR	Search And Rescue
ROS	Robot Operating System
DDS	Data Distribution Service
AMCL	Adaptive Monte Carlo Localization
SLAM	Simultaneous Localization and Mapping
UAV	Unmanned Aerial Vehicle
USV	Unmanned Surface Vehicle
MARL	Multi Agent Reinforcement Learning

Table of contents

1 Introduction

The advancement of autonomous systems coupled with modern artificial intelligence made the collaborative multi agent operations to a different pinnacle. Specified goal with decentralized system among multiple robots is one of the most impactful developments in the domain of multi-robot systems which is putting massive positive impacts for a number of complex tasks. Multi-robot systems is now playing a vital role from precision agriculture, logistics, industrial automation to search and rescue in any disaster situation, military based operations.

1.1 Background and motivation

Coordinated distributed agents in dynamic environments add a robust impact in the multi-robot systems. Leveraging shared information from different robots, localized decision making, and parallel communication among the robots, enhances multi-robot systems efficiency, scalability, fault tolerance, and adaptability. These attributes play a vital role in any search and rescue operations where environments are unknown maybe even hazardous demanding a rapid and autonomous action.

Development in the integration of real-time localization, object detection, autonomous navigation, and distributed tasks segmentation frameworks has made the MRS closer to practical deployment in different applications. Yet, there are still considerable challenges in the section of seamless collaboration among the robots, particularly when conducting task or any operation in limited sensing, uncertain localization, and sparse inter communication. This study addresses these challenges by developing and evaluating a multi-robot search and rescue system within a simulated environment using ROS 2 and Gazebo. Utilizing SLAM and AMCL for localization, a multi-robot systems performs an autonomous search and rescue for the designated target leveraging the deep image processing, and communication for optimal coordination among robotic agents.

1.2 Problem statement and research objective

Several technical challenges need to be surpassed while deploying multiple autonomous robots for coordinated search and rescue tasks to get an optimal result. These challenges include maintaining localization among the robots, robust communication where network maybe not up to par, detecting the rescue target maybe more complicated due to the complex

visual environments, and efficiently allocating tasks among the distrusted agents. This thesis investigates the central research question of how a decentralized multi-robot systems can accurately detect, localize, and coordinate the rescue of a target in an unknown environment leveraging the AMCL and SLAM localization techniques.

To address the research question a comprehensive analysis with real-time simulation has been conducted. Concentration has been put into designing a scalable MRS using ROS 2 which conducts an autonomous exploration, target detection and navigation. This framework integrates deep leaning based visual perception through YOLO to enable real-time detection of specified target objects. To compare localization efficiency, Simultaneous Localization and Mapping (SLAM) and Adaptive Monte Carlo Localization (AMCL) have been assessed under consistent experimental conditions. Additionally, a decentralized task coordination mechanism is implemented to autonomously determine which robot is closer to the target leveraging the A* (A-star) algorithm based on path efficiency and proximity. Finally, the system's performance is measured based on comprehensive evaluation of detection latency, communication reliability, map coverage, localization accuracy, and task completion time across the localization strategies.

1.3 Research scope

This study is focused on ground based autonomous robots in a controlled simulated maze environment for a search and rescue scenario. The simulation is performed using the TurtleBot3 Waffle platform configured in Gazebo Fortress integrated with ROS 2 Humble Hawksbill middleware. Four autonomous robots under unique namespaces each equipped with LiDAR sensor and RGB-D camera (Intel RealSense R200) were launched into the maze environment. Robots operate using the Nav2 stack for navigation while a deep learning-based detection model YOLO identifies red cylindrical targets. A ROS 2 node converts image space into world coordinates of the Gazebo world using depth data and robot odometry resulting in navigation for the robots toward the detected targets. Two different localization techniques are compared, SLAM and AMCL. While AMCL uses a preloaded map for immediate localization and navigation, meanwhile SLAM builds a map of the environment dynamically during exploration. Both localization techniques are evaluated for robustness, efficiency, and suitability for real-time, multi-robot coordination in semi-structured environment. This research outcome is applicable for a wide range of domains where human intervention is

limited or unsafe but reliable and effective deploying the autonomous robots considering the real world, mission critical environments.

1.4 Thesis organization

The structure of the thesis is conducted to begin with a logical progression from foundational research to implementation and evaluation considering a real-world search and rescue scenario. Chapter 2 contains literature review which explores existing research working in the domain of multi-robot coordination, search and rescue application, object detection techniques, inter robot communication strategies, and situational awareness frameworks. Chapter 3 includes methodology outlining the technical concepts and foundation of the study, detailing the hardware and software configurations, system architecture, ROS 2 packages, and simulation methods implemented in building the multi-robot framework. Chapter 4 titling experimental setup and implementation describes the configuration of the simulated environment, the process of deploying multiple robots with unique namespaces, and the implementation of exploration, detection and coordinate transformation components to reach the set target. Chapter 5 includes the results and discussion presented both quantitative and qualitative outcomes from the experiments, comparing the effectiveness of SLAM and AMCL localization methods with respect to system performance, detection accuracy, and task coordination. And in the final chapter, chapter 6 contains conclusion and future work which summarizes the key findings, the limitation with the system, and proposes directions for further research including physical deployment and incorporation of aerial and underwater robotics unit for more complex search and rescue operations.

2 Literature Review

A thorough literature review has been performed in the section of multi-robot systems, search and rescue in multi-robot systems, communication in multi-robot systems, object detection techniques, situational awareness in robotics, multiple robot task allocation, applications of multi-robot systems, critical challenges of multi-robot systems to understand the existing knowledge, methodologies, and technical advantages in the field of multi-robot systems. The study of these relevant literature helps to identify the proven methods, current limitations, and research gaps especially in the domain of decentralized coordination, sensor integration, and multi-agent decision making in dynamic environments. Reviewing both theoretical approaches and practical implementation, the literature review ensured that the research builds upon validated concepts while addressing unresolved challenges in search and rescue operations.

2.1 Overview of multi-robot systems

Different factors come together to allow Multiple Robot Systems (MRS) to redefine robotics and become an increasing necessity. The system coordinates and decentralizes interactional occurrences for multiple agents to accomplish a complex task which exceeds the boundaries of individual robots. This concept integrates cooperation, communication, and distributed intelligence and paves the way for the scalability, fault tolerance, and efficiency of the design. Advances in swarm intelligence, network communication in real-time, edge computing, and machine learning have facilitated the rapid inclusion of MRS in a wide spectrum of applications [9]. Coupled with the advancement of multi-agent reinforcement learning (MARL) [4], decentralized decision-making algorithms [1], [4], and 5G/6G communication technologies [3], MRS has gained an edge. The focus has also shifted to challenges like coordination under uncertainty, robustness to communication failure, and ethically problematic behaviors of autonomous groups.

Major strides have been made in the control and coordination of multiple robots, particularly in decentralized cooperative frameworks [3]. Decentralized control algorithms based upon distributed optimization is offered where robots use localized information to achieve global objectives, enhancing scalability and robustness in dynamic environments [1]. In task allocation, Barton Research Group [2] developed a learning framework that estimates agent capabilities and task requirements leading to efficient multi-robot task allocations. By

embedding learned capabilities as constraints in optimization models, the framework becomes more adaptable to varying mission requirements. Effective communication is a pillar with which the functionality of MRS stands. A critical review is performed of communication strategies in MRS [3], emphasizing that robot algorithms and communication systems should be co-designed to resolve issues such as latency and bandwidth constraints. In addition, the integration of machine learning, especially reinforcement learning [4], has boosted the MRS capabilities. Wang et al. [4] surveyed approaches toward distributed reinforcement learning, stressing how they can allow robots to learn cooperative behaviors for complex tasks. Furthermore, LLMs have expanded the horizons of human-robot interaction and decision-making procedures in MRS. Yet the obstacles of scalability, robustness, and security remain persistent in MRS [9], and, thus, Yang and Tron, (2024) [8] proposed the introduction of a planning algorithm that works via co-observation and reachability analysis to whitify security against adversarial threats in MRS.

Wide number of applications MRS has been explored such as space exploration, logistics and warehousing, disaster response [5], [6]. MRS is used in rapid search and rescue. Algorithms are being developed for multi-robot systems to operate in such environments by the Purdue SMART Lab [7] in the areas of environmental monitoring and emergency responses. While there are different types of multi-robot systems such as Unmanned Aerial Vehicles (UAV), Unmanned Surface Vehicles (USV), Unmanned Ground Vehicles (UGV), for the simulation of search and rescue operation here only ground autonomous vehicles have been utilized leveraging the turtlebot3 package.

2.2 Search and rescue in multi-robot systems

Instead of single robot or human lead search and rescue, the multi-robot systems has become more feasible and at times crucial for any search and rescue operation considering the complexity of the operation. Significant advancement has also been noticed utilizing multiple autonomous agents both aerial and ground based in situational awareness, operational search and reach and overall efficiency of SAR missions [12], [13].

Focusing on mobility, sensor fusion, and environmental mapping a robust mobile robot platform has been introduced which performs coordinated operations utilizing multiple autonomous robots in unpredictable environments [10]. Upon this, an innovative robotic system has been developed equipped with advanced navigation and perception capabilities, focusing the coordinated multi-robot deployments for improving coverage and reducing the

time to locate victims in expansive disaster zones [11]. Beyond hardware innovations, an in-depth analysis of multi-agent systems within the SAR domain, discussing how distributed sensing, data sharing, and adaptive task allocation can overcome the limitations of single-robot approaches [12]. Effective coordination depends on robust inter-robot communication protocols and real-time data exchange, which collectively enable dynamic collaboration and task reassignment as mission parameters evolve [12].

Unmanned Aerial Vehicles (UAVs) have also become crucial to modern SAR operations, especially while working in wide-area surveillance and rapid deployment [13]. Centralized and distributed task assignment frameworks such as genetic algorithms and market-based schemes for multi-UAV coordination have been studied along with exploring the state-of-the-art path planning strategies, including graph-based and biologically inspired algorithms like Particle Swarm Optimization, underscoring that efficient multi-agent coordination and autonomous path planning are essential for maximizing SAR mission effectiveness [13]. A crucial component in a multi-robot system is advanced sensory integration and user-centric control mechanisms. Though while working with the experiment, only ground based autonomous vehicles have been experimented but in the large setup of SAR operation human led robots also play a significant part. The prototype shows such an approach by combining multiple cameras, diverse environmental sensors, and intuitive operator interfaces [14]. Their work illustrates how remote monitoring utilizing vision-based human detection and robust communication modules enhances operational safety and scalability [14]. These studies represent how multi-robot systems play a significant role in the SAR system and improving the factors such as communication, deep image processing, hardware design which results in adaptive task allocation strategies, robust real-time communication, autonomous path planning can significantly improve the potential of multi-robot systems in any SAR operation.

2.3 Communication in multi-robot systems

Communication underpins the driving forces behind coordination, collaboration, and collective decision-making in Multiple Robot Systems [15], [16]. As multi-robot tasks become more complex and larger within applications such as space exploration, warehouse automation, agriculture, and disaster response [5], [6], inter-agent communications will need to be reliable and effective [16]. A primary focus in current MRS communication research is the ability to operate intermittently or with constrained connectivity. Research into intermittent connectivity is especially pertinent for contexts like subterranean exploration,

planetary robotics, or search-and-rescue scenarios, where line-of-sight communication may be intermittent; for this purpose, consider the ACHORD architecture [15]. ACHORD combines multiple levels of planning and drop-able radio relays to facilitate intermittent data transmission between robots so that they can continue to synchronize maps and gather sensor data without reliable connectivity [15]. The recently released MOCHA (Multi-robot Opportunistic Communication for Heterogeneous Agents) [16] framework also exploits decentralized communication protocols based on opportunistic gossip. The MOCHA framework [16] allows robots to share and propagate high-priority information every time they contact another robot. Both frameworks have been shown to scale on aerial and ground robot platforms and MOCHA can be especially useful in heterogeneous teams when the communicative modalities and physical capabilities are diverse [15], [16]. Simultaneously, there has been an increase in the focus on communication-efficient protocols that account not only for communication content but also for communication time and the communication recipients. Radke et al. presented a formal model for determining the times and recipients with whom a robot should communicate in order to avoid collisions while minimizing communication [17]. In this model, communication is modelled as a discrete action that will depend on the spatial context and expected motion, rather than re-thinking of communication as a continuous stream of updates. This notion is particularly valuable for reducing congestion during communication, especially with large teams or bandwidth-limited communication systems. Communication-efficient methods based in concurrent transmission control methods have also started to appear, such as in the leader-follower scenario.

A central problem in MRS communications is embedding the communication systems within the planning and control framework. Historically, very early robotic systems often developed communication, planning, and control systems as loose modules; however, research in this area has convincingly argued for co-design and co-optimization across communication, planning, and control layers [18]. Gielis et al.'s [3] critical review provides many examples of inefficiency and brittleness in robotic systems and identifies that decoupled planning and control from communication and calls for design paradigm where communication is not only designed in relation to motion planning and task allocation, but also the agents' beliefs. A more recent and growing area of attention consider "epistemic planning" [19]. Epistemic planning [19] is an extension of standard task-oriented planning, where a robot can reason about what it knows or believes as well as what others know or believe. Collaborative robotic systems can collaborate more broadly, when they can reason about their own beliefs and that

of others, like making coordination decisions even when communication is temporarily disrupted (e.g. MRS traffic negotiation or collaborative lunar mapping). By embedding epistemic models into decentralized planners, agents can reason about other agents' intentions based on their observation of team behaviors which reduces the need for real-time communication during collaborative planning [19].

The rise of large-scale, deep learning-enabled agents has also brought changes to the way that robots communicate by learning. Distributed reinforcement learning has typically been applied to train team(s) of robots to learn communication strategies without explicit human-directives. Wang et al. provided an overview of prior studies in this area, and remarked on how, when communication bandwidth is limited, robots may need to learn to create compressed or prioritized messages by using attention weights or learned encoding methods mimicking human communication in which the most salient information is prioritized over redundant messages [20].

The use of middleware platforms and fog computing has also improved communication performance in multi-robot systems. The latest configurations for the production of middleware (e.g., Robot Operating System 2, or ROS 2) demonstrate more capable messaging frameworks which allow for more real-time messaging, service calls and action servers, across a decentralized robotics network [21], [22]. DDS is capable of improved scalability and fault tolerance than previous generations of middleware. However, there are newer configurations of edge computing and middleware available today (e.g., FogROS 2), that allow robots to offload processing-heavy tasks to cloud or edge servers, reducing the local communication load and allowing for sophisticated communication functionalities, even in low-hardware robots, for complex collective behaviors like real-time SLAM, object recognition, and merging maps [21].

When utilizing simulation environments, be it Gazebo or Ignition with ROS 2, TurtleBots communicate using the Data Distribution Service (DDS) protocol, which handles publish/subscribe messaging, the operator of the ROS 2 middleware [22]. The DDS allows the TurtleBots to be a node in a distributed system that can publish sensor data, subscribe to control commands, or be able to broadcast coordination messages over a timeline service. Using DDS in simulation systems allows for the creation of messaging patterns that can isolate operations that act distinctly like research and industry, which usually include features for latency, packet loss, and bandwidth throttling, especially when using network emulators or

systems using NetEm. Within simulated implementations, topics like /odom, /cmd_vel, /scan, and own messages (ex. for map merger or task allocation) are messaging formats that communicate robot results to other robots and to the implementing assistant. The DDS layer supplies the discovery of processes for the robots and delivers communication following Quality of Service (QoS) settings (highlights of reliability, history, durability), where robots can enter or leave the communication network [23].

Communication in MRS applications based on ROS 2 is applicable and often used in the real world. The trick is the abstraction that ROS 2 provides: while the transport layer in a simulation may be virtual or loopback, in actual robots (e.g. TurtleBot3) the same DDS-based ROS 2 messages travel over real wireless networks, usually Wi-Fi (802.11) [22], [23]. The default middleware implementations of ROS 2 (e.g. Fast DDS, Cyclone DDS, RTI Connext) implement over standard IP layer transports (e.g. Wi-Fi, Ethernet) for the robots that the middleware implementations are directly usable in either simulations or physical robots [23]. As a contrast with Bluetooth and Zigbee which do not offer real-time or high throughput for MRS applications due to limited bandwidth and inadequate coordination among devices, overall performance implemented over Bluetooth and Zigbee does not be adequate.

2.4 Object detection techniques

Robots orient themselves to the world through a solid object detector as a minimum capability of autonomy within robotic platforms in multi-robot systems [24]. Real-time, robust object detection is needed for robots to navigate, manipulate objects, or collaborate with other robots. A variety of strategies can serve as object detection, but lightweight versions are often favored in UAVs where future autonomy requires inexpensive and performant computation. Algorithms tuned with MobileNet or quantized YOLO, for example, can help onboard detect humans or vehicles from onboard UAVs [26]. Likewise, the use of object detection in kinetic frames via visual serving and target tracking in robotics systems allows robots to close loop, which were demonstrated in the survey [25]. Several challenges such as recognizing small or occluded objects, recognizing objects under domain shifts and low-light conditions, high performance on low resource edge devices needs to be addressed while working with multiple autonomous systems.

Object detection has moved from manual feature-based methods to complex deep-learning based approaches, involving transformers and end-to-end learning strategies [25]. Object detection is still an active research area with new hybrid models, real-time performance

capabilities, and task-specific optimizations. Importantly, as systems that are multi-robot in nature become increasingly relevant, with major autonomous vehicle developments and edge computing devices, there will be an increased necessity for reliable and efficient object detection [26]. In the early days of object detection, it was grounded on classical methods involving image processing tools and techniques of machine learning. The prominent such methods were Viola-Jones and Haar cascades which had an edge over all methods with speed for real-time applications [24]. In its core, the Viola-Jones algorithm uses Haar-like features combined with an AdaBoost classifier, and face detection results can be quite satisfactory in confined conditions. Typically, these methods are associated with sliding window techniques and hand-coded features like Histogram of Oriented Gradients (HOG) or Scale-Invariant Feature Transform (SIFT) to locate objects in images [24].

However, these traditional methods struggled with scale invariance, occlusion, and generalization to different object categories. As a result, their usage is now limited to specific low-complexity or legacy systems [25]. The recognition of deep learning and its effects has led to the revolutionization of object detection in the computer vision space. While hand-crafted features or matching algorithms have always been integral to this approach, subsequently, Convolutional Neural Networks (CNNs) have replaced these with learned features or models based on learned features [25]. One important result of this transformation in the field is the added accuracy and flexibility for both object detection and optimal solutions. The algorithms that utilize CNNs, R-CNN and the family of algorithms that follow (Fast R-CNN, Faster R-CNN), and the YOLO (You Only Look Once) series (YOLOv5, YOLOv8) and SSD (Single Shot MultiBox Detector) have become standard [25], [27].

All research advances have delved into more efficient approaches to multi-scale feature extraction, while emphasizing transformer-based architectures. Vision Transformers (ViT), along with DETR (DEtection TRansformer), were transformational in the introduction of attention mechanisms for modeling global context to create better object detection models in more occlusion apartments [28]. For example, Zeng et al. proposed Hybrid-DETR, leveraging the benefits of CNN-based backbones with transformer decoders to improve the current class of models by utilizing small-object detection [28].

2.5 Situational awareness in robotics

Situational awareness is the awareness of what is happening around robots. This is important for robots as it helps them to react safely and intelligently in real time. There are three

essential elements to situational awareness for robots. The first is perception, or perceiving the environment through the assistance of cameras, LiDAR sensors, or other sensors. The second is comprehension, or the interpretation of information that is sensed. The third is projection, or the ability to anticipate what is likely to occur next based on the situation as it currently exists. Each of the three helps allow a robot to react with awareness instead of reacting blindly.

Many robot platforms today are designed to enhance perception through the use of different types of sensors. For example, some use LiDAR, RGB cameras, and depth sensors to achieve a more holistic understanding of the environment [29]. Such sensors help the robot to infer the existence of humans, objects, and other environmental factors. In one study, this sensor data was visualized in real time with zoom and filtering options, giving users an easy way to monitor both the robot and its surroundings. This helped improve safety and response time. Lightweight systems have also made progress in mobile settings. Using edge devices like Jetson Xavier NX, robots have successfully run object detection with YOLOv5 and human pose estimation with MediaPipe [30]. With ROS 2 used for data integration, the robot quickly processes sensor input and responds to changes around it. This setup not only improves perception but also supports real-time reactions in dynamic environments such as crowded spaces.

Comprehension refers to the extension of perception by the incorporation of meaning to the objects the robot is perceiving. For intelligent homes, the robot is now trained with data sets of RGB-D videos, audio, robot motion, and labeled human actions [31]. These cues are brought together through transformer-based architectures that allow the robot to understand human behavior. One more way to better understand is through question-asking systems for the robot. In the event the robot is not clear on the commands given to it, the robot can halt and clarify [32]. For instance, when one says, “bring the block,” the robot will ask back “you mean the red one or the blue one?” In this simple way, mutual understanding is better achieved. Errors are lessened and humans can trust the robot’s actions.

Accurate localization and map-building are also required for situation awareness. Some robots use spinning LiDAR sensors to build fine-grained indoor 3D maps [33]. Such maps enable the robot to detect walls, doors, and other obstacles. In more advanced systems, the robot builds semantic maps that not only show shapes but label objects like people, furniture, or

automobiles [34]. This increases the robot's ability to make the right decisions and navigate through hazardous environments more safely.

The third aspect, projection, helps the robot foresee the future. It is quite effective when humans and robots are collocated. In certain research, they use the LSTM networks to process sensor data such as motion, temperature, and IMU data. The robot learns patterns such as walking or stopping and uses them to foresee actions [35]. There are other robots that use wearable inertial sensors alongside RGB-D cameras to foresee the future actions of humans [36]. This helps the robot to plan ahead its actions and avoid collisions or help the humans. In factory environments, humans and robots work together. One of such system presents workers with virtual data on the robot's behavior with the help of augmented reality headsets. In the process, the robot keeps track of the worker's location and motion [37]. Such shared understanding leads to smoother and safer cooperation. Another approach instills the robot with primitive mental states and intentions of humans with the help of cognitive models. Predicting what the individual thinks or intends enables the robot to act likewise [38]. Such cognitive modeling keeps the surprises to the minimum and promotes trustworthiness in human-robot collaboration.

Situational awareness is just as important for high-risk or unforeseen environments. In maritime environments, robots have been designed with the incorporation of radar, LiDAR, cameras, and AIS data. Such data are managed with machine learning to detect hazards and chart safe navigation corridors [39]. The system is designed to operate on the cloud as well as on local equipment to permit flexibility to cope with low connectivity. In the event of disasters, robots are often required to take rapid decisions with partial information. Integrating information from thermal cameras, LiDAR, GPS, and vision feeds, the robots are able to map surroundings, detect victims, and disseminate such information across networks to permit prompt action by rescue teams [40]. These are all illustrations that provide evidence that situational awareness is not about using one tool or methodology. It is the combination of many data sources with smart algorithms and real-time reasoning. The better the robot is able to see, understand, and foresee its environment, the better it is able to act safely and constructively. If the robot is guiding the elderly through crowds, assisting within factories, or scanning through rubble when there is a disaster, good situational awareness helps it to be more than just a robot. It is able to act as a reliable assistant.

All these examples show that improving situational awareness is not about using one tool or method. It is about combining multiple data sources, smart algorithms, and real-time thinking. The better a robot can see, understand, and predict its environment, the better it can perform in a safe and helpful way. Whether the robot is guiding elderly people, assisting in factories, or searching through rubble after a disaster, strong situational awareness allows it to be more than just a machine. It becomes a reliable partner.

2.6 Multiple robot task allocation

MRTA (Multiple Robot Task Allocation) is the core problem of allocating optimally a set of tasks to a team of autonomous robots. MRTA is a fundamental aspect of multi-robot systems that enables robots to function intelligently in environments that may include use cases such as disaster management, factory/industrial operations, environmental monitoring, steering/speeding up search-and-rescue efforts [5]. The task allocation of multiple robots is fundamental to successful multi-robot coordination. The MRTA problem is also arduous owing to issues such as task interdependencies, heterogeneity of agents, uncertainty, limited communication, real-time constraints, etc. Challenges associated with multi-robot task allocation can be organized into three important dimensions. Situations can present themselves in the context of single-task robots versus multi-task robots, single robot tasks versus multi-robot tasks, and instantaneous assignment versus time-extended assignment [42]. These classifications determine how the problem will be handled by potential algorithms. For example, if tasked with a single robot task single-task robot, and if the algorithm is limited to greedy and auctions or similar methods, then either may suffice. However, in a multi-task, multi-robot environment, the algorithm is likely having to resolve to combinatorial optimization [42].

Modern MRTA systems are increasingly designed to be decentralized and autonomous, allowing agents to make local decisions with limited knowledge of global states. A notable 2024 study by proposes a ROS-based MRTA framework where a cluster of three micro-robots autonomously assigns and executes tasks in hostile environments using a greedy search algorithm [41]. Their system integrates AI logic, robust communication protocols, and real-time sensor fusion to dynamically assign tasks while minimizing energy consumption and task completion time. The robots operate in supervised and collaborative modes, coordinating tasks such as victim search and localization via LiDAR-based 3D digitization, while adapting to terrain changes and communication loss.

MRTA problems are often NP-hard, and the real-time criteria will force them to be solved via heuristics or approximations [42]. Many methods have been proposed for multi-robot task allocation including greedy algorithms which are simple and fast, are recommended for smaller tasks, market-based allocation which methods in which the robots bid for tasks based on their utility, cost, or priority, swarm intelligence, which draws inspiration from biological systems of swarm behavior, and market-based methods including robots bidding for tasks based on priority, utility or cost [41]. Swarm intelligence methods like ant colonies or bee swarms are also factors to consider where the robots operate autonomously [4], [5]. More advanced methods have emerged using genetic algorithms and reinforcement learning methods which allow for changing conditions and rescheduling of tasks [4]. Improvements in MRTA have come from using AI and machine learning heuristics to suggest adaptive allocation. Tamali et al. firstly configured their MRTA architecture using the Robot Operating System (ROS), and other simulation simulators, Gazebo and land-based hardware, JetBot-based [41]. Within their use case, they also provided collaborative mapping, ODOM transformation, and energy-aware motion planning. The MRTA logic is within URDF/ROS.MSG modules for compatibility and scalability. Some of the biggest obstacles in MRTA, aside from the algorithms being directly related to the problem complexity, fall under global systems considerations [42]. Scalability is a major issue, ensuring a vigorous performance with increases in agents. Reliable low-latency communications will be crucial, especially when faced with busy environments and variable traffic [42]. The interdisciplinarity of the robot teams, which may have different capabilities, sensors, and payloads to track, will make deliberating on coordinated tasks more difficult. Additionally, fault tolerance must be considered, where tasks must be reassigned efficiently if an individual robot fails. Aiming for system fairness and avoiding starvation with regard to task assignments means that reward functions should be specifically designed or iteratively improved through feedback for inspection.

New and constant research in MRTA is becoming more focused than before on increasing their scalability, adaptability, and use in real-world scenarios. Inspiration from nature via bio-inspired coordination patterns can provide systems that are resilient as well as scalable [3], [4]. Researchers are also considering task allocation strategies based on learning, which allow robots to dynamically allocate tasks based on changing environmental events [4]. Furthermore, there is a shift away from solely distributed or centralized methods for task allocation to combinations of cloud-edge hybrid architecture that emphasizes centralized

planning and optimization with the robot's ability to respond in real-time through local autonomy. Also, advances in onboard computation capabilities, sensor networks, and low-power AI chips have made gathering data and executing alternative models solely from an idea easier, moving zones of MRTA closer together in field robotics applications [5], [28].

2.7 Applications of multi-robot systems

A rapid growth and evolution in the domain of multi-robot systems has been obtained resulting in delivering notable advances in a number of sectors. State-of-the-art design of multi-agent deep reinforcement learning made robots collectively handle sophisticated tasks in a more efficient and adaptable manner than before [43].

MRS excels in large exploration and area coverage. Leveraging the MRS with deep reinforcement learning, MRS can comprehensively survey, generate more accurate maps of unknown environments, and monitor high-risk environments [43]. Such approach gives the system an edge in the domain such as search and rescue operation in known or unknown environments, high-risk environments such as fire disaster zones, precision agriculture, optimizing field operation, crop analysis, ocean and environmental monitoring where distributed robots significantly improve the coverage and optimize the data reliability [43], [44]. Another significant application of MRS is path planning and navigation. Leveraging advanced algorithms such as Proximal Policy Optimization (PPO) and Multi-Agent Deep Deterministic Policy Gradient (MADDPG), multi agents can learn collision avoidance, efficient route planning, and cooperation in the navigation tasks [43]. Addition of networks such as Graph Neural Networks (GNNs) and Long Short-Term Memory (LSTM) networks gives an edge in real-time centralized decisions, enabling robots to navigate complex setups in an intuitive human manner making it best suited for practical setups such as coordinated autonomous vehicles, drone swarms, and logistical operations in warehouses, where smooth navigation boosts both safety and productivity [43].

Another remarkable application is swarm robotics which leverages MRS by enabling collective behaviours such as flocking, formation maintenance, and group transportation of objects [45], [46]. Robotic swarms adjust in their environment dynamically, maintain a stable formation, and react group wise to obtain or change goals or external environments or events through MADRL enabling a robust decentralized learning system which makes the group manipulation and adaptive responses to a superior level highlighting the scalability of MRS in both industrial and field environments [43].

For the modern industrialization, the value of MRS in collaborative industrial settings where interaction between humans and robots are key priorities in any human-robot collaborative scenarios spanning manufacturing, logistics, and beyond. Collective integration of artificial intelligence, reinforcement learning, and distributed communication in MRS enhancing their ability to deliver coordinated, scalable, and intelligent robotic operations, are poised to transform a wide array of fields.

2.8 Critical challenges of multi-robot systems

Though multi-robot systems hold great potential in a wide range of applications such as search and rescue, exploration, and other complex applications, however, there are a number of challenges which impedes their wide range of deployment and effectiveness.

In the unpredictable and harsh environment achieving robust communication among MRS seems quite difficult. Wireless signals may collapse in disaster zones resulting in interference and delays degrading bandwidth and increasing latency which not only disrupt coordination but also compromise real time data sharing, a critical element for effective MRS in search and rescue operation [47]. Precise localization becomes a formidable challenge in underground or collapsed structures making GPS signals unavailable [48]. Sensor fusion, and vision-based methods such as SLAM which has been implemented in the simulation contributes to improve the navigation but still face challenges like sensor drift, dynamic obstacles, and the unpredictability of localization making path planning and collaborative actions suffer from increased error rate [48].

Complexity of coordination increases exponentially as agents increase in size in MRS [47]. Traditional approaches such as behaviour-based and market-based task allocation often fail to scale effectively resulting in conflicting behaviours at times [50]. Moreover, integration of heterogeneous robots with different sensing, actuation, and computational capabilities increases these issues accentuating the need for adaptive coordination mechanisms [49]. While simulation demonstration shows an edge, translating these achievements to real world deployment introduces several complexities particularly in SAR operation which demands more robust algorithms and system architectures capable of adapting to real-time uncertainties and physical constraints [48]. Collectively these challenges from communication and localization to coordination, task allocation, and real-world implementation from a simulation environment MRS form a multi faceted barrier that must be addressed to achieve their potential in dynamic and high-stake environments.

3 Methodology

The methodology section focuses on the selection of software, software libraries, packages, system architecture, and the design of the workflow along with presenting the structured approach followed in the development and simulation of the multi-robot coordination system. It begins with presenting detailed description of the hardware and software prerequisites, focusing on key packages such as ROS 2 Humble for middleware communication [23], Gazebo Fortress for simulation [51], Nav2 for autonomous navigation [52], and YOLOv8 [53], OpenCV [54], and PyTorch [55] for object detection by integrating deep learning image processing which is later progressed towards showing diagrams how the workflow of these multi-robot systems architecture works in both AMCL [56] and SLAM [57] localization.

3.1 System architecture and software components

For hardware architecture, a workstation or computer with the minimum specifications(recommended) of CPU of Intel i7 or higher, and RAM of 8 GB or more is recommended. The simulation here has been performed with Intel Core Ultra i7 (Max Turbo 4.8 GHz, 16 cores), and RAM of 16GB LPDDR5-6400. Ignored AMD based processor. Alongside the hardware configuration, several software were required to perform the simulations correctly. Though utilizing the virtual machine in windows-based operating system or iOS-based operating system, simulation could have been performed but for real time optimized performance measurement Ubuntu based open-source operating system has been used and the version was Ubuntu 22.04.5 LTS [58].

For the simulation along with seamless communication, robot operating system ROS 2 Humble along with compatible version of Gazebo, Gazebo fortress has been utilized. RViz [59] which is a 3D visualization tool for ROS has also been used which helped to visualize the robots' movement, obstacles viewed by the robots, and for the programming language Python 3.11 [60] has been used but any version above 3.10 is also recommended.

A number of packages such as turtlebot3_gazebo [61], nav2_bringup [62], rqt [63], rviz2 [64], tf2_ros [65], YOLO [53], OpenCV [54], Pillow [66], screeninfo [67], ultralytics [68], PyTorch [55] were installed and utilized for a reasonable and smooth simulation. For the turtlebots to detect obstacles and the target for rescue, LiDAR [69], and Intel RealSense R200 [70] have been integrated. Details of installed and worked software, packages, and components are discussed as well.

Simulation of environments and experiment before implanting in real-world environment is a must for testing algorithms, validating system designs, and reducing risks associated with physical experimentation. In modern robotics research and development, Gazebo is one of the most widely adopted simulation platforms for robotics. It offers a balance of precision, flexibility, and integration with middleware such as the Robot Operating System (ROS) which enables users to simulate robots and environments in three dimensions with realistic physics. It supports the use of sensors such as cameras, lidars as well as a variety of actuators. Developers can test control strategies, navigation stacks, and perception pipelines within a safe and configurable environment. The platform also allows adding custom plugins to model specific behaviours, making it highly adaptable for different research objectives.

There are several other simulation software available such as Isaac Sim, Webots, CoppeliaSim, MORSE and so on other than Gazebo. While Isaac Sim provides highly realistic rendering and GPU-accelerated physics suitable for AI training, but it has high complexity and hardware requirements which makes it less accessible for lightweight multi-robot testing. Though Webots is user-friendly and well suited for educational purposes, but it lacks the physics realism and scalability offered by Gazebo. CoppeliaSim is adaptable with strong kinematic support and remote APIs but has limited support of native ROS 2. A comparative overview of prominent robotics simulation planforms can be observed in Table 1 evaluating key features. From the comparative analysis with other similar simulation platform Gazebo shows more promising option. With limited computational resources and power, Gazebo puts a decent balance between physics accuracy, modularity, and seamless ROS 2 integration which is a must for such comprehensive robotic simulations.

Table 1. Gazebo vs other simulation platforms

Feature	Gazebo	Isaac Sim	Webots	CoppeliaSim	MORSE
Developer	Open robotics	NVIDIA	Cyberbotics	Coppelia robotics	LAAS-CNRS
License	Open source	Free (non-commercial)	Open source	Free (limited commercial)	Open source
Graphics	Moderate	High (realistic)	Basic	Moderate	Basic
Physics	ODE, bullet	PhysX (GPU-accelerated)	ODE	Multiple engines	Limited
ROS support	ROS 1 and ROS 2	ROS2 (bridge)	ROS 1 And ROS 2	ROS 1	ROS 1
Sensor simulation	Strong and customizable	Advanced, realistic	Basic sensors	Good variety	Limited

Feature	Gazebo	Isaac Sim	Webots	CoppeliaSim	MORSE
Best for	General robotics	AI, vision, data, generation	Education, small robotics	Arms, hybrid control	Teaching, simple tasks

To interpret and display sensor data, robot states, and environment models in real time RViz (ROS Visualization) a 3D visualization tool is used in the Robot Operating System. It is primarily used for debugging, monitoring, and validating robot behaviour in both simulated and real-world applications. Data published over ROS topics such as laser scans, images, maps, transformations (tf), and the robot's pose during the simulation users can visualize them using RViz. While performing the simulation in this study, RViz was used to monitor the simulated TurtleBot3 robot in the Gazebo environment, visualizing LiDAR scans, SLAM mapping, camera images, and the robot's planned trajectory during navigation.

Four robots were used in each of the AMCL and SLAM based localization for the SAR operation. To have a robust, conflict free communication, deployment, movement, navigation, share information without getting mixed for decentralized task allocation, and target detection ROS 2 Humble was used. ROS 2 Humble is built on top of the Data Distribution Service (DDS) middleware which enables real-time communication resulting in efficient decentralized system design. It supports multiple programming languages (primarily C++ and Python), and provides core tools for message-passing, node-based architecture, parameter handling, lifecycle management, and multi-robot systems. In the experiment, ROS 2 Humble was used as the core middleware to interface simulated sensors such as LiDAR and camera, perform sensor fusion, and control robot behaviour in a TurtleBot3-based Gazebo simulation. ROS 2 Humble’s compatibility with the Nav2 stack, lifecycle nodes, and modern launch system makes it easier to manage complex behaviour trees and parameter configurations required for decentralized task allocation and real-time object detection using YOLOv8.

To design and deploy the robots, turtlebot3_gazebo package is utilized. This package provides simulated TurtleBot3 models for use with the Gazebo simulator. The available models are Burger, Waffle, and Waffle Pi [71]. The TurtleBot3 Waffle model is a small, mobile robot platform equipped with a 2D LiDAR sensor and a front-facing RGB camera. It is designed for research and development in robotics, offering both environment perception through LiDAR and through the camera. As our study requires visual data for object detection along with LiDAR data for navigation and environment sensing, the Waffle model is the best choice. It

provides the combination of sensors needed to run detection algorithms on camera images while simultaneously performing mapping and obstacle avoidance using LiDAR. To perform simulation, it is required to spawn the robots into the Gazebo world using .sdf or .urdf files. In this system, it is used to define robot models, spawn them into custom maps, and manage simulation physics and sensors like LiDAR and camera.

As stated, each TurtleBot3 Waffle robot in the simulation is equipped with a 2D LiDAR sensor enabling the robots to perceive the surrounding environment. This LiDAR is not used for identifying specific targets but strictly geometric awareness such as obstacle detection, environment mapping, and safe path navigation. The sensor data is consumed by SLAM, navigation, and exploration node. The LiDAR is mounted above the robot's 5cm from the base and provides a full 360-degree view around the robot. It captures distance measurements every 0.5 degrees resulting in a total of 720 data points per scan. The sensor can detect objects as close as 12 cm and up to a maximum range of 3.5 meters. It performs 10 scans every second resulting in a real-time update. The sensor uses the gazebo_ros_ray_sensor plugin which automatically publishes data as sensor_msgs/msg/LaserScan to the appropriate topic in the robot's namespace. The sensor data is published to the base_scan frame and each robot broadcasts its LiDAR output on a unique topic /tb3_X/scan where X is the robot's identifier. In Gazebo Fortress, the LiDAR beam is visualized as a rotating blue ring of rays cantered on the robot. Each ray simulates a laser pulse and is terminated when it hits a solid surface in the environment. The result is a dense 2D scan of obstacles around the robot, rendered in real time.

The Intel RealSense R200 is an early-generation depth-sensing camera designed for real-time 3D perception. It features an infrared stereo depth system combined with an RGB sensor, allowing it to produce synchronized depth and colour images. In the simulation, the R200 was simulated on a TurtleBot3 within the Gazebo environment to provide virtual RGB-D data. It publishes RGB images via OpenCV and ROS. Images are passed into a YOLOv8 model leveraging the ultralytics library. The camera model publishes both colour and depth streams through ROS topics. It is used exclusively for detecting the target for rescue in the environment.

In this system, LiDAR and camera sensors are used for different goals. The LiDAR sensor provides 2D scan data for geometric obstacle avoidance and SLAM-based map building. It cannot identify objects as all surfaces are treated equally. The camera, on the other hand, is

used exclusively for semantic detection using YOLOv8 [53]. It identifies the specific target visually and informs the navigation system to respond. Both sensors publish to ROS 2 topics independently and are handled by different nodes. This separation of perception pipelines allows the system to treat navigation and detection as modular components.

For target detection, which is set as red cylindrical object in this study, leveraged YOLOv8, Ultralytics, OpenCV, Pillow, and screeninfo. YOLO is a real-time object detection model. It is used in the system to visualize the target from the camera image. Through ultralytics, the YOLOv8 model runs inside smart_final_multi_yolo_detector.py, takes in camera frames, and outputs target bounding boxes. This solves the semantic recognition which notifies the robot what it is looking at, not just where things are. A study that compared YOLO-based detectors with traditional R-CNN models indicates better speed with at least the same accuracy [25]. The main challenge with YOLO and many other models is that it treats object detection as a regression problem, mapping pixels from a given image directly to the bounding boxes around the object of interest and to class probabilities.

Ultralytics package is the official frontend for YOLOv8 which is developed and maintained by Ultralytics [68]. This python library offers a high-level API for training, evaluating, and deploying deep learning models with a focus on speed, accuracy, and ease of use. It is used in smart_final_multi_yolo_detector.py to load best.pt and run detections on camera frames. It solves the problem of real-time, low-latency object recognition inside a ROS 2 system. The best.pt file is the trained YOLOv8 model file. It helps recognize whether a given image contains the target object which is a cylinder in our simulation or not. This file stores all the learned patterns and is used during detection.

Since need to check each camera frame and decide whether a red cylinder is present, the best.pt file becomes essential. It analyses each image and determines if the target red cylinder is present. In the simulation, it takes live camera frames from the robot, loads the best.pt model, and performs detection on each frame. This is relevant because the file enables the robot to identify and respond when it sees the red cylinder.

Ultralytics requires PyTorch library which helps ultralytics to run detection. It solves the problem of executing high-performance neural networks inside python applications. It is the underlying deep learning engine that powers the YOLO model. It is an open-source deep learning framework developed by Meta AI. This handles model weights, tensor operation, and

GPU, CPU interface. To convert camera massages to image arrays, resize them, and feed them into YOLO OpenCV is used. It is a computer vision library.

OpenCV is used in the YOLO detection script to handle sensor_msgs/image, apply pre-processing, and optionally visualize results [54]. OpenCV is utilized as YOLO expects NumPy image arrays, not ROS image messages. It connects ROS sensor data with ML pipelines.

To visualize, log YOLO detections during runtime, Pillow is used in the system. It is used for image formatting tasks. It draws bounding boxes, resizing, or saving detected frames. It is not core to detection but helps when post-processing or debugging predictions. It solves auxiliary tasks around detection output.

To position the GUI window that displays every frame of the video with detection results, screeninfo, a python library, is used. The screeninfo library is used to get the monitor's resolution and layout. It helps automatically place windows, such as debug or video output windows, on the correct screen. In the system, this is used to position the GUI window that displays every frame of the video with detection results. This ensured the YOLO output window opened neatly in a fixed, centered position instead of appearing randomly or off-screen.

To summarize the target detection technique in the simulation utilizing these packages and libraries, YOLO via the ultralytics library is first used to quickly detect the target red cylinder, in images or video frames. OpenCV helps process these images by handling tasks like reading camera frames, resizing, and drawing detection boxes around identified targets. Then there comes the pillow which is used to manage image formats and perform image manipulations needed. Together, these tools work in a workflow where live camera images are captured, processed, and analysed in real time to identify specific targets. In real-world applications this library is mainly used where there is a task of object detection from the camera. Ultimately, allows robots to efficiently locate and detect objects in environments, speeding up decision-making and improving accuracy.

To manage coordinate frame transformations between different parts of the robot such as map, odom, base, sensor, tf2_ros library is used [65]. Map frame represents fixed environment or world, odom frame tracks the robot's movement over time, base frame is the main point on the robot's body used to track its position and movement, sensor frames are

used to show where each sensor such as LiDAR, camera is place on the robot and help tracking the position and direction of the sensors separately from the robot's main body. It is required in the system to integrate multiple sensors and localization sources. Here, wheel odometry is one of the localization sources. It uses data from the robot's wheel movements such as how much each wheel has turned to estimate the robot's position and orientation over time. This helps the robot keep track of where it has moved.

In the simulation, tf2_ros allows LiDAR and camera data to be aligned correctly with the robot's base and navigation frames such as if the LiDAR sees an obstacle 3 meters ahead, the robot also needs to know where the LiDAR is on its body to figure out where that obstacle actually is in space. Without tf2_ros, the robot might get the distance but not the direction, making the data useless for navigation or detection. So, without spatial consistency, even if sensors detect things, the robot won't be able to respond properly because it doesn't know where those detections are coming from. tf2_ros solves this problem by keeping all the coordinate frames connected and updated. For navigation, nav2_bringup package is utilized and it launches the full Navigation 2 stack which includes global planners, controllers, and costmap servers. It is necessary for autonomous robot movement and real-time path following on SLAM and obstacle data.

In the designed system, Navigation 2 enables each robot to avoid dynamic obstacles, replan when blocked, and follow goal paths. For autonomous movement in unknown or changing environments, this package is needed. For real-time visualization of ROS topics, node graphs, parameter trees, and plugin-based data views, debugging message flow, validating system behaviour, and fine-tuning algorithm performance, rqt, a Qt-based GUI framework is used. It helps to visualize how different parts of the code are running, which nodes are active, how they are connected, and what data is flowing between them. This way can look into rqt and quickly understand what's going on inside the system without digging into each code file. It is not essential to robot logic but is heavily used during development and debugging phase to visualize node graphs (rqt_graph), view images (rqt_image_view), or monitor different topics. Table 2 represents the list of topics monitored during the simulation using the rqt framework. These topics includes monitoring perception, navigation, and system diagnostics.

Table 2. ROS 2 runtime topics observed using rqt

Topic name	Purpose
/camera/image_raw	Raw camera image feed visualized using rqt_image_view

Topic name	Purpose
/camera/camera_info	Intrinsic camera parameters
/yolov8/detections	Object detection outputs (bounding boxes, class, confidence)
/scan	LiDAR sensor readings for obstacle detection
/odom	Robot's odometry data
/tf	Coordinate frame transformations
/tf_static	Static transforms such as base_link to camera
/amcl_pose	Pose estimate from AMCL
/map	The occupancy grid map
/goal_pose	Goal position to navigate toward
/cmd_vel	Velocity commands sent to the robot
/nav2_bt_navigator/behavior_tree_log	Logs of navigation behaviour trees (for debugging Nav2)
/parameter_events	Monitor parameter changes in nodes
/rosout	Logging output from all nodes

Utilizing these software, and packages, the system simulates TurtleBot3 robots in a shared Gazebo environment. Each robot explores unknown regions autonomously, and coordinates to detect red cylindrical target. The ROS 2 nodes are organized under dedicated namespaces per robot. In both AMCL and SLAM localization, the robots explore the environment and reaches the target once detected for rescue while maintaining closest robot reaching the target in the AMCL localization.

3.2 Workflow of conducted simulations

First, a static map of the environment is created with minimal configuration. Then, all the robots are launched into the world independently. The navigation framework is run on the static map. All robots autonomously explore the environment to detect the target.

For target detection, a labelled image of the target is trained using the YOLO-v8 model, which is then used to detect the target. Once detected the target, triggers the coordination logic, influencing robot actions, and the world coordinates of the target are calculated. The closest robot among all is then identified and assigned the goal of navigating to the target. Functionality is split across custom ROS 2 packages. This modular approach aids debugging and allows for independent testing and updates of each subsystem.

The workflow for the robots to search and rescue the target in autonomous manner leveraging the AMCL localization is shown in the Figure 1. The process starts with defining the number of robots and assigning each one a unique namespace so that though each robot contains the same topics such as camera feed, velocity, or laser scan, their data or messages does not mix with other robot's data, and messages ensuring work independently without interference. After setting the world and map paths, each robot's URDF file is created and the robot is spawned into the simulated environment. Every robot is equipped with a LiDAR sensor and a camera, allowing automatic mapping and exploration of the environment. Each camera continuously scans for a pretrained target object using a YOLO-based real-time detection model. During exploration, if a target is detected by any robot, its image coordinates are converted into global world frame coordinates and sent to a Nav2 path validator to check path feasibility. All robots then stop exploring while the system identifies the robot closest to the target, based on their current positions. The control node sends the goal position to the nearest robot via Nav2, and this robot autonomously navigates to the target, ensuring the task is efficiently handled by the most suitable robot. This workflow enables coordinated multi-robot exploration, real-time detection, and optimized task assignment in an unknown environment.

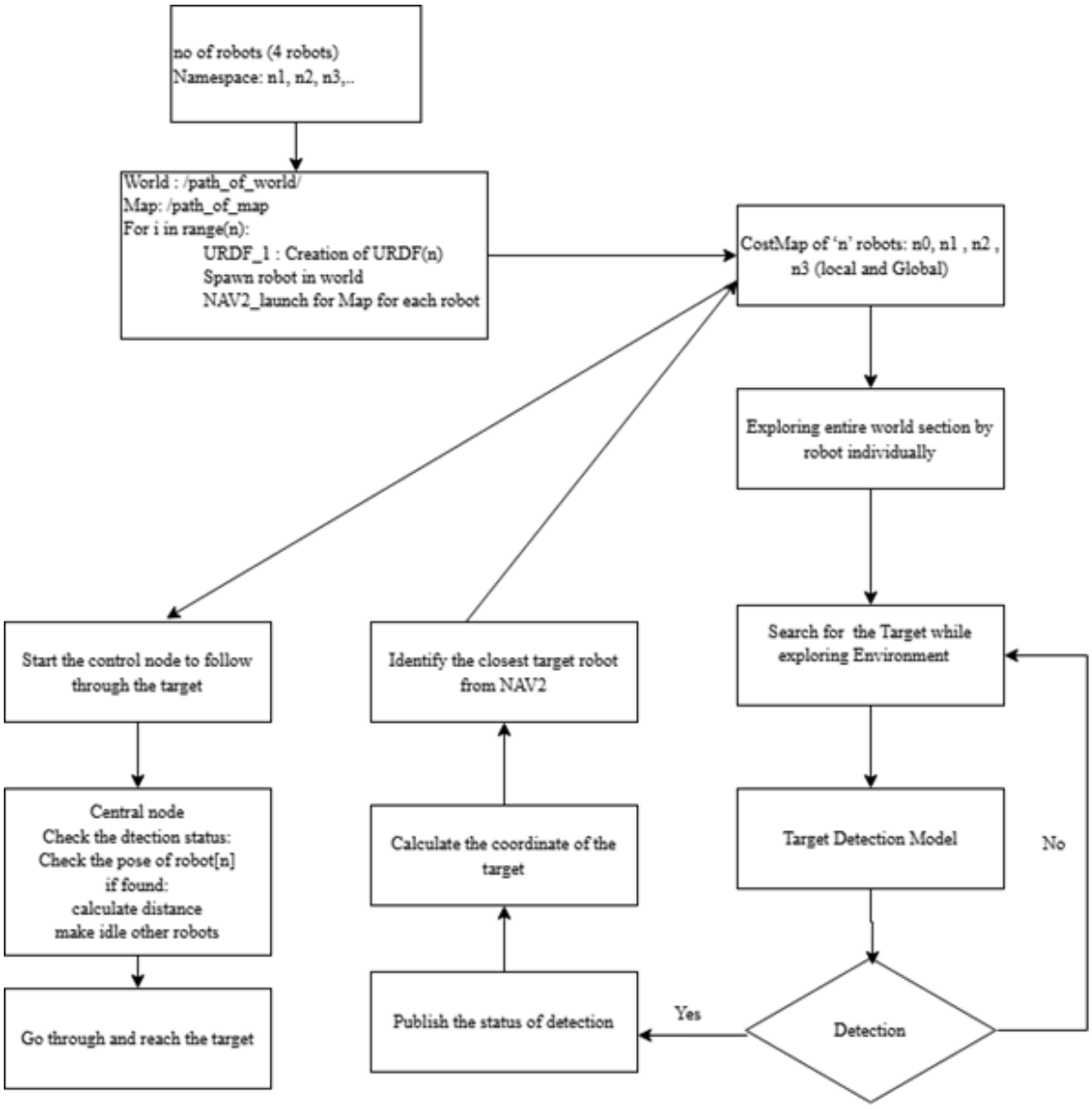

Figure 1. Workflow in AMCL localization

The workflow for the robots to search and rescue the target in autonomous manner leveraging the SLAM localization is shown in the Figure 2. Similar to AMCL localization simulation, four robots with unique namespace also deployed here each equipped with 2D LiDAR sensor for SLAM based mapping and automatic exploration as well as a camera for continuous scanning of a predefined target object. After setting the world and map paths, a URDF file is generated for each robot, which is then spawned into the simulation. As the robots explore their environments individually, they utilize the LiDAR for real-time mapping and obstacle avoidance, while the camera integrated with a YOLO-trained model performs real-time detection of the target object. When a robot's camera identifies the target, the image

coordinates are converted into global world frame coordinates. Given that a complete map is not available, the global goal coordinates are sent directly to the detecting robot. The control node then takes over by checking the robot's pose, aligning its direction along obstacles, and guiding it towards the target. The robot that detects the target halts further exploration and focuses on reaching the destination, while the remaining robots continue their search for the second red cylinder target. This process continues in a coordinated fashion, with detection status published for all robots, until successfully reaches both the targets, thereby completing the localization and tracking objective.

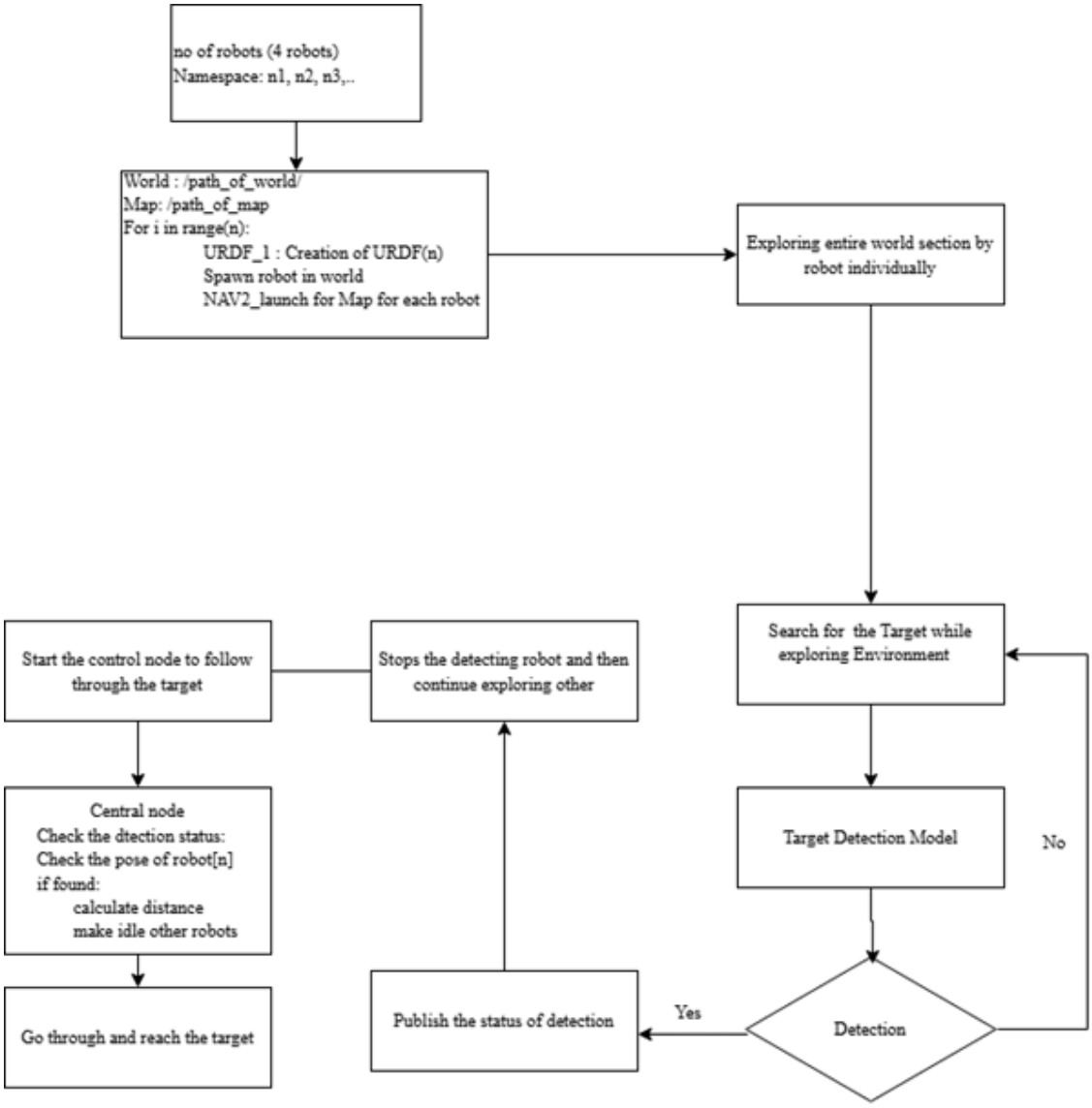

Figure 2. Workflow in SLAM localization

3.2.1 Testing and validation

The entire system is tested in a simulated Gazebo environment with RViz used for real-time visualization of the robot states, sensor data, and navigation path. Both SLAM and AMCL localization-based systems were launched and later verified by observing navigation behaviour against preloaded and dynamically generated maps. Navigation of the gazebo world reflecting real world rescue scenario, react to sensor inputs, and execution of planned trajectories based on map updates by the robots were monitored to ensure the path planning and exploration was performed correctly.

The system's communication efficiency, decentralized task allocation, and target detection mechanism were thoroughly tested. Visualization tool such as rqt_graph was used to monitor communication between robots and system nodes using ROS 2 topics.

Target detection via YOLOv8 was validated through multiple detection scenarios, confirming promising target recognition, and position broadcasting. Decentralized task allocation was tested by observing the runtime decision making process ensuring all robots responded appropriately to detection events, claim the target without any conflict, and executed coordination logic as designed. These validation steps ensured that all critical components of the system performed cohesively within the simulated multi-robot environment.

4 Experimental setup and implementation

Experiment and evaluations conducted using two primary localization and mapping techniques—AMCL (Adaptive Monte Carlo Localization) [56] and SLAM (Simultaneous Localization and Mapping) [57]—to optimize robotic navigation in search and rescue operations. Both experiments were carried out within a ROS-based simulation environment integrating Gazebo and RViz. The general setup, as described in the AMCL experiment, remains consistent across both trials, including the use of navigation stacks, robot configuration, and simulated rescue scenarios. However, while the AMCL experiment utilized a pre-defined static map for localization, the SLAM experiment required the robot to dynamically build and navigate an unknown environment without prior mapping. This distinction allowed for comparative assessment of performance, robustness, and applicability of each method under varying operational constraints.

4.1 Workspace and robot modelling

The model of the robot for this project is setup as “waffle” model where each robot output topic is explicitly setup as the /namespace/topic and the four urdf robot model is created. Four robots used under unique namespace in the simulation are described in Table 3. Assigning each TurtleBot3 with unique namespace ensures isolated communication and transform management with the ROS 2 environment allowing the robots to operate simultaneously without topic conflicts during exploration and navigation.

Table 3. Robot namespaces

Robot	Base Frame	Namespace
1	tbl/base_link	/tb1
2	tb2/base_link	/tb2
3	Tb3/base_link	/tb3
4	tb4/base_link	/tb4

In the MRS, all TurtleBot3 units run the same set of nodes that publish and subscribe to standard topics like /cmd_vel, /scan, /odom, and /tf. If all robots used the same topic names globally, their messages would mix, leading to confusion and malfunction. To prevent this, each robot is assigned a unique namespace as shown in Table 3 such as /tb1, /tb2, /tb3, /tb4 which acts like a separate folder for its ROS topics and transform frames. To elaborate this with an example from the simulation robot 1’s velocity command topic becomes

/tb1/cmd_vel, and its base frame is tb1/base_link. Similarly, robot 2 uses /tb2/cmd_vel and tb2/base_link, and so on. Such namespace separation ensures that even though the robots are running identical code and performing similar tasks, their data streams remain logically independent.

Four TurtleBot3 robots spawned into the maze-like gazebo world as shown in Figure 3. The blue rays from the robots are from the 2D LiDAR which can detect objects or obstacles as close as 12 cm and up to a maximum range of 3.5 meters. The LiDAR is mounted above the robot's 5cm from the base and provides a full 360-degree view around the robot.

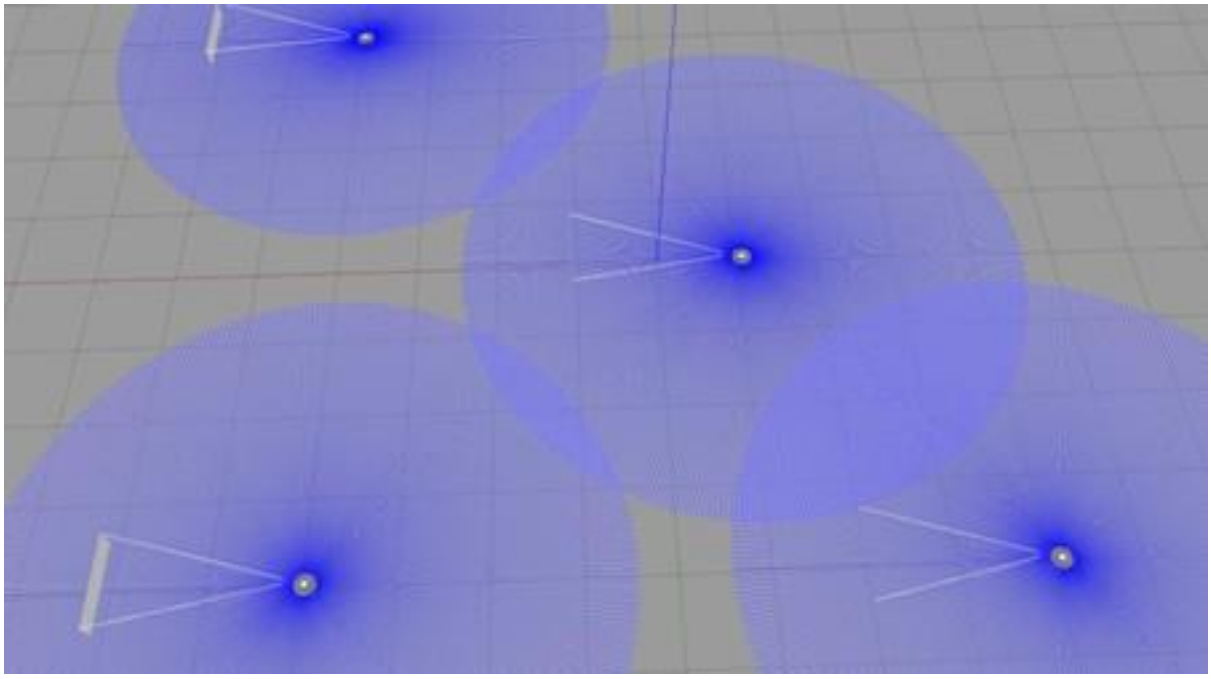

Figure 3. Robots exploring the environment

For completing the simulation aligning with search and rescue operation, obstacles-based gazebo world environment has created showing in Figure 4. This maze-like gazebo world is created utilizing the building editor feature of Gazebo simulator to mimic a real-world environment filled with obstacles.

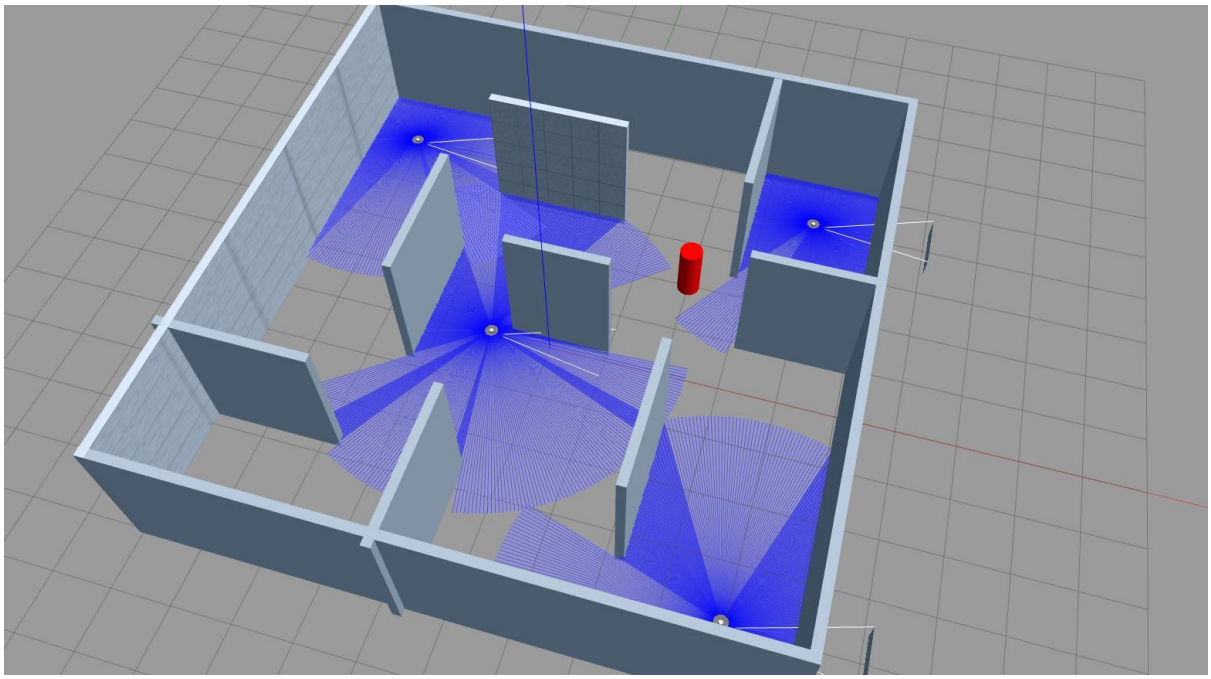

Figure 4. Gazebo world

Table 4 describes parameters for configuring the environment map used for robot localization and navigation in the simulation. This specifies the map image file, resolution, and coordinate origin, these three together determine how the robot aligns the map in AMCL localization with its operational environment.

Table 4. Map configuration parameters AMCL localization

Parameter	Value	Description
image	Maze1.pgm	Filename of the map image representing the environment
mode	Trinary	Map mode indicating three possible cell state (occupied, free, unknown)
resolution	0.05	Size of each map cell in meters (5 cm per pixel)
origin	[-14.8, -14.2, 0]	Real-world coordinates of the map's origin (bottom-left corner) and rotation
negate	0	Whether to invert pixel values (0 = no inversion)
occupied_thresh	0.65	Threshold above which a cell is considered occupied (obstacle)
fresh_thresh	0.25	Threshold below which a cell is considered free space

The robot is spawned in the gazebo world. The robots are driven and hence create the static map using SLAM shown in Figure 5. While robots are driven around the environment LiDAR

sensor scans the surroundings by emitting laser beams and measuring their reflection. The LiDAR data is processed using the SLAM toolbox node slam_toolbox in ROS 2 along with odometry. It builds a map while also estimating the robot's position. They grey areas in the Figure 5 represents unknown or unexplored regions the robot's sensors have not seen yet. The white areas show free or navigable space where the robot has sensed and found no obstacles. The black borders in the Figure 5 indicate obstacles or walls of the created gazebo maze-based world where the LiDAR detected solid surfaces.

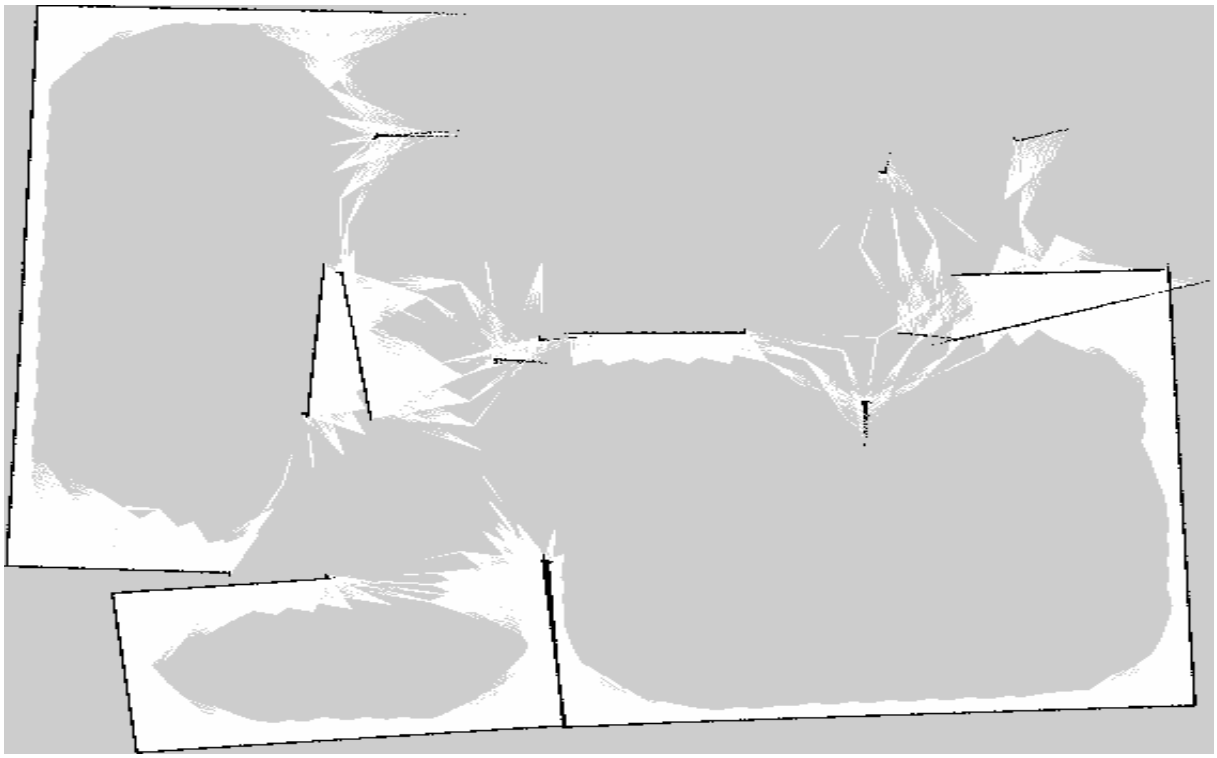

Figure 5. Early map in SLAM localization generated by the robots LiDAR

Configuration for SLAM based localization is summarized in Table 5. Using the SLAM toolbox node, how each robot with unique namespace manages coordinate frames, scan data, and updates map is defined. How frequently should the SLAM map updates is controlled by parameters like minimum travel distance and heading control while use_sim_time ensures synchronization with the simulation clock.

Table 5. Map configuration parameters SLAM localization

Parameter	Value	Description
use_sim_time	True	Use simulation time from the /clock topic
odom_frame	name + '/odom'	Name of the odometry frame for the robot (unique per robot via namespace)

Parameter	Value	Description
base_frame	name + '/base_footprint'	Base frame of the robot (usually fixed to the robot base)
map_frame	name + '/map'	The map coordinate frame
scan_topic	'scan'	Topic name for laser scan input (assumed to be remapped for namespace)
minimum_travel_distance	0.3	Minimum distance the robot must move before updating the SLAM map
minimum_travel_heading	0.3	Minimum heading change before updating the SLAM map
resolution	0.0	Resolution of the generated occupancy grid map
slam_toolbox_node_name	name + '_slam_toolbox'	Name given to the SLAM node (unique per robot)
debug_logging	True	Enable detailed debug logging output

For every robot a robot_state_publisher node is created. This node is responsible for publishing the robot's joint states and coordinate frame transformations based on its URDF (Unified Robot Description Format) model. This node helps the robot understand how its parts are arranged and how it fits into the world around it. By doing so, it allows other parts of the system such as the navigation stack or RViz to understand how the robot's body is configured and how it moves in the environment. The node operates using simulated time if the use_sim_time parameter is enabled. This is essential when running inside Gazebo or other simulation environments where system time must be synchronized across all nodes. Each robot is also launched within its own unique namespace such as /tb1, /tb2, /tb3, /tb4. This unique namespace ensures that multiple robots can coexist in the same simulation without topic or parameter conflicts. Additionally, the state publisher broadcasts the transforms over /tf and /tf_static topics. These are crucial for enabling proper localization, path planning, and visualization of each robot in the simulated world.

The spawn_turtlebot3_burger node spawns of multiple robots tb1, tb2, tb3 and tb4 with namespaces. Alongside the state publisher, each robot is also spawned into the Gazebo simulation using the spawn_entity.py node provided by the gazebo_ros package. In simpler terms, this is what actually drops the robot into the virtual world and tells it where to start. This node loads the TurtleBot3 SDF model and places the robot at a specific position in the simulation, defined by the x_pose, y_pose, and z_pose from the robot's configuration. Each robot is assigned a unique entity name and namespace to avoid conflicts. The -unpause argument ensures that the simulation starts immediately after spawning, enabling the robot to

begin interacting with the environment right away. This step is essential for enabling multi-robot simulations where different robots start exploring or performing tasks from different locations. By organizing the robot spawning process in this way, the system ensures that all robots are properly instantiated, uniquely identified, and ready to operate within a shared environment.

4.2 Navigation setup

Each robot runs the navigation stack and its lifecycle manager within its own namespace. The navigation stack in ROS 2 is a collection of tools and nodes that help a robot plan its path and move toward a goal while avoiding obstacles utilizing sensor data such as LiDAR, robot's map, goal position. The lifecycle manager controls the startup and shutdown process of the navigation stack. Navigation stack helps moving the robots autonomously so the robot can navigate a space on it own without manual control. Running each robot's navigation stack within its own namespace ensures their topics and data are separate which prevents confusion or errors while multiple robots are running at the same time. It helps to avoid conflicts in topic names and ensures robot follows its own map, commands, and goal. The configuration file is params.yaml, where all parameters are customized to ensure that each robot launches the navigation framework correctly.

The navigation process starts when the robot is launched, and the maps are visualized in RViz. The navigation framework is launched, and the cost maps for the robots are visualized here in RViz shown in Figure 6. Cost maps are visual representation used in robot navigation to show areas of the environment. The data received from the sensors such as LiDAR, navigation stack such as nav2 in ROS 2 generates cost maps. The greener, the safer for the robot to move, the redder the closer the robot to the obstacles resulting in higher cost or risk. Cost maps help the robot decide the safest and most efficient path to reach its target while avoiding obstacles. The grid shown in the centre in Figure 6, comes from the Grid display element in RViz. By default, in RViz grid is small and stay in the centre. We can visualize how far the robots are away from the centre as the grid is centred.

Figure 6. Cost maps in RViz showing robots location from the centre

Robots are autonomously exploring as well as searching for the target red cylinder as shown in Figure 7 while avoiding the obstacles. One robot appears black, while the others have grey or white colouring on top in Figure 7 due to GPU rendering effects in RViz. The robot models are affected by the camera angle, lighting, and obstruction by LiDAR rays. When the view is zoomed in, out or rotated, the appearance of the robots may change. This is purely a visual rendering issue, not a functional one. The blue rays are the laser emitted from the robots. The grey areas without the blue rays are the areas where the robots' LiDAR ray till now did not reach or may have reached earlier, the darker blue area is where the multiple LiDAR rays have been placed at similar time.

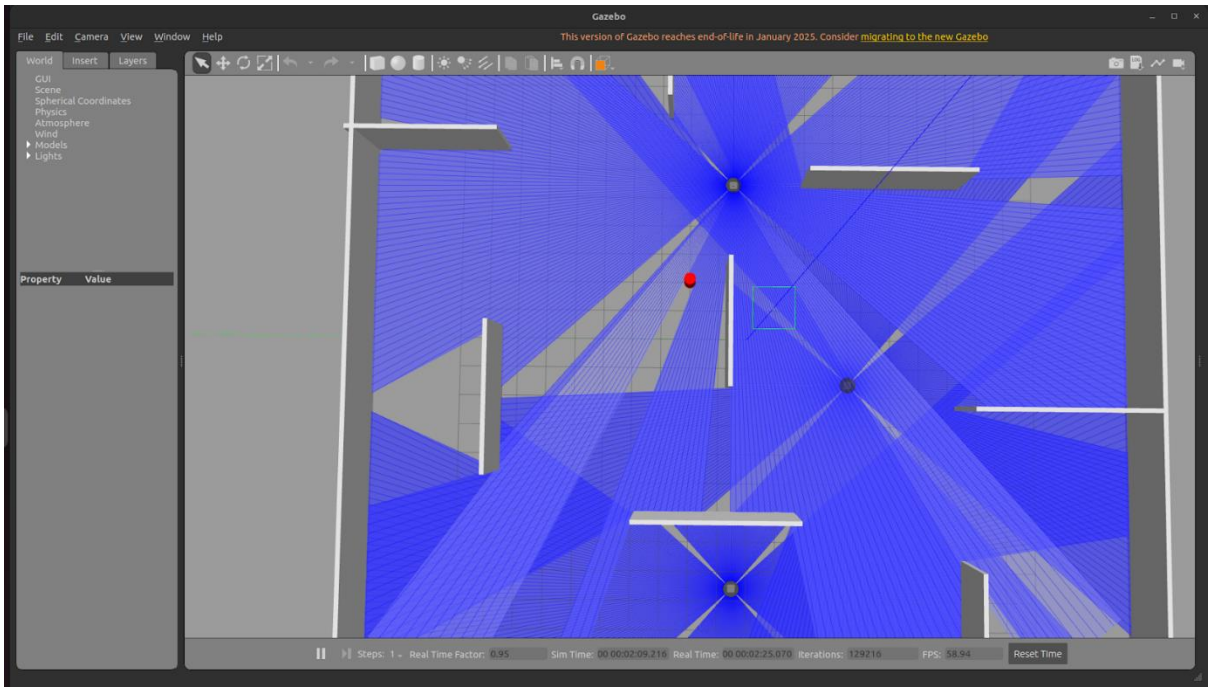

Figure 7. Robots in gazebo world exploring and searching for target

Each robot runs a navigation setup, generating an individual bringup_launch. This part of the launch file is responsible for including the main Navigation2 bringup launch file (bringup_launch.py) for each robot. It uses IncludeLaunchDescription to include an existing launch script located in the nav2_bringup package (specified by nav_launch_dir). The map_server node from the Nav2 package launches the Nav2 map server which serves a static map to the navigation stack. It loads the map from the specified YAML file (Maze1.yaml), which contains metadata about the map image and resolution. This node publishes the map data so that other nodes like localization and path planning can use it to navigate the environment. The remappings argument is used to adjust topic names as needed for multi-robot setups or namespace isolation. The map_server_lifecycle node runs the lifecycle manager for the map server node. Since Nav2 components use managed lifecycles, the lifecycle manager controls the state transitions such as configuring, activating of the map server node to ensure it is properly started and maintained. The parameters include use_sim_time for simulation time synchronization, autostart set to true for automatic activation, and a list of node names it manages, in this case, the map_server node. Together, these nodes, map_server and map_server_lifecycle, ensure the map server is properly loaded, activated, and integrated into the navigation stack for consistent and reliable navigation.

4.3 Exploration mode

After individual robots are launched, a centralized explorer node subscribes to all robot params and hence controls the robots based on their output of the camera frame. The functionality of this node is to manage 4 robots which are tb1, tb2, tb3, tb4 with individual velocity control, state machine with three distinct states such as exploring, target detected, waiting for resume, graceful handling of target detection/loss with timeout mechanism, integration, turtlebot3_drive for exploration and Nav2 for navigation, comprehensive logging and state monitoring.

During normal operation in the EXPLORING state, the node continuously monitors incoming velocity commands from the turtlebot3_drive system on each robot's /cmd_vel/dd topic. These commands are then forwarded to the robot's actual /cmd_vel topic, allowing the robots to move autonomously according to the exploration algorithm.

Robots are exploring the environment and the GUI feeding the camera view in real-time leveraging the rqt_image_view plugin from the rqt framework as shown in Figure 8. The GUI shows the Gazebo simulation environment with four robots and a custom-built interface for object detection status. There are 4 boxes, each displaying the camera feed from a robot's Intel RealSense200 camera. These feeds are visualized using OpenCV in Python. The integration is done by subscribing to the camera image topics from each robot and displaying them using a custom Python GUI. Each of the box includes live video feed from the robot, status which shows if the target is detected or not, detection which highlights the detected object, here the red cylinder, and save image which can capture and save the current frame. In the feed, dark grey walls and light grey floors reflects the gazebo environment. The top of each robot may appear grey or black depending on the rendering and camera angle which is a GPU rendering effect. The GUI allows toggling between showing and hiding images providing real-time visual feedback from all four robots during simulation.

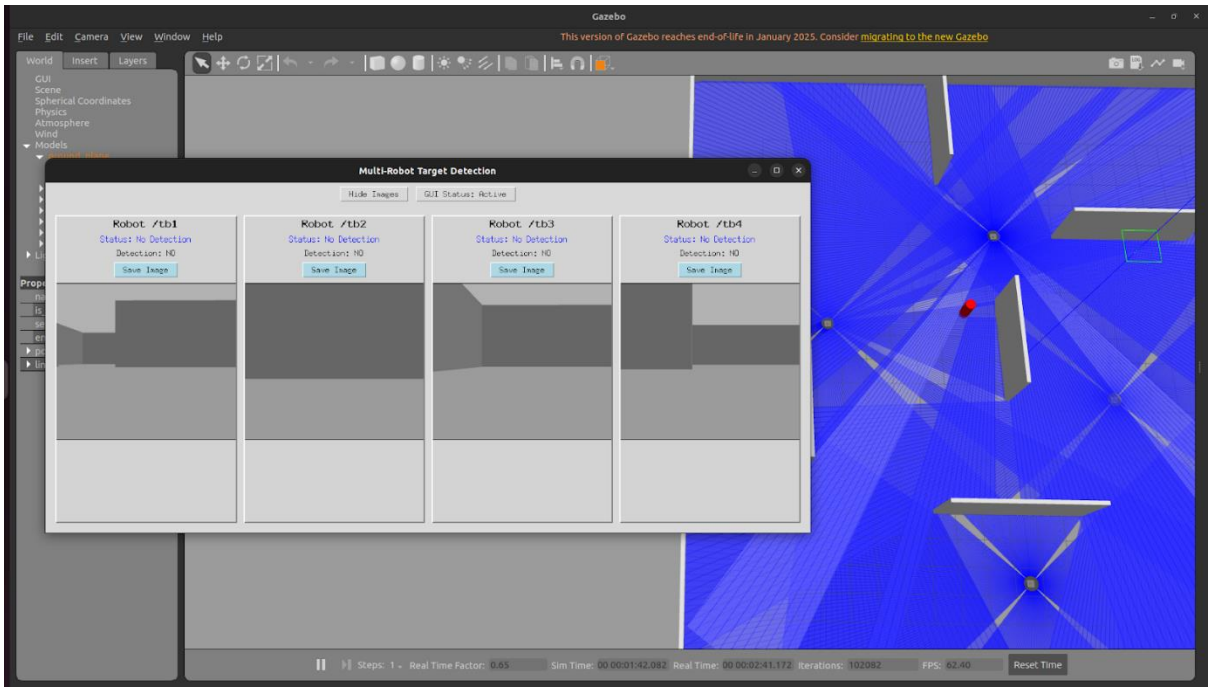

Figure 8. An example situation of target searching with live camera feed in AMCL

When a target is detected through the /global_cylinder_detections topic, the node immediately transitions to the TARGET_DETECTED state. In this state, the node performs several critical actions: first, it sends multiple stop commands (zero velocity Twist messages) to each robot's /cmd_vel topic in rapid succession to prevent overshooting; second, it begins blocking any incoming exploration commands from turtlebot3_drive, effectively giving full control of the robots' movement to the Nav2 navigation system. This ensures smooth transition from exploration to target approach behaviour. If the target is subsequently lost, the node enters a WAITING_FOR_RESUME state and starts a 3-second countdown timer. During this period, the node continues to block exploration commands while monitoring for potential target re-detection. If the target reappears within this window, the node cancels the timer and returns to the TARGET_DETECTED state. However, if the timer completes without re-detection, the node transitions back to the EXPLORING state, once again allowing velocity commands from turtlebot3_drive to reach the robots' /cmd_vel topics and resuming autonomous exploration.

Target is found by the robot /tb1 which is visible in the video feeding GUI shown in Figure 9. The detection status is seen in the GUI. There is green text in the first box displaying target found by: /tb1 as well as with pixel position of detection, depth value and live camera feed

shown in Figure 9. For the detection status each frame is refreshed about one second each frame so the detection status may momentarily show Yes or No even if the target is present.

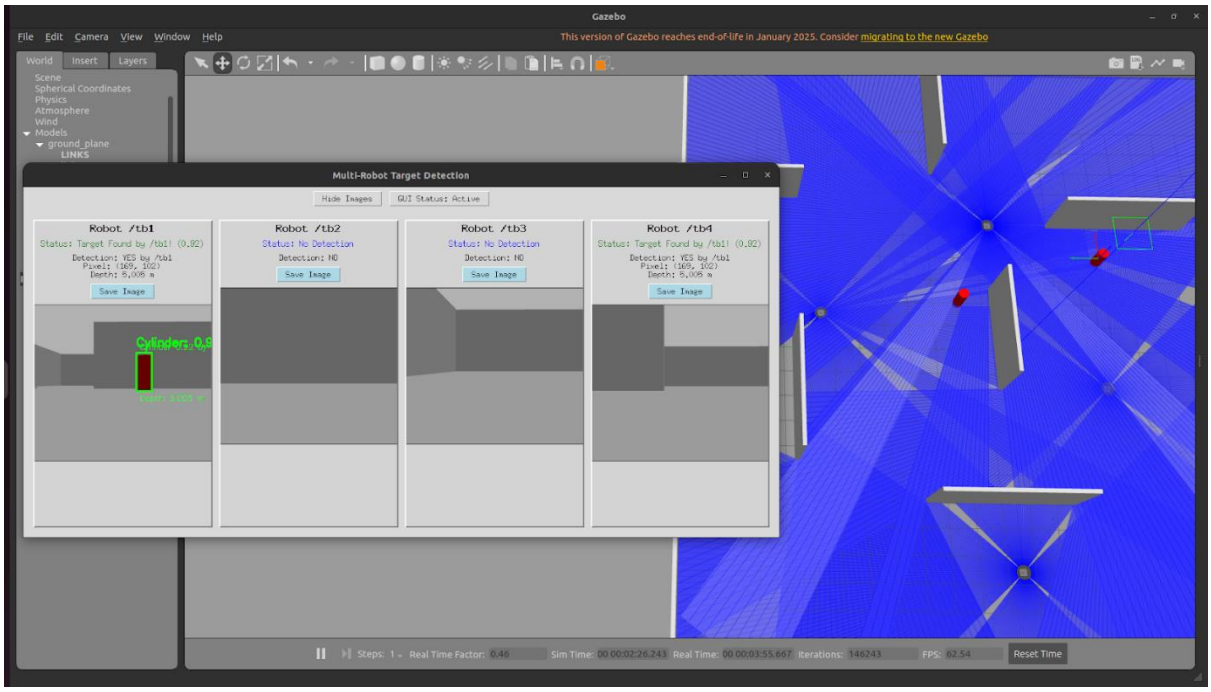

Figure 9. An example of target found in AMCL

There are two red cylinders placed in the gazebo environment for SLAM localization as shown in Figure 10. Each robot continues exploring until it detects a target. When one robot detects a target, it broadcasts this detection, so all other robots are updated via the shared system and GUI. To avoid confusion between the two targets each detection is handled separately. If the same target is detected by multiple robots, it's recognized as the same based on pixel position and depth and the system avoids duplicate detections. The robots continue exploring until both targets are found. Gazebo world with four robots exploring and searching for two targets, two red cylinders, is shown in Figure 10. This is an initial phase of exploration where the robots are actively exploring the environment, and no detection is made by any of the robot as shown in the GUI in the Figure 10. Similar system configuration has been implemented in those GUI just as AMCL localization.

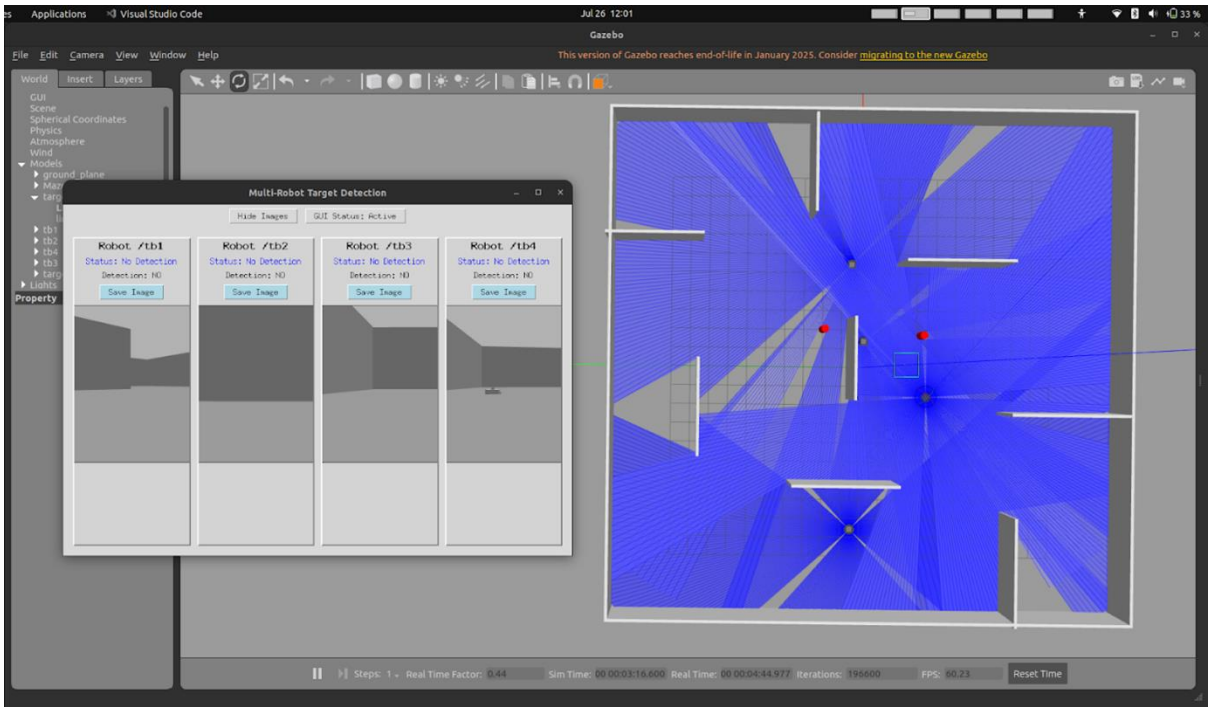

Figure 10. An example situation of target searching with live camera feed in SLAM

After the detection of the target, DepthToWorldConverter node processes detection data from the robot's sensors. It converts image coordinates into real-world positions. After the detection, need to accurately determine the exact position of detected objects in the real world relative to the robot and its environment. Raw camera or sensor data gives only relative positions, which aren't enough for precise navigation or interaction. By converting detection points from the camera frame through the robot's frame to global world coordinates, the problem of knowing where target truly is in the environment is solved. The node receives a detection from /tb1 TurtleBot robot shown in Figure 9 and calculates its depth and position in both camera and robot frames. Then applies the robot's pose and orientation to publish the target's absolute world coordinate.

4.4 Target detection and coordination

The system is designed to detect red cylindrical target objects in the environment by the help of YOLO model and coordinate the actions of multiple robots to address these targets efficiently. The system processes images from multiple robots, detects red cylinders, computes their 3D positions, and shares detection data across the other robots. Obstacles are detected using the laser from the LiDAR represent the world in the 2D plane point. The robot exploration is then done with the help of the LiDAR obstacles.

The obstacles are shown as red and then parallelly the same object in the world is shown in Figure 11. On the left side of the Figure 11 is the LiDAR view, showing how the robots perceive the environment through their sensors. On the right side is the corresponding Gazebo visual, which displays the actual 3D simulated environment with walls, floors, and robots. This side-by-side setup helps compare sensor data with the real environment. Red dots in the left side of the Figure 11 represents the laser scan points showing detected obstacles such as walls, robots, targets. Red lines are the laser rays extending from each robot's LiDAR sensor indicating what direction and range the laser is scanning. These red dots are auto generated in real-time by the LiDAR sensor and visualized through RViz as shown in the Figure 11 on the left side. The grid in left side in the RViz is a visual aid that helps to understand the scale, position, and orientation of objects in the simulation environment, and the black area indicates the base, background of the RViz. This dual view helps in verifying the red points from LiDAR match with the actual walls and obstacles shown in gazebo confirming proper sensor operation.

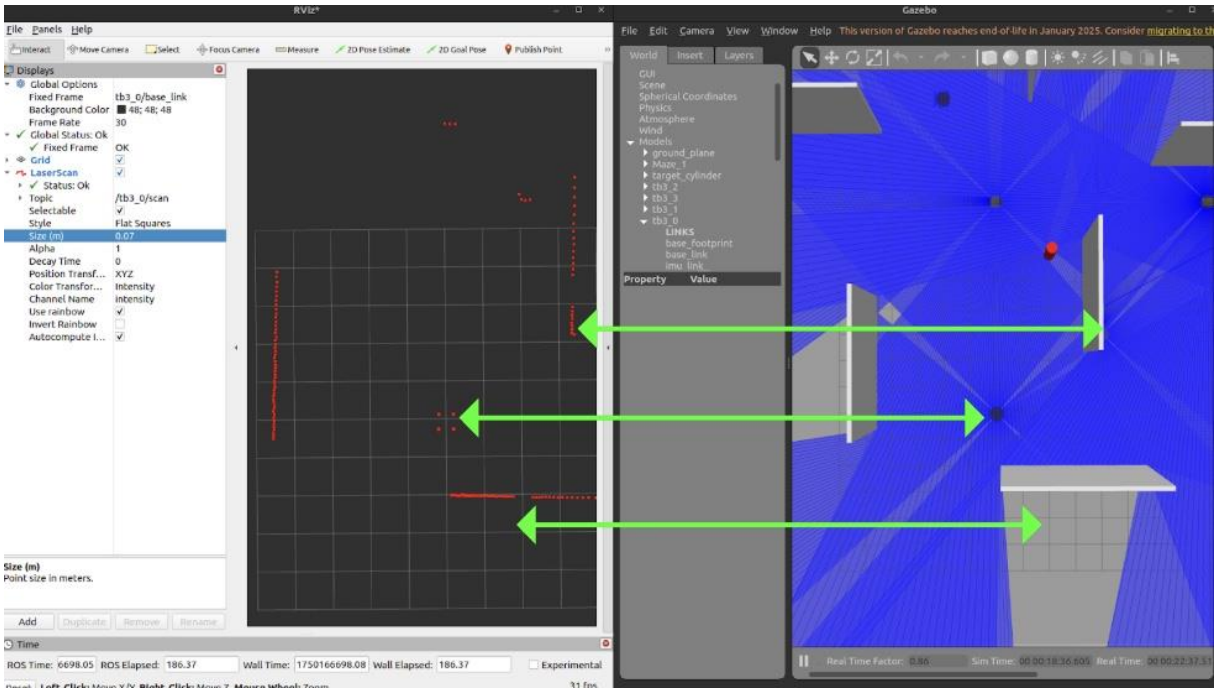

Figure 11. Obstacles detected by LiDAR visualized in RViz

The target is detected with the help of the camera (not with the LiDAR). From ultralytics, YOLO is imported where model = YOLO("model.pt"). YOLO() loads a pretrained model. A '.pt' file is a PyTorch checkpoint. When using YOLO (v8) from ultralytics, this .pt file contains neural network architecture and learned weights. When an image or frame from the

robot camera is passed to the model a preprocessing is done by resizing to fixed size like 640×640, normalize pixel values to [0, 1], and convert to a tensor with shape [1, 3, 640, 640]. The model processes the image frame through multiple layers and gives bounding box predictions to locate the target. In this way, the obstacles are overseen from the LiDAR whereas the camera handles the target detection.

Target Detection is done by the TargetDetector node. It performs key functions such as image acquisition through subscribes to RGB and depth images from multiple robots, object detection by using YOLOv8 to detect red cylindrical targets in images, 3D position estimation by computing the 3D position of detected cylinders using depth data, multi-robot coordination by Publishing detections globally for other robots to use, and GUI visualization which provides a real-time monitoring interface for all robots. Features of leveraging a pre-trained YOLOv8 model for accurate red cylinder detection which is the target in this search and rescue operation-based simulation is listed in Table 6.

Table 6. Features for target detection

Feature	Description
Multi-robot support	Handles 4 robots (/tb1, /tb2, /tb3, /tb4) simultaneously
YOLO detection	Uses a pre-trained YOLOv8 model(best.pt) for cylinder detection
Depth processing	Converts 2D detections into 3D coordinates using depth images
Global detection sharing	Publishes detections to/global_cylinder_detections
Interactive GUI	Displays detection status, images, and bounding boxes in real-time

Workflow of the target detection and notifying other robots via communication begins with loading the YOLOv8 model from the pre-trained best.pt file which is set for object detection tasks. Established essential publishers and subscribers in the ROS communication channel. Configuration of the topics image subscriber (/intel_realsense_r200_depth/image_raw), depth subscriber (/intel_realsense_r200_depth/depth/image_raw), camera info subscriber: /intel_realsense_r200_depth/depth/camera_info, detection publishers: /target_detected, /target_position, global publisher: /global_cylinder_detections has made. This subscribes to RGB images, depth images, and camera information from the RealSense R200 camera. Publishers broadcast detection outcome including detection status and object positions and a global topic for sharing detection information among multiple robots.

A GUI is also initialized using Tkinter which do live feed of detection events for monitoring. During exploration, RGB images received do a image callback function which processes these frames forward to the YOLOv8 model for object detection for rescue which is a red cylindrical object. Depth data is used by the system to calculate the 3D position of the target which is then published among other robots. The GUI updates in real time showcasing visual feedback at immediate level. For the coordinated multi-robot operation, each robot shares its detection result through the /global_cylinder_detection topic. All robots have access to each other detection data for collaborative decision making as they subscribed to this shared channel. Some of the key functions for multi-robot coordinated object detection are outlined in Table 7.

Table 7. Function overview for target detection

Feature	Description
__init__()	Initializes the ROS node, YOLO model, GUI, and ROS connections.
setup_gui()	Creates the Tkinter-based monitoring interface.
setup_ros_connections()	Configures ROS publishers and subscribers.
image_callback()	Processes incoming images and triggers detection.
depth_callback()	Stores depth images for 3D position calculation.
detect_with_yolo()	Runs YOLO inference and returns detection results.
update_gui()	Refreshes the GUI with the latest detection data.
global_detection_callback()	Handles detection messages received from other robots.

4.5 DepthToWorldConverter (World Coordinate Converter)

DepthToWorldConverter ROS 2 node is designed to convert detected red cylinder coordinates from camera pixel space to world coordinates. The primary need is to get the real-world coordinates of the target. However, object detection only gives the target's location in pixel coordinates relative to the camera frame. These pixel values alone are not useful for navigation or interaction. To solve this, used the DepthToWorldConverter node to convert the pixel coordinates into actual world coordinates, allowing robots to understand and act on the target's position in a shared physical space. It is part of a multi-robot coordination system where different robots detect objects and need to share their positions in a common world frame.

Key functions of the node include subscribe to cylinder detections from any robot in the system, dynamically subscribes to depth images, camera info, and odometry of the detecting

robot, converts pixel coordinates and depth into camera frame coordinates, publishes final world coordinates for navigation purposes. Pixel coordinates to camera frame transformation are done by converting a 2D image point to 3D point in the camera's coordinate system. Pixel location (u, v), corresponding depth value z, and the intrinsic parameters of the camera is used for this process. Camera intrinsics include focal lengths (fx, fy), and the principal point coordinate (cx, cy). Utilizing these parameters, the system calculates the real-world position of the point relative to the camera making accurate detected object's 3D localization.

From camera to world coordinate transformation explains how a robot system converts what its camera sees into real-world locations. The process takes a point that appears on the camera screen and figures out exactly where that point exists in the physical world around the robot. To complete the process number of steps been implemented. The first step is from camera screen to camera space conversion. Converted a point on the camera's flat screen into a 3D position relative to the camera itself. When something is visible on a camera image, its position is known as measured in pixels and how far away it is known by the depth. The system uses the camera's settings to calculate where this point actually sits in 3D space in front of the camera. The camera has specific properties that help with this conversion. The focal length tells how much the camera zooms in or out. The image centre tells where the camera is pointing. Using these camera properties along with the pixel location and depth, can find the real 3D position relative to the camera.

Next step camera space to robot space accounts for how the camera is mounted on the robot. Cameras and robots think about directions differently. The camera might consider forward as the direction it's pointing, right as moving across the image, and down as moving down the image. But the robot has its own ideas about forward, left, and up based on how it moves. This step rotates and flips the coordinates to match how the robot thinks about space. For example, what the camera sees as forward becomes what the robot considers forward. What the camera sees as right might become what the robot considers left. This ensures both the camera and robot agree on directions.

For robot space to world space step, places the point in the global world coordinate system. The robot knows its own position and orientation in the world. However, the point found is still only relative to where the robot is standing and which way it's facing. This step rotates the point based on the robot's current orientation in the world. If the robot is turned sideways,

the point needs to be rotated accordingly. Then it adds the robot's current world position to find where the point actually sits in the global coordinate system.

Next robot orientation is converted. The robot's orientation is stored as something called a quaternion, which is a mathematical way to represent rotation in 3D space. However, for this calculation, robot's yaw angle is needed, which is simply how much the robot has rotated around its vertical axis, like spinning in place. The system converts the quaternion into three rotation angles, roll (tilting side to side), pitch (tilting forward and backward), and yaw (spinning left and right). Only the yaw angle is used for the world coordinate conversion.

The entire process can be thought of as a chain of transformations. First, we transform from camera coordinates to robot coordinates. Then we rotate based on the robot's orientation in the world. Finally, we add the robot's position to get the final world coordinates. This system allows the robot to see something with its camera and immediately know exactly where that object is located in the real world. This is essential for the robot to navigate toward objects, avoid obstacles, or manipulate items it sees through its camera.

In AMCL localization [56], closest robot node helps multiple robots work together to reach a common target efficiently. It listens to the current positions of all robots and the location of the goal. For each robot, it calculates a path to the target using the A* algorithm, which finds the shortest route by exploring possible paths on a map and choosing the best steps based on distance and cost. After computing all paths, the node compares their lengths to find which robot has the shortest distance to the goal. It then sends a command to that closest robot to move towards the target while the others stay put. Additionally, it shares information about the distances to help monitor the system’s performance. Using the A* algorithm ensures that each robot’s path is optimized, making the coordination faster and more effective.

With the pose acquisition each robot sends or shares its current position and orientation with the central system. Here pose refers to the robot’s position and orientation, and acquisition means collecting which collectively means getting the robot’s current position and orientation. Once the target, here the red cylinder, is identified by any of the robot, its pixel location is converted into real-world coordinates using the DepthToWorldConverter node. This transformation utilizing camera depth, robot position, and orientation completes accurate localization. The resulting coordinates are then published on the /target_world_pose topic. All robots in the system subscribe to this topic to receive the target position.

For path computation for each robot, in both simulation and real-world scenarios, each robot computes a path to navigate from its current position to a target location. In the SLAM localization, two red cylinders are placed as targets to simulate detection tasks. The robots do not confuse the two targets, as each detection is processed with its pixel coordinates and depth value. Once a robot detects a target, it shares this information with the others, and the system ensures that each target is detected once by any one robot. The robots continue to explore until both targets are detected.

In the AMCL localization, the robot uses its current position estimate from the AMCL system as the starting point. Then, it defines the goal pose based on the target's coordinates, which could be the location of an object detected by its sensors. Using this start and goal information, the robot calls a path-planning service known as the ComputePathToPose action server that calculates the best possible path to reach the target while avoiding obstacles. After the path is generated, in AMCL localization, the robot calculates the total length of the path to understand the distance it needs to travel. Next, the system selects the robot with the shortest valid path to the target and commands that robot to navigate there using the NavigateToPose action.

5 Result

Four TurtleBot3 robots autonomously explored in the maze-like gazebo environment aligning with real world search and rescue environment. While AMCL mapping utilizes pre-loaded map, in the same environment SLAM based map utilizes and builds maps dynamically. To evaluate the performance topics of the simulation, communication efficiency, situational awareness of the robots, target detection, task distribution and decision making among the robots have been evaluated and discussed in this section.

5.1 Topic behaviour under SLAM and AMCL

The behaviour of ROS 2 topics reflects the underlying performance of localization, perception, coordination mechanism among the robots. The system's responsiveness, stability, and consistency in SLAM and AMCL localization are checked by monitoring some of the key topics developed in the ROS 2 system.

Topics observed for both SLAM and AMCL are summarized in Table 8. In SLAM as the map generates while exploring leads to minor delays, odometry drift, and irregular velocity commands whereas AMCL starts with preloaded map resulting in stable scan data, accurate odometry, and smooth command velocity. Object detection delay was closed in both cases however in SLAM localization higher false positive rates were observed.

Table 8. Observation of topics under SLAM and AMCL localization

Output type	SLAM observation	AMCL observation
/scan	Dynamic updates, minor delay in unknown zones	Stable and complete from beginning
/odom	Minor early-stage drift	Accurate pose feedback
/cmd_vel	Irregular in initial frame	Smooth from initiation
/target_info	Triggered after detection (~350ms delay)	Same latency, more deterministic
/global_cylinder_detections	Higher false positive rate early	Fewer compared to false positive
/claimed_targets_info	Conflicts under poor network conditions	Timestamp filtering handled well
/tb3_X/map	Evolving with each scan	Preloaded from /map_server
/initialpose	Not applicable	Mandatory for stable start

Stable AMCL map from /map_server shown in Figure 12. The map is with fully defined static layout with solid walls from the first frame. The grid shown in the centre comes from the grid display element in RViz. It is a static reference grid used to visualize scale and position. By default, in RViz, grid is small and stay in the centre. Its size can be increased. We can visualize how far the robots are away from the centre as the grid is centred.

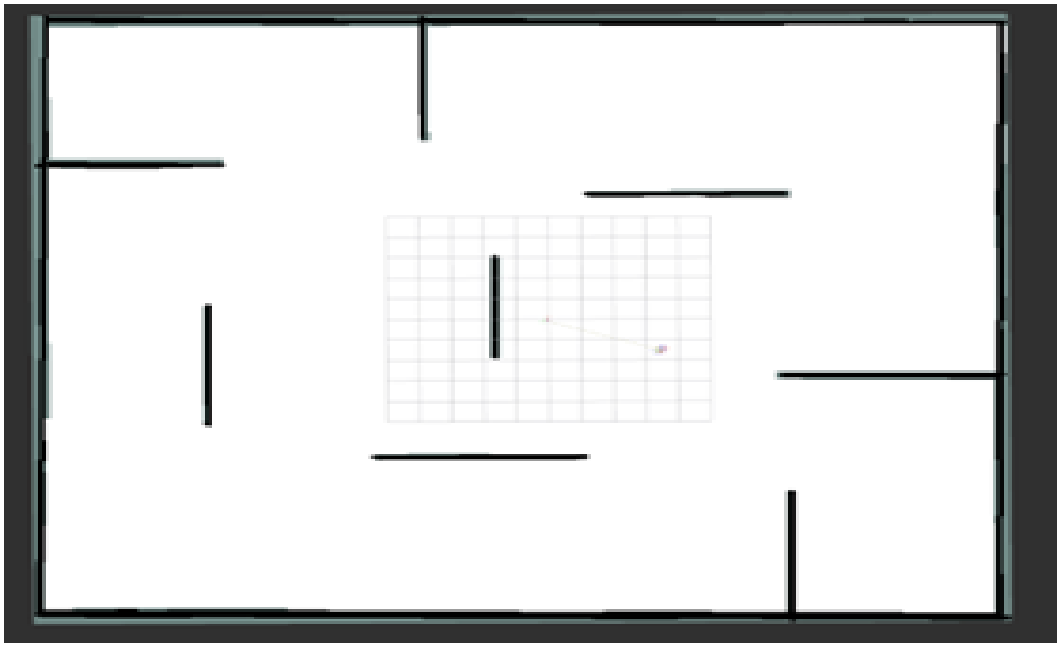

Figure 12. AMCL map preloaded from map server

Scattered initial map generated SLAM while robots exploring the environment shown in Figure 13. Incomplete boundaries and light thick walls are visible due to scan limitations. The green portions in the Figure 13 represents unknown or unexplored space whereas white portions represent free or open navigable space. The black portions are walls or obstacles detected by LiDAR. The green area is heavier where the robot has not scanned. Scattered dots near the top-left, bottom-left and other corners show incomplete scans. Two dots near the middle wall and one dot at the right side in Figure 13 are unresolved scan points or object detected by LiDAR but not fully mapped yet. Grids in the middle come from the static RViz grid which can be increased and can be utilized to visualize how far the robots or obstacles are from the centre.

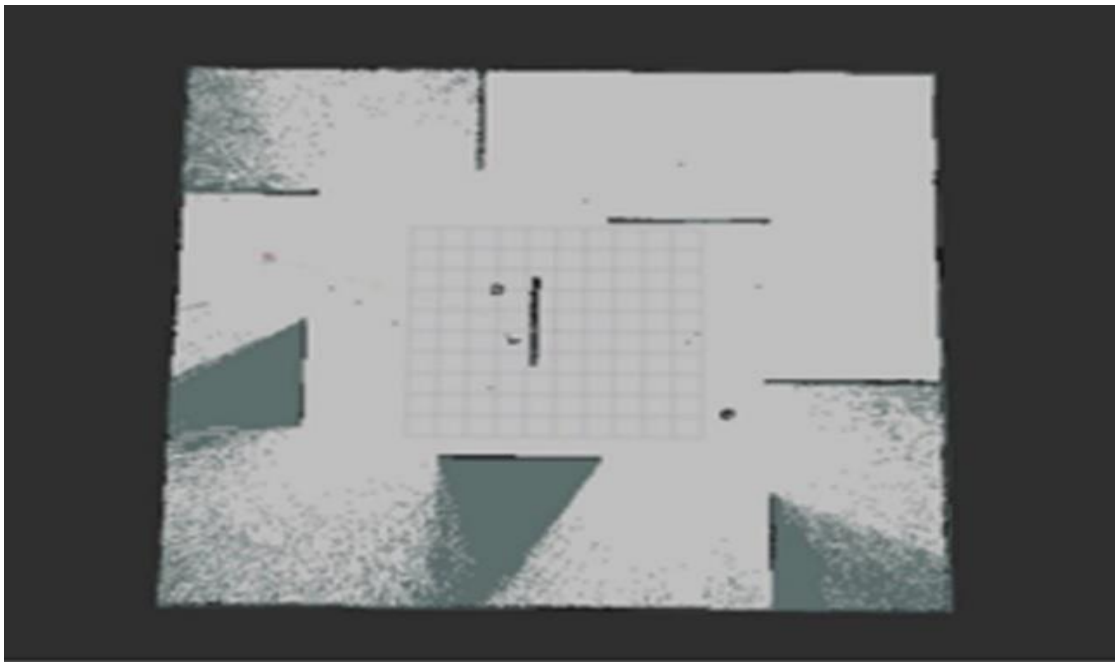

Figure 13. Initial SLAM map output generated by LiDAR scans

So, with AMCL, in known environment utilization can be peaked whereas SLAM is more applicable while the environment is unknow and reliable mapping is needed while exploring the environment.

5.2 Communication efficiency

For communication ROS 2 DDS (Data Distribution Service) is implemented. Communication efficiency differs slightly as well between AMCL and SLAM localization. Utilizing the ROS 2 DDS offers a decentralized, peer-to-peer messaging system which is paramount for such MRS applications. Edge computing was leveraged by doing computation such as object detection, navigation, mapping on the robots itself allowing parallel communication between robots without overloading the network which reduces delay and dependency on external systems.

SLAM updates its map every 2~3 seconds due to continuous mapping as noted from the simulation and shown in Table 9 whereas using the static map made AMCL no runtime updates. It is also observed that SLAM shows higher network delays and occasional message dropout whereas AMCL maintained low latency of <0.05 seconds showcasing more stable network performance.

Table 9. ROS 2 DDS message

Metric	SLAM (avg.)	AMCL (avg.)
Message frequency	10 Hz	10 Hz
Map update rate	2~3 Hz	Static
/performance_metrics publishing	1 Hz	1 Hz
Delay observed (WiFi)	~0.24 s	<0.05 s
Dropout simulated	1 every 25s	Raely occurs

5.3 Situational awareness and obstacle handling

Situational awareness plays a critical role in multi-robot operations specially in the search and rescue operation. In such dynamic environment real-time perception, localization, and decision making are essential. In the conducted simulation for representing such situational awareness maze like environment is adopted.

Under the two localization methods, SLAM and AMCL, robots' behaviour varied significantly. SLAM based localization does not have any preloaded map. The map is generated while the robots explore the environment utilizing the LiDAR sensor. This way of approach, makes the robots occasionally showcasing misalignment when they encounter obstacles. This resulted in navigation inaccuracy, increased in recovery behaviours which is in average three recovery events per run. These recovery actions generally occurs when unexpected collisions or failed path planning attempts caused by incomplete obstacle data in the map. On the other hand, AMCL based simulation runs on preloaded static map with well defined costmap which gave it an edge in obstacle avoidance. Required fewer than one recovery event on average as they recognise and navigate around obstacles with minimal deviation. AMCL based localization proven to be effective and safe for robot movement due to the static nature of the costmap making robots stable and predictable obstacle handling.

Based on the output maps in the SLAM and AMCL localization, situational awareness for the robots also differs. In SLAM based localization, the initial generated map after almost two minutes of runtime partially defined boundaries with incomplete walls due to limited sensor coverage and ongoing loop closure corrections. This incomplete mapping influenced both path planning and obstacle avoidance during the initial stage of the search and rescue operation of the simulation. In contrast, AMCL map is directly sourced from the /map_server which has fully defined and stable environment from the beginning. This static map ensured performing accurate costmap generate through the navigation system and making it a reliable obstacle handling without uncertainties associated with active mapping.

When target is detected, it is broadcasted using the /target_info topic and path planning and motion control flow is initiated. Navigation stack computes a suitable motion plan resulting in generation of velocity commands on the /cmd_vel topic which directs the robots towards the target. Motion plans often required re-planning in SLAM localization making it changes of map data especially in the early exploration phase which led to irregular velocity outputs and cautious movement till the map stabilized. Meanwhile a more direct and stable path planning is observed utilizing the AMCL based mapping leveraging from a fixed environment representation.

5.4 Target detection evaluation

The simulations use YOLOv8 for object detection [53]. Synthetic RGB-D images from Gazebo's cameras are processed through PyTorch based detection model. Target detection under varying conditions can be observed from Table 10. Here the performance is evaluated based on response time and robustness under varying visibility conditions. Under clear view environment, system consistently achieved detection-to-move latency within 350 milliseconds resulting in the robots to react promptly when the target is identified. Such quick response is a must for maintaining real-time coordination and efficient task execution while working with dynamic multi-robot operations. On the other hand, detection latency increased almost close to double in the presence of occlusion making it to around 500 to 700 milliseconds along with a poor recovery highlighting the sensitivity to visual obstructions with YOLOv8 based detection pipeline. Duplicate detections also been addressed which may occur when robots simultaneously observe the same target. Duplicate detections cause a delay around 80 milliseconds which was managed through timestamp-based filtering mechanism which ensures duplicate detection messages did not occur any coordination errors or task assignment conflicts resulting in consistent coordination and minimizing false claims among the robots.

Table 10. Target detection performance

Condition	Detection to move latency	Occlusion recovery
Clear view	~350 ms	Immediate
Occlusion	~500 to 700 ms	Poor
Duplicate detections	~80 ms	Resolved by timestamp logic

These results showcase the system's strong performance in unobstructed environments which may be difficult to observe in real-world scenarios.

5.5 Task allocation and decision making

While working with multi-robot systems in dynamic environments effective task allocation and decision making is a must where agents need to coordinate autonomously. In the simulation and experiment, task for the robots to complete is decentralized and based on real-time perception and evaluation of each robot's suitability for the task.

Task allocation among the robots begins with target detection and broadcasted over the /global_cylinder_detections topic. When the robots receive the message of the target detection each robot examine their own position, and compute path from them to the target. Whichever robot is near the the target reaches the target and publish the message over /claimed_target_info topic. This decentralized approach makes sure that even with simultaneous detection by multiple robots' whichever robot contains the typical shortest valid path or earliest detection completes the task by reaching to the target for rescue. If no progress been reported within three seconds span, the claim is reevaluated and the task maybe assigned to another robot.

Task allocation between the robots differs noticeably between the SLAM and AMCL localization. Robots show more deterministic behaviour with AMCL mapping where task completion takes time between 150 to 220 seconds with a redundancy rate of just 0.7 per run which was made possible due to the initial localization provided by the AMCL resulting in for immediate and confident task claiming, reducing unnecessary reassignment and overlap. On the other hand, it takes around 300 to 400 seconds with SLAM mapping for the task to complete with a redundancy rate of more than 2 per run. As SLAM based mapping starts with uncertainties and delayed localization makes frequent reassignment and overlap especially while in early exploration phase.

Target claim conflicts also higher with SLAM which is 8% whereas with AMCL it is only 3% making AMCL more feasible for the structured know environment. Average response time from target detection to motion command execution for the system was almost same for both localization method around 400 to 600 milliseconds. Though both AMCL and SLAM based localization successfully managed the decentralized tasks, but AMCL shows consistently outperforming SLAM for the known environment making it efficient, and lower redundancy due to its reliable initial localization. However, in the case of real-world challenge where the surroundings can be very unpredictable SLAM based localization gets an edge due to its flexible and continuous learning concerning the environment.

The simulation conducted in align with search and rescue operation highlights the key performance concerning communication, perception, situational awareness, and task coordination. Using the ROS 2 DDS middleware [23], the system maintained a robust real-time topic exchange. In SLAM based operation average message delay was 0.24 seconds. Object detection average response latency was 380 milliseconds with up to two false positive per run detection due to early-stage map building whereas AMCL consistently maintained low false detection.

In the situational awareness, SLAM required 30 to 50 seconds for initial mapping resulting in some minor planning delays whereas AMCL localization utilizing the /initialpose topic localized within 1 to 2 seconds resulting in minimal localization error. Task allocation through timestamp-based decision making where timestamp records the time at which a specific evert such as target detection or a task completion occurs achieved 96% success rate in first claim decision with AMCL showing a lower task redundancy rate (0.7 per run) compared to SLAM (2.1 per run). These results show robustness of decentralized framework among multi-robot systems and AMCL efficiency in known environments whereas SALM's suitability is aligned in exploration in unmapped areas despite its higher initial overhead.

6 Conclusions

This thesis is conducted in a comprehensive manner which includes the design, development, and evaluation of a decentralized multi-robot coordination framework specifically for SAR operation addressing critical aspects of autonomy, perception, situational awareness, communication, object detection, and collaborative decision making. Performing reviews of several articles in these domains, the research recognized the growing significance of MRS in environments where scalability, adaptability, and fault tolerance are paramount. While several key advantages along with real-time data showcasing promising advantages of MRS in search and rescue operations, there are some limitations within the simulation itself as well as complex real-world scenarios which also outlines notable future works can be performed to enhance its reliability, robustness, and practicality.

SAR operations are needed in disastrous situations where convenient environment which is conducted in the simulation while having consistent communication, dynamic real-time image processing, obstacle identification and path planning which gives the best route are very difficult to obtain. A significant limitation of this study is the use of only ground based robots for simulation and testing considering that SAR operations may need to be conducted underwater as well or may need aerial drones for understanding the environment correctly. In real-world scenarios, collaboration between heterogenous robots especially in SAR operations UAVs and USVs is a must for obtaining more robust and edge performance. Though AMCL localization results shows promising due to its pre-loading feature of mapping but in the real-world scenario obtaining a well-known map before exploring the area is quite difficult to obtain which impacts the robots' movement, sensor drift, calibration errors resulting in adverse effect on performance metrics. Implementing attention-based transformers such as vision transformer, training detection networks on occlusion rich datasets, deploying models on edge AI hardware with optimized inference pipelines should be inspected for enhancing object detection in any complex scenario along with improving real-time performance. Using ROS 2 DDS as a communication middleware [23] makes the connectivity among the robots to near perfect condition which is hard to maintain in any disastrous area, directly affecting coordination efficiency and massage handling observed in simulation. Frameworks such as ACHORD, MOCHA should be adopted for future enhancement aligning adaptive communication strategies capable of operating under unreliable network conditions [15], [16]. Task allocation among the robots in this experiment followed decentralized task allocation

which is employed utilizing timestamp-based decision logic and path computation metrics. Such approach is built upon fixed logic. But state of the art architecture reinforcement learning (RL) and MARL shows promising potential for more adaptive and scalable decision-making [4], [49]. MARL allows multiple robots to learn cooperative task allocation strategies by interacting with dynamic environments and optimizing long-term rewards which is out of scope for timestamp-based architecture [4]. MADDPG and QMIX algorithms enable decentralized agents to resolve conflicts, adapt to changing task loads, and improve overall efficiency without relying on predefined rules [49].

This research addresses both the advantages and challenges identified in prior research such as reliable communication, decentralization of tasks among multiple robots, heterogenous robots' coordination, robust situational awareness while trying to integrate practical solutions within the simulation framework. The developed system designed to have modularity for independent navigation operation, exploration, object detection, and task coordination nodes which is critical for SAR operation. SLAM and AMCL based localization strategies are implemented and compared critically. While AMCL offers superior stability, task efficiency, and obstacle handling in a known environment, but SLAM remains essential even with initial map inaccuracies for unknown or dynamically evolving surroundings which often occurs in any disastrous situation where SAR operation will be employed. Object detection leveraging YOLOv8 integrated with real-time navigation and decentralized decision making though susceptibility to occlusions and environmental variability represented potential for autonomous target navigation which can be more enhanced with advanced perception models. Implementation of timestamp-filtered logic though successfully minimizes task conflicts and responsive coordination among the robots but utilizing MARL could significantly enhances the adaptability and learning capacity of such systems in dynamic mission scenarios. Looking forward, the research also identifies key areas of enhancement for a thorough, robust, and practical SAR operation such as integrating multimodal sensor fusion for improved perception, employment of edge computing for real-time inference, adopting robust communication protocols for network constrained environments, and exploring learning-based decentralized task allocation for adaptive mission execution. Conclusively this study establishes a foundational approach for scalable, intelligent, and cooperative MRS which contributes to the broader goal of developing resilient robotic teams capable of operating autonomously in mission-critical environments where human access is limited or hazardous.

References

[1] Yi Dong, Zhongguo Li, Xingyu Zhao, Zhengtao Ding, Xiaowei Huang (2023). Decentralised and Cooperative Control of Multi-Robot Systems through Distributed Optimisation.

[2] Barton Research Group, “Multi-Robot Systems,” University of Michigan. Accessed Mar. 2025. [Online]. Available: https://brg.engin.umich.edu/research/multi-robot-systems/

[3] Gielis, J., Shankar, A., & Prorok, A. (2022). A Critical Review of Communications in Multi-Robot Systems. Department of Computer Science and Technology, University of Cambridge, United Kingdom.

[4] Wang, Y., Damani, M., Wang, P., Cao, Y., & Sartoretti, G. (2022). Distributed Reinforcement Learning for Robot Teams: A Review.

[5] Tsiotras, P., Pomares, J., Felicetti, L., & Varagnolo, D. (2023). Editorial: Multi-robot systems for space applications. Frontiers in Robotics and AI, 10, 1253381. doi: 10.3389/frobt.2023.1253381

[6] Wang, L., & Liu, G. (2024). Research on multi-robot collaborative operation in logistics and warehousing using A3C optimized Yolov5-ppo model. Frontiers in Neurorobotics, 17. https://doi.org/10.3389/fnbot.2023.1329589

[7] Smart Laboratory, “MRS Research,” brg.engin.umich.edu/research/multi-robot-systems/. Accessed Mar. 2025. [Online]. Available: https://brg.engin.umich.edu/research/multi-robot-systems/

[8] Yang, Z., & Tron, R. (2024). Enhancing Security in Multi-Robot Systems through Co-Observation Planning, Reachability Analysis, and Network Flow.

[9] Ichnowski, J., Chen, K., Dharmarajan, K., Adebola, S., Danielczuk, M., Mayoral-Vilches, V., Jha, N., Zhan, H., Llontop, E., Xu, D., Buscaron, C., Kubiatowicz, J., Stoica, I., Gonzalez, J., Goldberg, K. (2023). FogROS2: An Adaptive Platform for Cloud and Fog Robotics Using ROS 2.

[10] G. A. Novotny, M. W. Maier, M. Pfanne, and M. Neupert, “A Mobile Robot Platform for Search and Rescue Applications,” Proceedings of the 30th DAAAM International Symposium, 2019, pp. 0622–0627, doi: 10.2507/30th.daaam.proceedings.086.

[11] Cachia, A., Huda, M. N., Liu, P., Saha, C., Tickle, A., Arvanitakis, J., & Aziz, S. M. (2019). Development and evaluation of a novel robotic system for search and rescue. In Lecture notes in computer science (pp. 370–382). https://doi.org/10.1007/978-3-030-25332-5_32

[12] D. S. Drew, "Multi-Agent Systems for Search and Rescue Applications," Current Robotics Reports, vol. 2, no. 3, pp. 145–155, 2021, doi: 10.1007/s43154-021-00038-5.

[13] M. Lyu, Y. Zhao, C. Huang, and H. Huang, "Unmanned Aerial Vehicles for Search and Rescue: A Survey," Remote Sensing, vol. 15, no. 11, 2023, Art. no. 3266, doi: 10.3390/rs15113266.

[14] Arabboev, M., Begmatov, S., Nosirov, K., Shakhobiddinov, A., Chedjou, J. C., & Kyamakya, K. (2021). Development of a Prototype of a Search and Rescue Robot Equipped with Multiple Cameras. 2021 International Conference on Information Science and Communications Technologies (ICISCT), 1–5. https://doi.org/10.1109/icisct52966.2021.9670087

[15] M. Saboia, L. Clark, V. Thangavelu, S. Mehta, C. Xia, B. M. Sutherland, R. T. Vaughan, and M. Schwager, "ACHORD: Communication-Aware Multi-Robot Coordination with Intermittent Connectivity," IEEE Robotics and Automation Letters, vol. 7, no. 4, pp. 9202–9209, Oct. 2022, doi: 10.1109/LRA.2022.3193915.

[16] F. Cladera, Z. Ravichandran, I. D. Miller, M. A. Hsieh, C. J. Taylor, and V. Kumar, "Enabling Large-Scale Heterogeneous Collaboration with Opportunistic Communications (MOCHA)," arXiv preprint, arXiv:2309.15975, Sep. 2023.

[17] Á. Serra-Gómez, B. Brito, H. Zhu, J. J. Chung, and J. Alonso-Mora, "With Whom to Communicate: Learning Efficient Communication for Multi-Robot Collision Avoidance," arXiv preprint, arXiv:2009.12106, Sep. 2020.

[18] S. Bharadwaj, K. Gonabattula, S. Saha, C. Sarkar, and R. Raja, "Concurrent Transmission for Multi-Robot Coordination," arXiv preprint, arXiv:2112.00273, Dec. 2021.

[19] V. Belle, T. Bolander, A. Herzig, and B. Nebel, "Epistemic Planning: Perspectives on the Special Issue," Artificial Intelligence, vol. 315, Art. no. 103841, May 2023, doi: 10.1016/j.artint.2023.103841.

[20] Wang, Y., Damani, M., Wang, P., Cao, Y., & Sartoretti, G. (2022, April 7). Distributed Reinforcement Learning for Robot Teams: A review. arXiv.org. https://arxiv.org/abs/2204.03516

[21] Rovira-Más, F., Chen, M., & Rajkumar, R. (2023). FogROS2: Cloud Robotics for the Edge Era. Journal of Field Robotics.

[22] Robotis, "TurtleBot3 Overview," Robotis e-Manual, accessed June 2025. [Online]. Available: https://emanual.robotis.com/docs/en/platform/turtlebot3/overview/

[23] ROS.org, "About Different Middleware Vendors," ROS Foxy Documentation, accessed June 2025. [Online]. Available: https://docs.ros.org/en/foxy/Concepts/About-Different-Middleware-Vendors.html

[24] F. Jalled and I. Voronkov, "Object Detection Using Image Processing," arXiv preprint arXiv:1611.07791, Nov. 2016. [Online]. Available: https://arxiv.org/abs/1611.07791

[25] Vashisht, M., & Kumar, B. (2020). A survey paper on Object Detection Methods in Image Processing. 2020 International Conference on Computer Science, Engineering and Applications (ICCSEA), 1–4. https://doi.org/10.1109/iccsea49143.2020.9132871

[26] S. Vaddi, A. Jannesari, and C. Kumar, "Efficient Object Detection Model for Real-Time UAV Applications," arXiv preprint arXiv:1906.00786, 2019. [Online]. Available: https://arxiv.org/abs/1906.00786

[27] Yang, Mingji, et al. "Hybrid-DETR: A Differentiated Module-Based Model for Object Detection in Remote Sensing Images." Electronics, vol. 13, no. 24, 20 Dec. 2024, pp. 5014–5014, https://doi.org/10.3390/electronics13245014.

[28] Fei, Lunlin, and Bing Han. "Multi-Object Multi-Camera Tracking Based on Deep Learning for Intelligent Transportation: A Review." Sensors, vol. 23, no. 8, 10 Apr. 2023, pp. 3852–3852, https://doi.org/10.3390/s23083852.

[29] Y. Wang, X. Liu, M. Chen, and L. Zhang, "Robot-Track: A Multi-Sensor Interface for Real-Time Situational Awareness," in Proc. IEEE Int. Conf. on Robotics and Automation in Industry (ICRAI), 2024, pp. 114–120, doi: 10.1109/ICRAI62391.2024.10894257.

[30] T. Pääkkönen and J. Pakkala, "Edge-Based Human-Robot Interaction using YOLOv5 and MediaPipe," IEEE Access, vol. 11, pp. 90234–90245, 2024.

[31] M. Sarker, R. Gomez, and J. K. Tan, "CHARMIE: A Dataset for Context-Aware Human-Robot Interaction," in Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS), 2023, pp. 670–675.

[32] D. Amodei, J. Clark, and S. McCandlish, "Learning to Ask Questions for Clarifying Ambiguity in Human-Robot Interaction," arXiv preprint arXiv:2301.00001, 2023.

[33] D. González-Jorge, C. Riveiro, and S. Lagüela, "3D SLAM Mapping with Rotating LiDAR on Mobile Robots," IEEE Sensors Journal, vol. 23, no. 5, pp. 3412–3420, 2023.

[34] A. Faria, M. Reis, and H. Gonçalves, "Semantic Mapping for Mobile Robots using YOLOv5 and LiDAR Fusion," Robotics and Autonomous Systems, vol. 159, 2023.

[35] A. El-Sayed, N. El-Bendary, and S. Mahmoud, "Activity Recognition using Sensor Fusion and LSTM in Robotics," IEEE Sensors Journal, vol. 24, no. 2, pp. 880–888, 2023.

[36] M. Koutsoukou-Argyraki and A. Aspragathos, "Predicting Human Motion Using Wearable Sensors and Gaussian Mixture Models," IEEE Trans. on Haptics, vol. 16, no. 1, pp. 34–42, 2023.

[37] S. Kuthirummal, T. Varghese, and A. Thomas, "Augmented Reality Interfaces for Industrial Human-Robot Collaboration," in Proc. IEEE Int. Symp. on Mixed and Augmented Reality (ISMAR), 2023.

[38] C. Chandler, R. Holladay, and A. Thomaz, "Modeling Human Beliefs in Robots using Theory of Mind," ACM Trans. on Human-Robot Interaction, vol. 13, no. 4, pp. 1–24, 2024.

[39] B. Thanikkal, K. Raman, and S. Nair, "Sensor Fusion and AI for Maritime Situational Awareness," in Proc. IEEE OCEANS Conf., 2024.

[40] S. Khan, T. Raza, and M. Abbas, "A Review on Situational Awareness in Disaster Robotics," J. Field Robotics, vol. 40, no. 1, pp. 15–28, 2023.

[41] Abderrahmane Tamali, et al. “Distributed and Autonomous Multi-Robot for Task Allocation and Collaboration Using a Greedy Algorithm and Robot Operating System Platform.” IAES International Journal of Robotics and Automation (IJRA), vol. 13, no. 2, 12 May 2024, pp. 205–205, https://doi.org/10.11591/ijra.v13i2.pp205-219. Accessed 4 June 2025.

[42] Testa, Andrea, et al. “A Tutorial on Distributed Optimization for Cooperative Robotics: From Setups and Algorithms to Toolboxes and Research Directions.” ArXiv.org, 2023, arxiv.org/abs/2309.04257. Accessed 28 July 2025.

[43] Orr, James, and Ayan Dutta. “Multi-Agent Deep Reinforcement Learning for Multi-Robot Applications: A Survey.” Sensors, vol. 23, no. 7, 30 Mar. 2023, p. 3625, https://doi.org/10.3390/s23073625.

[44] Dinelli, Chris, et al. “Configurations and Applications of Multi-Agent Hybrid Drone/Unmanned Ground Vehicle for Underground Environments: A Review.” Drones, vol. 7, no. 2, 1 Feb. 2023, p. 136, www.mdpi.com/2504-446X/7/2/136, https://doi.org/10.3390/drones7020136.

[45] X. An, C. Wu, Y. Lin, M. Lin, T. Yoshinaga and Y. Ji, "Multi-Robot Systems and Cooperative Object Transport: Communications, Platforms, and Challenges," in IEEE Open Journal of the Computer Society, vol. 4, pp. 23-36, 2023, doi: 10.1109/OJCS.2023.3238324.

[46] Heřt, Daniel, et al. “MRS Drone: A Modular Platform for Real-World Deployment of Aerial Multi-Robot Systems.” Journal of Intelligent and Robotic Systems, vol. 108, no. 4, 17 July 2023, https://doi.org/10.1007/s10846-023-01879-2. Accessed 12 Oct. 2023.

[47] Hossain, M. A., Mahmud, S. A., Rahman, M. M., & Bera, R. (2023). Internet of Robotic Things for Mobile Robots: Concepts, Technologies, Challenges, Applications, and Future Directions. Internet of Things, 21, 100716.

[48] Chowdhury, A., Pal, A., & Natraj, A. (2023). From Simulations to Reality: Enhancing Multi-Robot Exploration for Urban Search and Rescue. arXiv preprint, arXiv:2311.16958.

[49] Liang, M., Jain, D., Lusk, P., Su, J., Luong, K., Yu, T., Liu, Y., Nair, A., & Finn, C. (2025). EmbodiedAgent: A Scalable Hierarchical Approach to Overcome Practical Challenges in Multi-Robot Control. arXiv preprint, arXiv:2504.10030.

[50] Hamid, H., Idrees, A., Mahmood, A., Zaidi, S. A. R., & Zikria, Y. (2021). Optimization Techniques for Multi-Robot Task Allocation Problems: Review on the State of the Art. International Journal of Advanced Computer Science and Applications (IJACSA), 12(9), 329–336.

[51] Gazebo, “Install Ubuntu Source (Gazebo Fortress),” Gazebosim.org, [Online]. Available: https://gazebosim.org/docs/fortress/install_ubuntu_src/. Accessed: Jul. 2025.

[52] Open Source Robotics Foundation, “Nav2 Configuration Reference,” Nav2 Documentation. [Online]. Available: https://docs.nav2.org/configuration/index.html. Accessed Jul. 2025.

[53] YOLOv8: State-of-the-Art Object Detection Architecture,” YOLOv8 Documentation, [Online]. https://yolov8.com/.Accessed: Jul. 2025

[54] Open Source Computer Vision Foundation, “OpenCV: Open Source Computer Vision Library,” OpenCV.org. [Online]. Available: https://opencv.org/. Accessed: Jul. 2025.

[55] PyTorch Team, “PyTorch: Deep Learning Framework,” PyTorch.org. [Online]. Available: https://pytorch.org/. Accessed: Jul. 2025.

[56] OSRF, “Adaptive Monte Carlo Localization (AMCL),” ROS Wiki. [Online]. Available: http://wiki.ros.org/amcl. Accessed: Jul. 2025.

[57] Steve Macenski, “Slam Toolbox 2.6.10 Documentation,” ROS 2 Humble API Documentation. [Online]. Available: https://docs.ros.org/en/ros2_packages/humble/api/slam_toolbox/. Accessed: Jul. 2025.

[58] Canonical Ltd., “Ubuntu 22.04 LTS (Jammy Jellyfish) Release Information,” Ubuntu Releases, [Online]. Available: https://releases.ubuntu.com/jammy/. Accessed: May 2025.

[59] Open Robotics, “RViz,” ROS Wiki. [Online]. Available: http://wiki.ros.org/rviz. Accessed: Jul. 2025.

[60] Python Software Foundation, “Python 3.11.0 Release – Oct. 24, 2022,” Python.org. [Online]. Available: https://www.python.org/downloads/release/python-3110/. Accessed: Jul. 2025.

[61] Open Robotics Foundation, “TurtleBot3 in Gazebo Simulation,” ROS Wiki. [Online]. Available: http://wiki.ros.org/turtlebot3_gazebo. Accessed: Jul. 2025.

[62] Open Robotics, “Nav2 Bringup Package for ROS 2 Humble,” index.ros.org. [Online]. Available: https://index.ros.org/p/nav2_bringup/. Accessed: Jul. 2025.

[63] Open Robotics Foundation, “rqt (Qt-based Framework for ROS),” ROS Wiki. [Online]. Available: http://wiki.ros.org/rqt. Accessed: Jul. 2025.

[64] Open Robotics Foundation, “RViz2 API Reference (ROS Rolling),” ROS 2 Documentation. [Online]. Available: https://docs.ros.org/en/rolling/p/rviz2/. Accessed: Jul. 2025.

[65] Open Robotics, “tf2_ros (Coordinate Transforms in ROS),” ROS Wiki. [Online]. Available: http://wiki.ros.org/tf2_ros. Accessed: Jul. 2025.

[66] Python Software Foundation, “Pillow: The Friendly PIL Fork,” PyPI Documentation. [Online]. Available: https://pypi.org/project/pillow/. Accessed: Jul. 2025.

[67] Python Software Foundation, “screeninfo: Screen resolution and display information,” PyPI Documentation. [Online]. Available: https://pypi.org/project/screeninfo/. Accessed: Jul. 2025.

[68] Ultralytics, “ultralytics: Object Detection, Segmentation, Tracking & Classification,” PyPI Documentation. [Online]. Available: https://pypi.org/project/ultralytics/. Accessed: Jul. 2025.

[69] Robotis, “TurtleBot3 Appendix: LDS-02 (LiDAR Sensor),” Robotis e-Manual. [Online]. Available: https://emanual.robotis.com/docs/en/platform/turtlebot3/appendix_lds_02/. Accessed: Jul. 2025.

[70] Robotis, “TurtleBot3 Appendix: RealSense Sensor Integration,” Robotis e-Manual. [Online]. Available: https://emanual.robotis.com/docs/en/platform/turtlebot3/appendix_realsense/. Accessed: Jul. 2025.

[71] Robotis, “TurtleBot3 Features,” Robotis e-Manual. [Online]. Available: https://emanual.robotis.com/docs/en/platform/turtlebot3/features/. Accessed: Jul. 2025.

Appendices

Appendix 1 list of figures

Appendix 2 list of tables