Development and Simulation Of ACES: An Autonomous System for Collaborative Multi-Robot Exploration and Map-Building Using ROS

Group Team Report

MEng Team Project – MMH723842-20-AB

Project Supervisor	Dr Mario Mata
Project Team	Mateo Alvarez Luke Byrne Jamie Hardie Kamil Shabkhez Ryan Walker

ABSTRACT

This report presents development work undertaken in the creation of the **Automated Collaborative Exploration Swarm (ACES)**: an open-source multi-robot system designed to perform collaborative SLAM with multiple robots for efficient exploration and mapping of unknown indoor environments. The software has been developed using ROS middleware and open-source packages, with the final system and experiments simulated using *Gazebo*. The development of the system has been informed by a methodical review of literature, providing outcome of study in areas including SLAM, autonomous navigation, exploration, and collaborative map-building. While all work within the scope of the project has been carried out in simulation due to lab access limitations, hardware requirements and practical viability of the system for use in real-world applications are also considered.

Development work carried out on ACES system includes the implementation of both 2D and 3D SLAM packages, as well as a thorough account of ROS software configuration for robot control, the frontier exploration package, the navigation stack, and the map-merging solution, as well as 3D modelling of simulated robots.

Extensive validation testing of the system has been conducted, followed by a range of experimental testing procedures to determine robustness, scalability, and accuracy of the multi-robot mapping capabilities achieved, using simulated mazes developed by the team.

This paper provides analysis and discusses planning and development work carried out by the team as a group, discussing challenges faced and strengths displayed, as well as considering future work in the project timeline.

ACES project code and models are available on the project GitHub:

https://github.com/Luke-Byrne/ACES.

Table of Contents

LIST OF FIGURES

LIST OF TABLES

ABBREVIATIONS

Abbreviation	*Description*
ACES	Autonomous Collaborative Exploration SWARM
ROS	Robot Operating System
CAD	Computer Aided Design
SLAM	Simultaneous localization and mapping
CSLAM	Collaborative simultaneous localization and mapping
VISLAM	Visual-Inertial SLAM
CPU	Central Processing Unit
MCU	Microcontroller
OS	Operating System (e.g., Ubuntu)
PWM	Pulse width modulation
LIDAR	Light detection and ranging
IMU	Inertial Measurement Unit
URDF	Unified Robotic Description Format
AMCL	Adaptive Monte Carlo Localization
VM	Virtual Machine

ACKNOWLEDGEMENTS

The team would like to express deep appreciation and sincere gratitude to the Project Supervisor, Dr Mario Mata, for his enthusiasm, guidance, motivation, and humour throughout the year.

We would like to extend our gratitude to everyone involved directly or indirectly in the completion of this work, including the staff of Glasgow Caledonian University, and in particular, the Engineering department that has made it possible to conduct this year of studies through remote learning, despite exceptional circumstances.

Furthermore, each member of the team would like to acknowledge and thank their friends and families for the continuous support and motivation, and for understanding the number of hours each engineer has had to invest in the development of this project.

DECLARATION

The team declares that this project submitted to Glasgow Caledonian University, titled "Development and Simulation Of ACES: An Autonomous System for Collaborative Multi-Robot Exploration and Map-Building Using ROS" is except where explicitly stated, an original work completed by the team under the supervision of Dr Mario Mata.

This work is submitted for the fulfilment of the requirements for the award of *Master of Electrical and Electronic Engineering* and will not be submitted in part or in full for the completion of any other award or diploma in any other University or Institute.

Chapter 1 INTRODUCTION

The engineers have been provided a brief for this master's project requesting the development of a Jetson-based robot using ROS, with the ultimate scope of the investigation and resulting system design left to the group's discretion.

In response to the brief, the team has chosen to focus on multi-robot collaborative robotic mapping and exploration, an area of active research and ongoing development. Over recent decades, many systems for Collaborative Simultaneous Localization and Mapping (CSLAM) and autonomous navigation have been proposed. However, there is currently no consensus as to which system frameworks are most efficient; nor are there many fully encapsulated robot and software packages available for consumers or researchers who wish to use this new technology.

As such, the engineers have undertaken development and validation of **ACES (Automated Collaborative Exploration Swarm)** - a system that uses SLAM-equipped robots to autonomously explore an environment, transmitting individual maps that can be merged with the aim of increasing completion reliability, completion speed, and accuracy of resulting output maps.

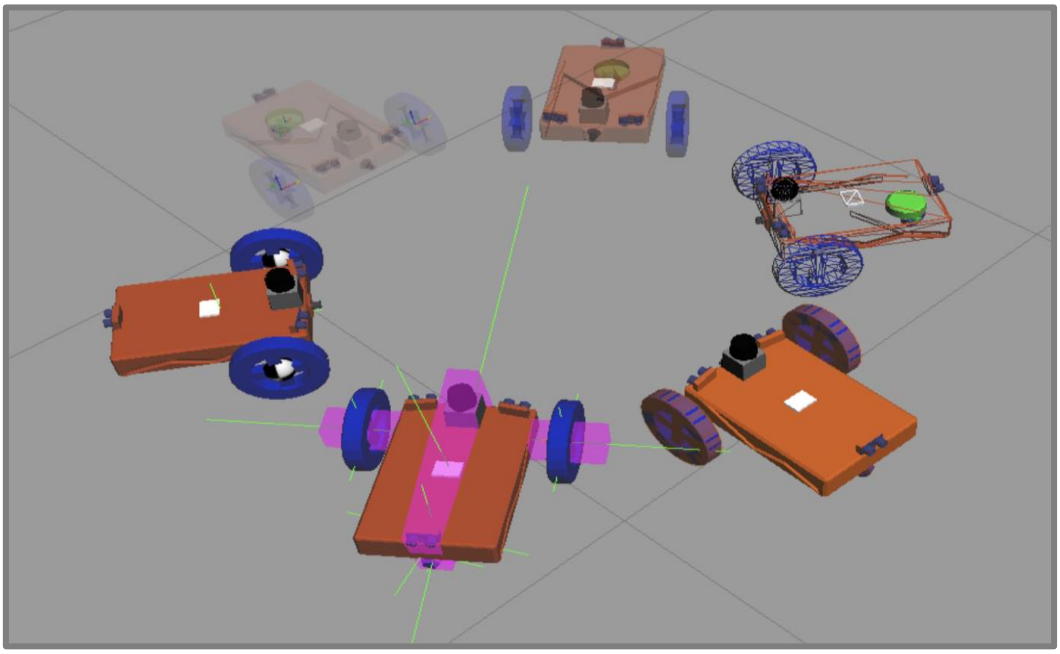


Figure 1-1 – Autonomous Collaborative Exploration SWARM (ACES)

Multi-robot systems can perform complex tasks that are inefficient or unfeasible for a single robot to accomplish: the fundamentals of multi-robot systems involve splitting a complex problem into sub-problems, then distributing sub-tasks accordingly to individual units within the multi-robot system. The robots within the system interact with each other to solve a task. To achieve complex tasks, simple robots can be designed to cooperate to achieve their goal.

Multi-robot systems have the potential to be very cost-efficient as opposed to designing a single expensive robot – such configurations are fault-tolerant, so using a team of robots increases the reliability and robustness of the system because they are typically decentralised and distributed, providing more redundancy with increased scaling of the system.

Due to the ongoing Covid-19 pandemic legislation prohibiting access to Glasgow Caledonian University resources, the engineers have been limited to implementing and validating the system design in software. *Gazebo* simulation toolkit is to be used for this purpose, providing dynamics simulation, sensor simulation, and graphical representation of the mapping and exploration capabilities to provide experimental results.

The team aims to package the developed and validated ACES software as an open-source deliverable, granting free access to users for educational and developmental purposes. Through hosting the package on GitHub, the team aims to provide means of importing and adapting the system to end-user requirements and hardware compatibility. With the development scope of this project is limited to a simulated system, the team has a wide range of choices regarding mapping sensors available for potential development. To take advantage of this, the team aims to focus the system development to investigate 2D and 3D SLAM solutions for use in the ACES framework.

This report thoroughly details the development of the ACES system over the timescale of the project, informed by a comprehensive literature review on the topic of robot automation, mapping, and collaborative operation, as well as building on the engineers' experience working with ROS in previous laboratory work focussed on Intelligent Robotics and utilising skills learned from studies in undergraduate Mechatronics modules at Glasgow Caledonian University.

1.1 SUMMARY OF PROJECT OBJECTIVES

- Design and implement a multi-robot collaborative mapping and exploration system of suitable complexity, directed by the research of current literature in the field of robotic design based on ROS middleware. The scope of the project should be ambitious but practical to achieve within the allowed timescale.
- Develop 2D and 3D SLAM capabilities for the system to compare the advantages and disadvantages of each strategy.
- Package and present a final deliverable of the system in a format rendering it suitable for open-source access.
- Design and execution of experiments to assess the valid operation, accuracy, and robustness of the fully developed system.
- Present a discussion of experimental results and assess the validity of the implemented ACES system.

1.2 STRUCTURE OF THE REPORT

The team has aimed to provide a clear, concise account of the research, development, and testing of the ACES system, reflecting the structured approach taken by the engineers in the undertaking of the project.

Chapter one of this report introduces the project, outlining the aims and objectives of project development and contextualising them within the current state of multi-robot system research.

Chapter two presents a comprehensive literature review, providing outcomes of an investigation into critical development areas - ROS, multi-robot communication, SLAM, map-merging, autonomous exploration and navigation, and robot hardware configuration.

Chapter three provides a detailed account of work undertaken by the team in developing and implementing the ACES system. This includes an account of preliminary experimentation with the development software, the design, and simulation of the ACES robot within *Gazebo*, and the development of the ACES ROS package, providing the team of robots with exploratory collaborative mapping functionality based on SLAM.

Chapters four and five describes the design and outcome of tests used in the verification of ACES functionality and details of experiments carried out assessing scalability, mapping accuracy, and optimal maximum robot velocity.

Chapters six and seven and eight provide a discussion of the project outcomes, including an analysis of the strengths and weaknesses of the final system based on verification testing and experimental results and a summary of recommendations regarding future development. The report concludes with a critical evaluation of team performance and closing remarks.

Chapter 2 LITERATURE REVIEW

2.1 RESEARCH METHODOLOGY

Following pre-development experimentation with the simulation software, the team has carried out a literature review relevant to the project objectives. The key research areas considered during the conduction of this literature review are:

- ROS and *Gazebo* functionality
- Collaborative mapping and communication in multi-robot systems
- SLAM sensor configurations, identifying the application to map-merging and datasets for verification.
- Exploration strategies for autonomous navigation
- Robot hardware design principles

To cast as wide a net as was practicable, a broad search was conducted using the open-access research archive platform arXiv and IEEE Xplore journal database, collecting a wide range of documentation relating to multi-robot systems. With the initial search providing a list of 420 papers, researchers have proceeded to manually highlight relevant material and condense the initial list to 165 entries. Relevant papers have been analysed by the group, grouped by topic (for example Navigation, SLAM, Map-merging, etc.), and ranked in terms of relevance to the project objectives. As well as expanding technical knowledge in key development areas, the team has been able to use this phase to identify open-source ROS packages that can potentially be implemented into the ACES system, allowing for rapid prototyping and implementation of features during the following developing stage.

2.2 ROS – ROBOT OPERATING SYSTEM

Robot Operating System (ROS) is an open-source middleware suite. It includes hardware abstraction, low-level device control, implementation of commonly used features, message-passing between processes, and package management, as well as other services expected from an operating system [1]. It is equipped with tools and libraries for downloading, designing, writing, and running code on multiple computers. ROS is broken up into three

levels of concept: The Filesystem level, the computational graph level, and the community level [2].

The Filesystem level contains the resources the user encounters when building the project. Resources that are found within the filesystem are as follows: Packages, Metapackages, Package manifests, message types, service types. ROS computation graph consists of nodes, master, parameter server, messages, service, topics, and bags. Nodes are responsible for handling the computational processing.

Robot systems can include numerous nodes. These handle tasks such as path planning, motors control, map merging, etc. The master allows the nodes to find each other, providing name registration which allows the nodes in the system to be located, exchange messages, and invoke services. Nodes communicate with each other by passing data structures known as messages. By sending a message, nodes can publish information to the topics. In the opposite direction, if a certain type of data is being published to a topic, a node can subscribe to receive that data. There can be multiple publishers and subscribers to a single topic. Services allow the publish/subscribe model to be flexible which is needed for a distributed system. Bags store the information and can playback the message data this is useful when sensor data needs to be saved [3].

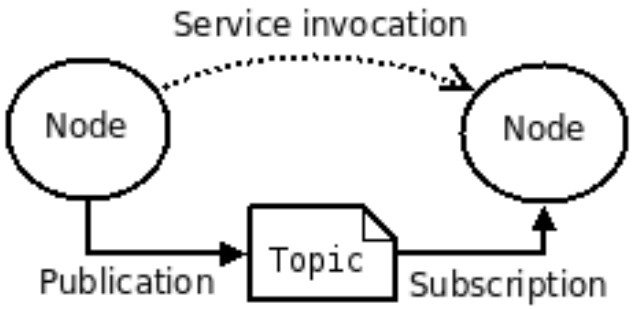


Figure 2-1 – Flow diagram of Node and Topic Communications [3]

Figure 2-1 displays the communications between nodes and the topic, illustrating how nodes can connect directly with each other using the namespace provided by the master.

ROS community level allows for ROS users to communicate with each other to exchange software and knowledge. This is achieved through distributions, repositories, and ROS wiki [3].

2.3 SIMULATION IN GAZEBO

Gazebo is the default physics-based robot simulator/GUI that integrates with ROS. This software provides 3D simulations of robots, environmental obstacles, and other factors. This simulation environment allows the user to encapsulate and customise the physical properties and kinematics of each component of the simulation robots and their environment. *Table 2-1* provides a summary of these properties identified in the review of associated documentation:

Table 2-1 – Properties of Gazebo Models

Properties	Description
Links	Contains the physical properties of each component of a model.
Collision	Provides a collision checking element of the selected geometry.
Visual	Provides a visual element of the selected geometry.
Inertial	Describe the dynamic inertial properties of the selected geometry.
Joints	Provide a connection between two links/components of a model.
Plugins	Provide control and functionality for the model.

2.4 COLLABORATIVE MAPPING

Collaborative mapping involves the contribution of different individuals or entities, supplying complementary parts of information that will ultimately create a geographic information system [4].

Technological modernisation influenced positively on the development of collaborative mapping, with services like Google map creator or OpenStreetMap allowing users to share portions of data that will contribute towards the generation of a large-scale map.

Collaborative mapping has many applications: Disaster damage assessment & logistics [5], public transport maps [6] [7], Epidemiological risk assessment [8], Migratory patterns of certain animal species [9], and many others.

Applied to robotics, Collaborative mapping can be performed by a group of mobile robots that generate 2D or 3D maps using Simultaneous Localisation and Mapping (SLAM).

2.5 MULTI-ROBOT SYSTEMS

In recent years, research has been conducted into replacing complex single robots with a multi-robot or Swarm-robot system. These robots work cooperatively to achieve the same results or exceed the single robots' capabilities. Cooperative robots are expected to have little human intervention while achieving the goal-orientated task provided by the user.

SWARM robotics is described as a field of multi-robotics that coordinates a large population of robots in a decentralised manner [10]. It is an approach based on the strength and flexibility of the group that can perform complex tasks with basic individuals. Beni [11] defines the connection between SWARM robotics and certain insects' behaviour as:

"The group of robots is not just a group. It has some special characteristics, which are found in Swarms of insects, that is, decentralised control, lack of synchronisation, simple and (quasi) identical members."

Swarm robots are becoming increasingly popular for task allocation and mapping. Using multiple robots allows for areas to be mapped at a faster rate, while task allocation can be split between the robots as opposed to using one complex robot to complete the tasks.

Data communication between the robots in a multi-robot or SWARM robot system is not trivial. When designing a collaborative mapping system, a global map must be created to avoid unknown areas being mapped more than once, which would decrease the efficiency of the system. There are two main methods to handle the mapping of Swarm robots' systems, single-master and multi-master systems [12] [13]

2.5.1 SINGLE-MASTER SYSTEM

Using a single master, the mapping is handled on one machine which is referred to as the master. The nodes are distributed throughout the other machines connected to the network, apart from the driver nodes which are connected to the master. The ROSCORE command will only be executed by the master. The main drawback with implementation is if the master has any connection problems or any other technical issues the system will become unresponsive and become unable to perform the tasks set by the user [12] [14] [13].

2.5.2 MULTI-MASTER SYSTEM

Using a multi-master system allows for each robot connected to the network to run the ROSCORE command. The multi-master system resolves the main issues that are common in a single master system. With each robot becoming the master it means if any one robot fails the systems can still run with the available robots. Due to each of the nodes becoming localised this reduces the bandwidth as only a limited number of nodes are being shared. However, using multiple masters creates a larger computational need which slows the system and creates the need for larger memory capacity to be available [12] [14].

2.5.3 CLOUD ROBOTIC SYSTEM

Using a cloud robotic system allows a multi-robot system to cut down on the heavy computational tasks being handled by the onboard hardware. A robotic cloud engine can be used to handle the management of the multi-robot system. Using the nodes available in the ROS package the tasks can be moved to the cloud engine. Using a cloud engine requires both software and hardware implementation to the system [13] [14] [15] [16].

2.6 COLLABORATIVE SLAM

Distributed and multi-robot SLAM systems can be effectively divided into two sections: the local SLAM mapping algorithm for each robot, and the global map-merger system. Local map creation is often performed using traditional SLAM algorithms such as ORBSLAM, which use feature extraction methods such as SIFT and SURF, bundle adjustment, and pose graph optimisation to produce a point cloud map and robot pose estimates [17]. Several approaches have been taken to tackle the map merger problem, often incorporating the use of ROS2 communications functions [18].

2.6.1 GLOBAL MAPPING

Using a map builder to stack the sensor data from each robot in the system will allow the implementation of SLAM. Combining the data from each robot will output a global map which can then be localized to each robot. This will allow each robot to use high-level path planning and exploration which can be relayed back to the interface providing the user with visualised results [15] [16].

2.6.2 LOCAL SLAM

Traditional SLAM algorithms may be used to produce local maps for each robot, provided the point cloud and pose estimations of the algorithm are compatible with the global map merger system. As such, the same issues must be considered when selecting hardware for both single and multiple robot SLAM systems. The primary concern is sensor selection, as many sensor technologies have been incorporated into real-time SLAM systems, and each has associated positives and negatives [19]. However, decisions must also be made as to whether a 2D or 3D system is most appropriate for the given application.

2.6.2.1 SLAM SENSORS

Monocular SLAM systems use the video feed of regular RGB cameras to estimate robot pose and produce point cloud maps. These systems rely on repeated observation of visual features, to derive feature parallax [20]. Monocular SLAM is based on keyframes and bundle adjustments which have a low computational cost and high accuracy. One of the flaws in this method is the information that is stored in the edges and shapes is lost. Using a direct approach can solve this issue, however, object detection and place recognition become inaccurate [21] [22] [23]. Monocular systems are also subject to scale drift unless inertial sensors are used to correct this [24]

Stereo camera systems consist of two coupled monocular cameras at a fixed distance and angle. The use of two cameras allows for rapid triangulating of matched points between images. As such depth information is easily extracted. These systems are highly robust to dynamic environments, but still suffer from errors when operating within low light levels [20]

RGB-D cameras, such as the MicROSoft Kinect, use an RGB camera and a separate infra-red transmitter/receiver working as a depth sensor [25]. This system is also sometimes called active stereo. Unlike monocular and stereo camera systems, RGB-D systems are robust to changes in lighting, and function well in low texture environments. However, the depth sensors typically do not have the range of LIDAR sensors [19].

LIDAR sensors can be used for scan matching where consecutive scans are matched to gain the trajectory of the robot. Using SLAM Hector these scans can be matched using little processing power [8]. Hector SLAM uses a Gaussian-Newton equation which best fits the laser beams with the map. Using high-rate scans for accuracy can have its drawbacks. When the

robot is traveling at high speeds the pose estimation will become inaccurate due to it relying on the endpoint of the laser beams within the map being produced [21] [22].

Visual-Inertial SLAM (VISLAM) systems combine visual sensors such as monocular or stereo cameras with a tightly coupled inertial sensor system [26]. The incorporation of inertial sensor measurements overcome many of the limitations of visual SLAM such as scale drift and spurious loop closure. However, these systems are still vulnerable to changes in lighting [24].

Visual-Inertial SLAM (VISLAM) systems combine visual sensors such as monocular or stereo cameras with a tightly coupled inertial sensor system [26]. The incorporation of inertial sensor measurements overcome many of the limitations of visual SLAM such as scale drift and spurious loop closure. However, these systems are still vulnerable to changes in lighting [24].

2.6.2.2 CURRENT RESEARCH TRENDS

As with most areas of robotics research, a great deal of focus has been paid to the use of machine learning in SLAM development. Such methods have been applied to feature extraction [27] [28], pose estimation [29] [30], loop closure [31], and object detection [32]. However, the speed, accuracy, and processing power requirements of these systems have been heavily criticized and as such, they have not yet seen widespread adoption [33].

As discussed previously, there is currently a trend towards the inclusion of inertial measurement unit data in visual slam implementations, as these offer more robust and accurate pose and map estimates [24] [26]. There is also a move towards the inclusion of other sensor types such as GPS and sonar using extended Kalman filters [31] [34]. However, although the inclusion of more sensor data increases overall system accuracy, this also leads to increases in processing power requirements.

Visual SLAM systems rely on comparing the positions of static features between image frames to estimate feature location and camera pose [31]. However, reliance on this technique typically causes SLAM systems to become inaccurate or completely non-functional in dynamic environments containing a high degree of movement [35]. As such, there is currently a trend in the research towards increasing SLAM system robustness in dynamic environments. This area is significant to multi-robot systems wherein the robots are in view of each other, and to robots that may operate outdoors where weather conditions may cause a high degree of movement in the environment [36]. Coupling of the visual slam system with other sensor

units has shown some success [34], as have incorporation of the BRISK algorithm in the SLAM front-end [37] and the introduction of object detection with probabilistic data association [38].

2.6.2.3 SLAM ACCURACY

SLAM systems perform two functions simultaneously, mapping and localisation. These two processes produce two separate outputs, the area map and the camera or robot pose within the map. As such, different methods have been proposed to assess both map and pose accuracy. In both cases, the output estimates from the SLAM system are compared to known 'ground truth' values. In simulated environments, the ground truth values can be extracted directly from the simulator [36]. However, measuring a system's accuracy in the real world typically relies on the use of datasets. SLAM datasets contain ground truth data about the camera pose, and as such can be used to assess SLAM pose accuracy. However, the resources required to produce ground truth data about the environment itself means that currently there are no datasets that can be used to assess map accuracy.

Pose accuracy is typically assessed using absolute trajectory error (ATE) and relative pose error (RPE) [24]. These methods have been criticized for their failure to properly account for the impact of loop closures, which are a vital part of modern SLAM algorithms [39]. However, despite this, they are still the most widely used method of benchmarking SLAM algorithms. ATE is the root means squared value of translational errors for a dataset sequence, or section of a sequence [40]. RPE is the root means squared value of the sum of both translational and rotational errors for a sequence or section of a sequence [41].

2.6.2.4 SLAM DATASETS

SLAM datasets contain images, videos, or ROS bag files recorded from certain sensors, along with accompanying ground truth data about sensor pose. These are used to standardise the testing of SLAM systems. However, some researchers [42] have noted that full and proper testing requires the use of multiple datasets. For this investigation, LIDAR and RGB-D datasets were of particular importance, as LIDAR SLAM systems are easily the most robust, although most expensive of the sensor types. Although RGB-D and monocular systems are not as robust, they are the most cost-effective solutions. RGB-D sensors produce a regular colour

camera feed and a depth camera feed. As such RGB-D datasets are capable of testing both RGB-D and monocular SLAM systems.

Table 2-2 – Table of Available SLAM Datasets

Dataset	Sensor Type
[43] KITTI SLAM	LIDAR
[44] Complex Urban	LIDAR
[45] Ford Campus	LIDAR
[46] Marulan	LIDAR
[47] CollaborativeSLAM	LIDAR
[48] Sugar Beets	LIDAR, RGB-D
[49] Robot@Home	LIDAR, RGB-D
[50] CoRBS	RGB-D
[51] TUM RGB-D	RGB-D
[52] ICL-NUIM	RGB-D
[53] ICL-Dataset	RGB-D
[54] SceneNet RGB-D	RGB-D
[55] InteriorNet	RGB-D, Stereo

2.6.3 MAP MERGER

Multiple solutions have been proposed for the problem of global map merger [56]. These systems tend to use either a centralized, or a distributed system. Centralized systems use individual ROS nodes which take in multiple 2d or 3d maps, perform merger, and publish a global map [57]. Distributed systems are considerably more complicated. They often rely on direct peer-to-peer communications [58], or mesh network systems working in addition to

ROS [59]. However, some distributed systems work within ROS, sharing sparse data to perform loop closure on a global scale [17]. These distributed systems claim to be more efficient in terms of computation and bandwidth requirements.

2.7 AUTONOMOUS EXPLORATION

2.7.1 SWARM NAVIGATION STRATEGY

Consideration must be given to the autonomous navigation strategy for the Swarm of robots. Usually, robots in these numbers have limited capabilities, achieving their goals by emerging behaviours [60]. Exploration strategies can be generally classed as **random**, **information** or **guide** based:

Guide-based approaches are impractical for development considering that the ACES system goal involves exploration of unknown environments. The mapping units will not have resources such as a pre-created map or specific position goals to inform movement, therefore a method of dictating.

Random exploration approaches promote uniform dispersion of the Swarm in the environment from an undefined initial position – examples of algorithms for implementing this behaviour include Brownian motion, L'evy walk [61], correlated random walk, and ballistic motion [62]. Random walk variant based on Brownian motion, correlated random walk, and L'evy walk share the same mathematical model shown in *Figure 2-2*: robots choose a direction at random and move towards it until a new direction is selected. Movement for each variant is defined by parameters 'ρ' which determines the turning angle when the robot selects a direction at random, and 'μ' is defined as the time the robot keeps moving along its chosen trajectory.

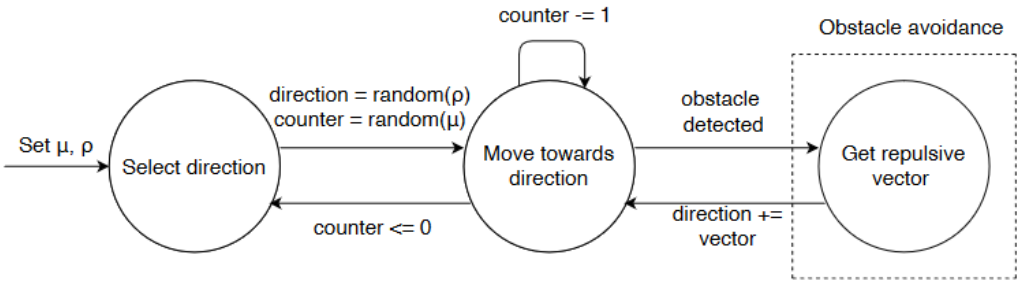


Figure 2-2 – Finite state machines comparison of random walk variants [62]

Alternatively, ballistic motion drives each unit in the Swarm with constant straight motion, only changing the angle when an obstacle is detected as illustrated in *Figure 2-3.* This approach can be classed as a random walk behaviour because the robots select the new direction at random [62].

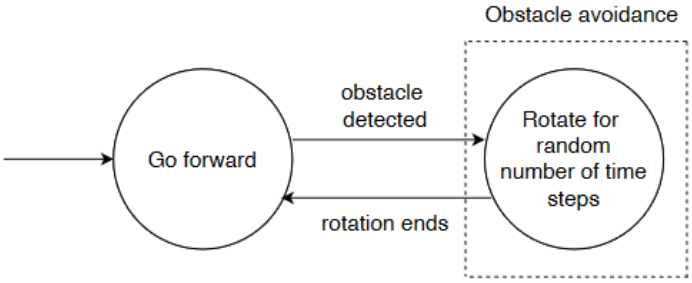


Figure 2-3 – Ballistic Motion State Machine [62]

Information-based strategies are significantly more computationally intensive but allow individual robot navigation to be carried out more efficiently, informed by data gathered from the environment and other units in the Swarm. Arpino et al compared several approaches, including Individual and Group Dynamic Coverage, Potential Fields, and DMA-RRT [63]. The Rapidly-exploring Random Tree (RRT) model as illustrated in *Figure 2-3* is highlighted as a particularly robust algorithm with minimal tuning required: a key advantage of RRTs include an approach heavily biased toward unexplored regions of the environment, thus providing consistent robot behaviour and a strategy that directs the Swarm to deliver a completed map of an environment. The algorithm is relatively simple making it efficient and adaptable to a range of path planning systems [64]. A search of existing open-source ROS packages identified RRT exploration [65] as a viable solution for robot development. Information-based strategies are significantly more computationally intensive but allow individual robot navigation to be carried out more efficiently, informed by data gathered from the environment and other units in the Swarm. Arpino et al compared several approaches, including Individual and Group Dynamic Coverage, Potential Fields, and DMA-RRT [63]. The Rapidly-exploring Random Tree (RRT) model as illustrated in *Figure 2-4* is highlighted as a particularly robust algorithm with minimal tuning required: a key advantage of RRTs include an approach heavily biased toward unexplored regions of the environment, thus providing consistent robot behaviour and a strategy that directs the Swarm to deliver a completed map of an environment. The algorithm is relatively simple making it efficient and adaptable to a range of path planning systems [64].

A search of existing open-source ROS packages identified RRT exploration [65] as a potential solution for robot development.

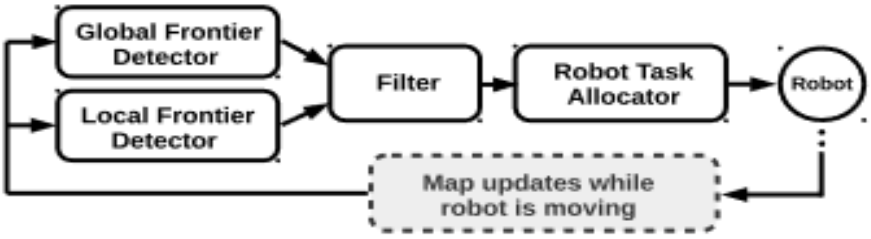


Figure 2-4 – RRT Exploration Strategy Diagram [64]

Research of robot navigation strategies also revealed the explore_lite [66], a greedy frontier-based [67] exploration package adapted from explore [68] that distinguishes optimal direction for exploration based on the frontier_exploration algorithm [69] packages. Advantages of this package include its 'lightweight' description since it does need to create a costmap, reducing reliance on processing resources, and easier to configure. *Figure 2-5* shows the relatively simpler explore_lite functional flow diagram – the node subscribes to nav_msgs/OccupancyGrid and map_msgs/OccupancyGridUpdate topics from the robot's SLAM system and geometry_msgs from the navigation stack and the algorithm efficiently aims for an unexplored region only considering the immediate vicinity and communicates this to the unit's navigation system.

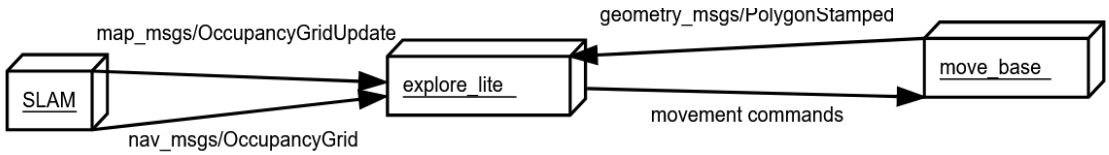


Figure 2-5 – explore_lite function block diagram [49]

Informed by the team's research regarding communication strategies for the ACES robots, it was decided that a decentralised SWARM navigation approach would be preferable to employ a coordinated path-finding system where the ROS master program dictates travel instructions to the mapping robots. Primarily, the SWARM strategy is advantageous in that it is more robust: due to each robot determining an individual exploration strategy locally without the need to wait for instructions before moving, the decentralised approach guards against the mapping robots freezing if they leave the signal range of the master. Centralised exploration instructions would also be vulnerable to errors and interference, while also limiting the

effectiveness of the map merging functionality in the field. A decentralised method also eliminates the likelihood of a bottleneck in data transfer between the ROS master and the mapping robots: whereas a centralised method may be practical in providing faster coverage of an area for smaller teams of robots, as the total number of units increase each robot would need to wait longer to receive instructions on where to travel. Remapping of areas is also potentially advantageous in that accuracy of the final merged map can potentially be increased through robots scanning the same part of the unknown area from different angles.

The literature review identified Rapidly-exploring Random Tree (RRT) package *rrt_explore* [65] as the optimal strategy for multi-robot pathfinding for collaborative mapping – this method is advantageous in that it provides superior pathfinding ability. However, this approach proved ultimately impractical for use in the ACES system as the team was unable to source an open-source package using the RRT algorithm currently maintained for ROS Noetic or Melodic versions used by the systems other components – after considering the potential for development of the existing RRT package to migrate it to the more recent OS version, the team elected to explore an alternative to assure that a functional system would be available for testing of the robot mapping functionality. Alternatively, the *explore_lite* package is currently maintained for ROS noetic and melodic – for this reason, the team expected that this option would be faster to implement and configure than *rrt_explore* to achieve the primary project goal: testing the SLAM and map merging capabilities of the ACES system. Further to this, explore_lite was found to be implemented alongside the *m_mapping* package during the development of map merging functionality, assuring the team of likely compatibility.

2.7.2 COLLISION AVOIDANCE

To perform local collision avoidance in a realistic environment, a robot in a multi-robot system must employ reliable position estimations of itself within the workspace. Methods must also be used to deal with uncertainty in positions and possible actions of the other robots in the Swarm. While centralised systems can update individual units on positions of robots in the Swarm, alternative methodology such as collision avoidance based on velocity obstacle paradigm described by Claes et al [70] is required in a decentralized approach.

A novel secondary mechanism for the detection of fellow Swarm robots in the Swarm is implemented by Bultman et al, wherein the navigation system checks for a 'signature' composed of unique structural characteristics of the robots in use via data from the mapping camera. If the signature is detected close to the robot's position, a stop command avoids a collision [71]. A third barrier to collisions between robots and the environment requires the triggering of evasive manoeuvres triggered by close-proximity sensors, as considered in the following investigation of robot hardware configuration.

2.8 ROBOT PHYSICAL CONFIGURATION

2.8.1 STEERING GEOMETRY

Different steering geometries options were researched to identify the optimal configuration. Popular geometries for autonomous robots are **Differential Drive** [72], **Ackerman Steering Geometry** [73], **Skid-Steer Configuration** [74].

In an evaluation of the Differential Drive configuration, several key advantages have been identified. Most notably a high manoeuvrability while maintaining simplicity both mechanically and operationally. Ackerman Steering Geometry, while structurally robust, lacks the manoeuvrability and simplicity mentioned above. For this reason, Ackerman drive has been deemed inferior to differential drive for this application. Evaluating the suitability of the Skid-Steer Configuration, it has been determined that in terms of simplicity and manoeuvrability it was on pair with Differential Drive but requires a supply of power to four wheels instead of two. While this could prove useful in navigating uneven terrain, this results in poor energy efficiency, basing angular rotation on overcoming the friction of the wheels with the ground. As the ACES system is designed for indoor mapping, this does not provide an advantage over employing a Differential Drive configuration.

Justified by these findings, the team has carried out further investigation of kinematics associated with the Differential Drive steering configuration. The kinematics of the robots involve the physical sizes and geometric properties of the robot. It is the association between the robot's dimensions and its position, velocity, and acceleration.

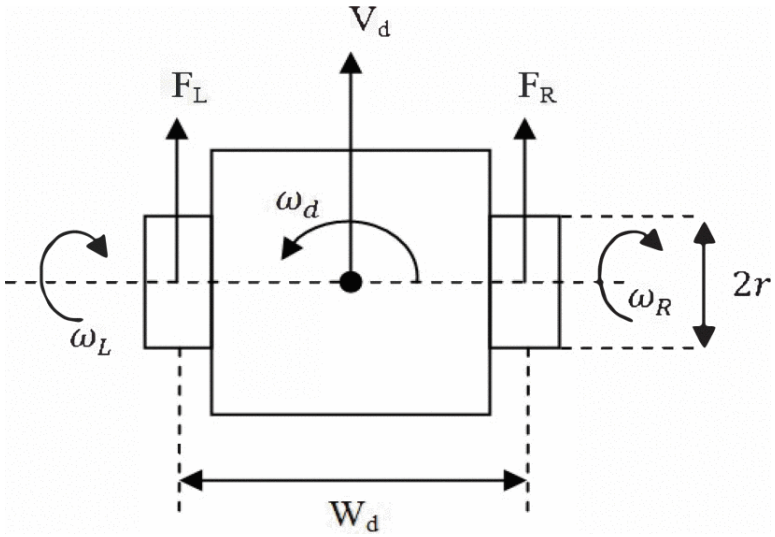


Figure 2-6 – Representation of differential drive robot [75]

As illustrated in *Figure 2.4*, the linear velocity of the robot is represented by V_d, while ω_d is the rotational speed around its center of mass. Both V_d and ω_d are derived from ω_L and ω_R, the left and right wheel angular speed. Finally, F_R and F_L denote the linear speeds that each wheel is moving at. The geometric properties of the robot are reflected on the schematic are W_d representing the distance between wheels, and r indicating radius of the wheels, which should be of equal dimensions. The equations defining the kinematics of the differential-drive robot are provided in *Table 2-3*:

Table 2-3 – Kinematic Equations [75]

Factor	Equation	
Overall angular velocity	$\omega_d = \frac{r}{W_d}(\omega_R - \omega_L)$	*(Equation 1)*
Angular velocity [left wheel]	$\omega_L = V_d - \frac{W_d}{2}\omega_d$	*(Equation 2)*
Angular velocity [right wheel]	$\omega_R = V_d + \frac{W_d}{2}\omega_d$	*(Equation 3)*
Linear velocity	$V_d = \frac{r}{2}(\omega_R + \omega_L)$	*(Equation 4)*
Linear velocity [left wheel]	$F_L = \omega_L(\mathrm{R} + \frac{W_d}{2})$	*(Equation 5)*
Linear velocity [right wheel]	$F_R = \omega_d(\mathrm{R} - \frac{W_d}{2})$	*(Equation 6)*

Angular velocity values for the left and right wheel are obtained from wheel encoders. Using these equations, the angular and linear velocity from the robot is calculated from the rotational speed occurring at each wheel, derived from the encoder count. A limitation of the

simulated representation of the physical robot is that ideal conditions of 'zero' slippage are imposed on the system.

2.8.2 ROBOTS HARDWARE CONSIDERATIONS

Despite this project being developed entirely in a virtual environment, hardware elements required for the functionality of mapping robots have been investigated, ensuring the developed system is suitable for physical implementation in the real world.

A prominent product used for experimental mapping has been identified as the TurtleBot3 Burger shown in *Figure 2-7*. The hardware platform is designed for research and educational use in a range of robot applications. The device notably features a scalable structure for simple positioning and interchanging of components such as sensors [76].

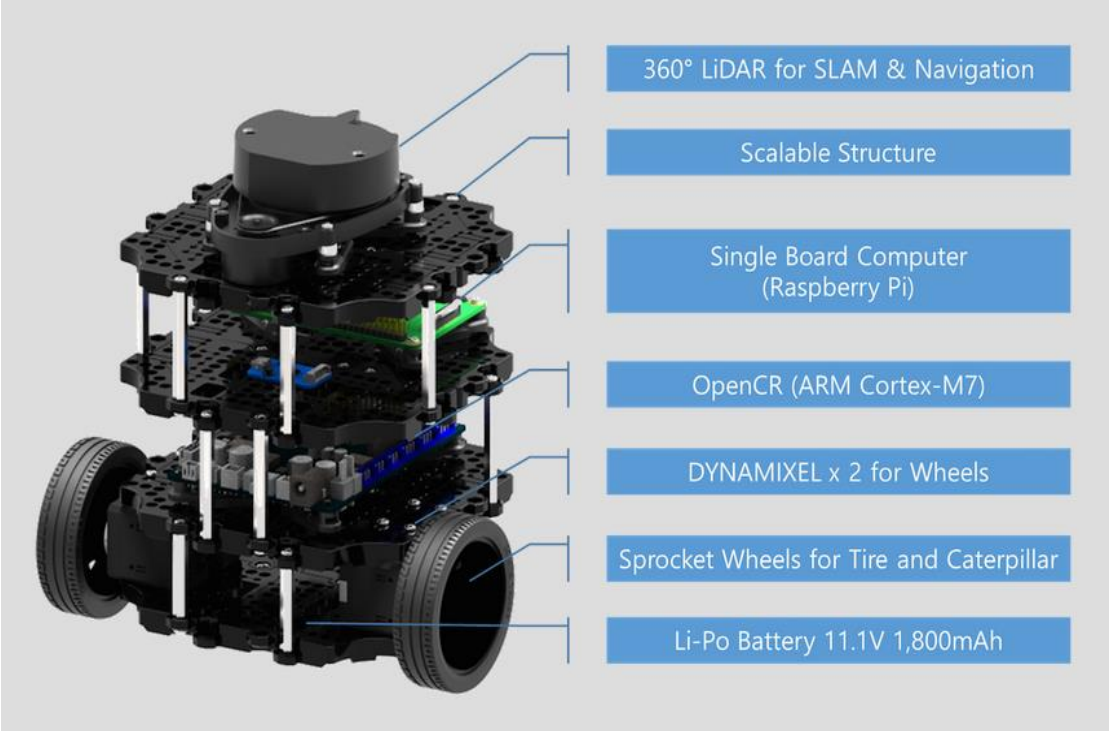


Figure 2-7 - Turtlebot3 Burger Hardware Configuration [76]

Table 2-4 presents an investigation of hardware components critical for navigation and mapping present using the TurtleBot3 Burger bill of materials as a template, as well as identifying alternative and expanded hardware considerations for achieving required functionality dictated by the final developed program.

Table 2-4 – Investigation of Robot Hardware Components

Component	Considerations
Power Supply	In a physical version of the robots, each will have to be powered individually with local batteries. Research indicates the TurtleBot3 can operate supplied by an 11.1V 1,800mA lithium-polymer battery providing power to sensors, motors, and processors [76].
CPU	The onboard CPU specified in the brief for the final implementation of the ACES system is the *NVIDIA Jetson Nano* developer kit. This CPU board is a powerful single-board computer with a small footprint that allows running different neural networks in parallel applications running in 5 Watts [77] and deemed adequate for SLAM, navigation, and map-merging processing tasks proposed by the team. In comparison, the Raspberry Pi board utilised by the TurtleBot3 waffle is a less powerful choice.
MCU	Used to interface, control, and gather data from the electronic components of the robot, the MCU carries out minimally complex processing tasks. This functionality is achieved in the TurtleBot3 Burger unit using a 32bit ST Cortex-M3 Microcontroller board operating at 72MHz [78]. This is implemented virtually in *Gazebo* simulation thus configuration is out of the scope of the project.
Motors	In *Gazebo* simulation, inertia and speed values can be transmitted directly to the wheels. A possible option is to install a set of 2 DC motors with encoders per robot, driven by Pulse width modulation (PWM) with a dual H-bridge motor driver per unit. Alternatively, servomotors can provide similar functionality, such as the pair of Dynamixel XL430-W250 employed as wheel actuators in the TurtleBot3 Burger robot unit [79]

Encoders	An encoder assembly is required to position the motors actuating each wheel. The simplest solution to the addition of encoders would be to acquire motors with pre-installed encoders, such as the Dynamixel units. This is another feature of the robot that is implemented virtually in *Gazebo* within the scope of the project.
SLAM Sensor	SLAM sensor choice for the ACES system is heavily dependent on continuing software development and testing. Hardware elements such as chassis structure, power supply requirements, and motor power required for individual units may vary depending on the final choice.
IMUs	The accelerometer is an Inertial Measurement Unit (IMU) that measures acceleration forces, allowing to derive the speed and calculate the motion of the system [47] and can be also configured in *Gazebo* simulation according to development requirements. Gyroscope IMU measures rotational motion in degrees per second [46] and can be implemented in *Gazebo* simulation according to eventual design.
Close-range proximity Sensor	For the close-range obstacle avoidance system two different sensors were studied: Ultrasonic Sensor and Infrared Range Finder. Ultrasonic sensors are popular components in robotics due to their simplicity and reliable performance. It is based on an emitter unit that sends a 40 kHz ultrasonic wave [79] that bounces off any solid body and returns to be sensed by a receiver unit. Distance is calculated based on the time between emission and reception of the ultrasonic waves. The Infrared Range Finder operates similarly, with the difference of emitting a pulse of infrared light that if reflected by a surface and sensed by the receiver unit allows calculation of the distance of that reflection's origin [80]. Several studies [81] [82] [83] show that the Ultrasonic sensor has more reliability and accuracy compared to the IRrange finder, and therefore it will be the choice of sensor for close-range obstacle detection for the ACES robots.

Wireless Communications	The primary considerations for the robot communications system will be scalability, bandwidth use, and likely robustness in a real-world situation. An example of hardware providing this functionality is the Waveshare Dual Mode Wireless NIC AC8265 [80] – this module offers communication via dual-mode 802.11ac Wi-Fi suitable for the intended robot application, as well as a Bluetooth 4.2 connection.
Robot Chassis	The robot chassis should be physically robust to tolerate exploration of unknown environments which may include static or dynamic hazards. As opposed to the broadly expandable base of the TurtleBot3 unit, the base should maintain a low centre of gravity for stability and efficient manoeuvrability. An increase in weight will reduce battery lifespan, therefore, should be kept as low as possible when considering chassis material, however, manufacturing costs can be high for more advanced materials.
Auxiliary Hardware	Equipment such as cooling systems and wheel tyres should be selected based on the robot's operating environment, with more cooling or softer tyres requiring more power consumption leading to reduced operating time.

Chapter 3 DESIGN & IMPLEMENTATION

3.1 DEVELOPMENT SOFTWARE

The middleware selected for this project is *Robot Operating System* (ROS*)*. The reasoning behind this decision is the versatility and scalability that ROS modularity offers over other options available [81]. The default ROS simulation software *Gazebo* was used to generate virtual testing environments. CAD packages such as *Autodesk Fusion 360, PTC Creo*, and *Blender* were used to create 3D models of the robot. Software package *Blender* also aided the process of rectifying the formatting of the files.

The development of the ACES system has been projected to include the latest version of ROS (Noetic) running on *Ubuntu 20*. However, incompatibility issues within the **tf_prefix** inside the **state_publisher** node provoked the necessary retrograding of the development ROS version to Melodic and corresponding OS *Ubuntu 18*.

3.2 DESIGN METHODOLOGY

The methodology adopted to design the ACES system is illustrated using *Figure 3-1* below.

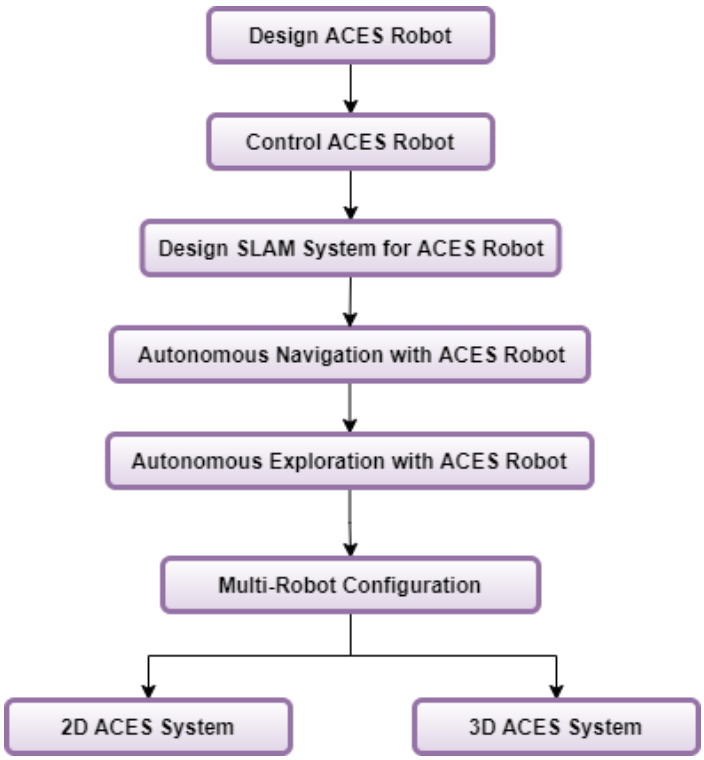


Figure 3-1 – Design Methodology of ACES

To meet the development goals outlined in the ACES project plan, development was split into two phases: designing a single fully functional ACES robot and then enabling the multi-robot configuration.

To achieve this, the steps illustrated in *Figure 3-1* were considered:

- Designing and assembling the 3D model of ACES robot.
- Implementing a method of controlling the robot.
- Designing and implementing a SLAM system for the ACES robot
- Implementing a feature that allows the robot to autonomously navigate.
- Implementing a feature allowing the robot to explore an unknown environment.
- Implement multi-robot configuration for SWARM capability.

3.3 Design & Assembly of ACES Mapping Robot

As justified by the literature review, the steering configuration selected for the ACES robot is a differential drive configuration. Hence, the construction of the robot has been conceived with the kinematics structure required for differential drive steering.

As illustrated in *Figure 3-2*, the final design of ACES robot includes two actuating wheels which determine the direction and speed of the robot and a free moving caster wheel that provides balance and rotation of the robot.

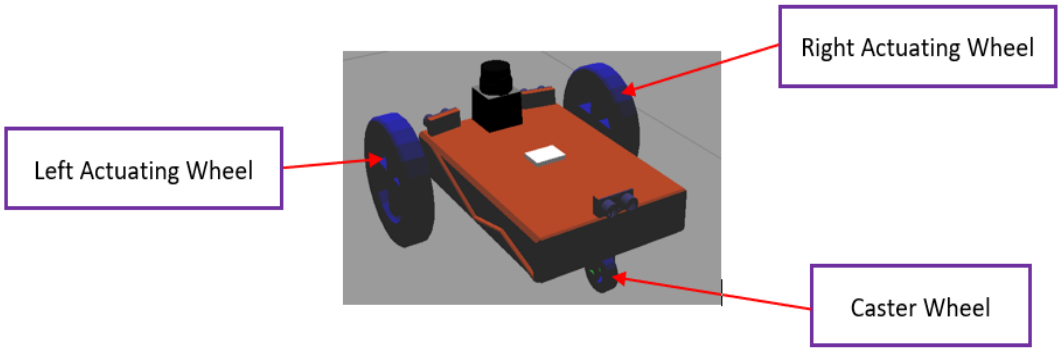


Figure 3-2 – Layout of ACES Robot

To provide an efficient communication channel between the ACES robot and other nodes within the ROS framework, an arbitrary frame called **Main_Frame** is added to the design structure of the ACES robot. To effectively interact with the surrounding environment, three

types of exteroceptive sensors were added to the design of the ACES robot: a LIDAR sensor, three sonar sensors, and a Kinect camera unit. For additional functionality, an IMU sensor is also added. The design structure of the ACES robot also included a base Chassis with all six sensors, the actuating wheels, and the caster wheel attached to this physical link.

The final configuration and design structure may be seen using *Figure 3-3* below:

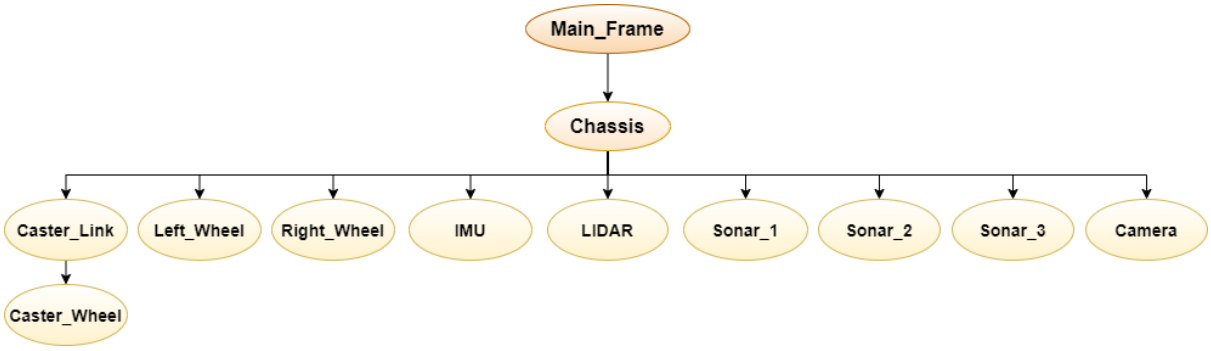


Figure 3-3 – Transfer Tree of ACES Robot

The construction of the simulated ACES robot has included creating separate 3D models for each of the individual components using *Creo Parametric 6.0* and *Autodesk Fusion 360* software packages. Before being imported to the ROS workspace, the 3D design file for each component is passed through *Blender,* which acts as a rectifying tool to convert *.STL* files to *.DAE* files. The *.DAE* files for each component of the robot are then assembled within URDF frameworks to ensure compatibility with ROS and *Gazebo.*

The ROS method of describing a robot is to define its properties in URDF files. URDF files consist of definitions of different links of the robot such as the chassis, wheels, and sensors. Each link includes a description of physical properties such as the **Collision** and **Visual** components. Along with this, the definition of the **Inertial** properties, and the formation of **Joints** between different links are also defined. An example of this is shown in *Figure 3-4* below:

```
<!--.............................Left Wheel.....................................-->
<link name="left_wheel">
  <collision name="collision">
    <origin xyz="0 0 0" rpy="0 1.570796327 0"/>
    <geometry>
      <cylinder radius="0.075" length="0.03"/>
    </geometry>
  </collision>
  <visual name="left_wheel_visual">
    <origin xyz="0 0 0" rpy="0 0 0"/>
    <geometry>
      <mesh filename="package://jet_description/meshes/Left_Wheel.dae"/>
    </geometry>
  </visual>
  <inertial>
    <origin xyz="0 0 0" rpy="0 0 0"/>
    <mass value="0.75"/>
    <inertia
      ixx="3.51e-4" ixy="0.0" ixz="0.0"
      iyy="1.11e-3" iyz="0.0"
      izz="1.11e-3"/>
  </inertial>
</link>

<joint type="continuous" name="left_wheel_joint">
  <origin xyz="0.15 -0.1 0.02" rpy="0 0 0"/>
  <child link="left_wheel"/>
  <parent link="chassis"/>
  <axis xyz="1 0 0" rpy="0 0 0"/>
</joint>
```

Figure 3-4 – URDF Assembly of Left Wheel

All components of the robot follow the same structure as shown in *Figure 3-4.* To control or enable the features of these components, *Gazebo* compatible ROS plugins have been added to the existing URDF assembly of the ACES robot. These unique plugins are pre-built algorithms that give the 3D models functionality and can also tie in ROS messages and service calls.

3.4 CONTROL OF ACES ROBOT

To control the robot with a differential drive steering configuration, a *Gazebo* compatible *Differential Drive* ROS plugin has been added to the existing URDF assembly of the robot. Within this pre-built plugin, details of the robot, such as the name of the actuating joints, the separation distance between the right and left actuating wheels, and the diameter of the actuating wheels are defined. Two topics are set up for the robot: the first one receives movement commands and subscribes to the topic **cmd_vel**, while the second publishes the odometry messages to a topic called **odom**. This allows calculation of estimated robot position, based on its starting location within a given environment. Finally, two frames are set up: the first one defines the odometry frame, and the second one defines the robot's

frame, which is used as a reference for calculating odometry messages. The name of the odometry frame is **odom** and the base frame of the robot is **Main_Frame**.

The final configuration of this plugin may be seen in *Figure 3-5*:

```
<!-- DIFFERENTIAL DRIVE PLUGIN -->
<gazebo>
  <plugin name="differential_drive_controller" filename="libgazebo_ros_diff_drive.so">
    <updateRate>30</updateRate>
    <leftJoint>left_wheel_joint</leftJoint>
    <rightJoint>right_wheel_joint</rightJoint>
    <wheelSeparation>0.27</wheelSeparation>
    <wheelDiameter>0.15</wheelDiameter>
    <wheelAcceleration>20</wheelAcceleration>
    <wheelTorque>10</wheelTorque>
    <commandTopic>cmd_vel</commandTopic>
    <odometryTopic>odom</odometryTopic>
    <odometryFrame>odom</odometryFrame>
    <robotBaseFrame>Main_Frame</robotBaseFrame>
    <odometrySource>1</odometrySource>
    <publishWheelTF>false</publishWheelTF>
    <publishOdomTF>true</publishOdomTF>
    <publishTf>true</publishTf>
    <publishWheelJointState>true</publishWheelJointState>
    <legacyMode>false</legacyMode>
    <rosDebugLevel>na</rosDebugLevel>
  </plugin>
</gazebo>
```

Figure 3-5 – Differential Drive ROS Plugin Configuration

After these parameters were successfully set inside this plugin, the differential drive steering control was tested in the ACES system. The default velocity instructions were mapped to the ACES robot's **cmd_vel** topic for control using a ROS package named **teleop_twist_keyboard**.

3.5 ACES SLAM Development and Configuration

For the exploration and navigation systems to operate properly, the ACES robots must be capable of producing maps and estimating their location within these maps in real-time. As shown in the literature review laid out in *Section 2.6.2* of this report, both 2D and 3D SLAM mapping techniques are available with positives and negatives associated with each method. However, 2D SLAM is typically reliant on the use of direct depth measurement, whilst 3D SLAM utilises visual sensors. Of the available SLAM sensor types, LIDAR solutions have been identified as particularly interesting for 2D mapping as these offer unparalleled robustness and accuracy. However, the cost of LIDAR units is higher than units required for visual SLAM methods typically used in 3D mapping applications. For 3D mapping, visual-inertial and RGB-

D solutions are considered the most balanced solutions in terms of cost vs accuracy and robustness. For investigation and experimentation, both 2D and 3D slam systems have been implemented in the ACES robots.

3.5.1 QR CODE LOCALISATION

Following work carried out on a partially completed ROS project provided by the project supervisor, an attempt has been made to implement a custom QR code localisation system in *Gazebo*. This localisation method is unsuitable for use within the final design of the ACES system as it is incapable of dealing with unknown environments. However, the implementation of this system has proved a challenging and effective way of familiarising the team with the *Gazebo* software and the production of custom ROS nodes working within the simulation framework.

This localisation system relies on several QR code posters being placed at known points within the simulated environment as shown in *Figure 3-6*.

QR codes are identified using image processing and the robot's pose is estimated based on the degree of deformation of the posters as seen through the robot's camera. The code for this system is available on the project GitHub [82] as part of the final project deliverable.

This localisation system relies on several QR code posters being placed at known points within the simulated environment. QR codes are identified using image processing and the robot's pose is estimated based on the degree of deformation of the posters as seen through the robot's camera. The code for this system is available on the project GitHub [82] as part of the final project deliverable.

Figure 3-6 – QR Code Used for Early Attempt and Localisation

3.5.2 2D SLAM - GMAPPING

LIDAR is a scanning technique that uses a rotating laser emitter/receiver to calculate the distance of objects in the environment via time-of-flight calculation. Both 2D and 3D LIDAR sensors exist, however, 2D sensors are significantly cheaper and more energy-efficient. This lower cost, combined with the superb accuracy and robustness of LIDAR sensors makes them a common choice for 2D SLAM applications. When combined with an effective SLAM system, the 2D LIDAR produces a 2D occupancy grid map. For the ACES robot, a Hokuyo LIDAR sensor is modelled and placed on the ACES robot. Open-source ROS SLAM package **gmapping** then processes the LIDAR data.

3.5.2.1 LIDAR SETUP

Illustrated in *Figure 3-7,* the lines contained by the **<plugin>** tags are appended within the existing URDF file of the ACES robot to activate the scanning feature of the LIDAR sensor.

```
<plugin name="gazebo_ros_hokuyo_controller" filename="libgazebo_ros_laser.so">
  <topicName>laser/scan</topicName>
  <frameName>hokuyo</frameName>
</plugin>
```

Figure 3-7 – Activation Code of LIDAR Sensor

Once activated the ACES robot can fully utilize the features of the LIDAR sensor as visualised by the blue laser beams detecting objects in *Gazebo*, displayed in *Figure 3-8*.

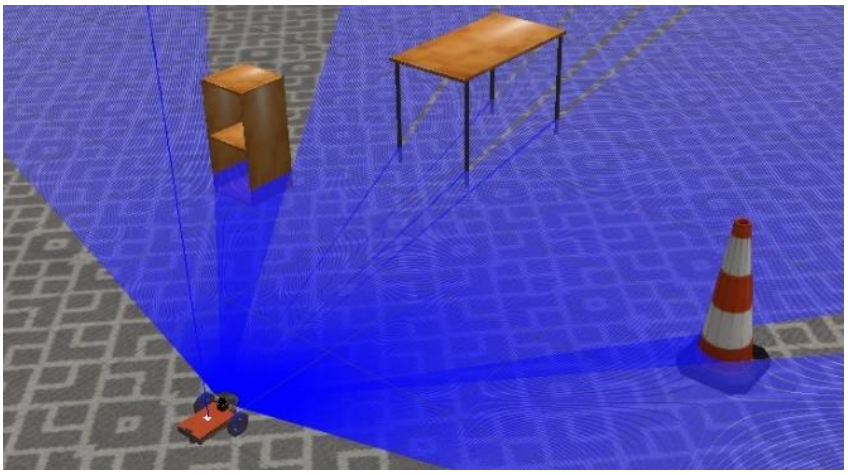


Figure 3-8 – LIDAR Scans Visualized in Gazebo

3.5.2.2 GMAPPING CONFIGURATION

With a fully functioning LIDAR sensor, the configuration requirements of ***gmapping*** package have been fully met. The configuration of this package is set as follows:

The package subscribes to the transform data **/tf** published by the ACES robots to relate frames of LIDAR sensor, robot's base, and robot's odometry for accurate localization. This package also subscribes to the laser scan topic **laser/scan** which reads the LIDAR data to create a map and publishes the created map to a topic called **map**. Once configured, this package is launched for the ACES robot and the combined output of *Gazebo* and *RViz* is retrieved, as displayed in *Figure 3-9*.

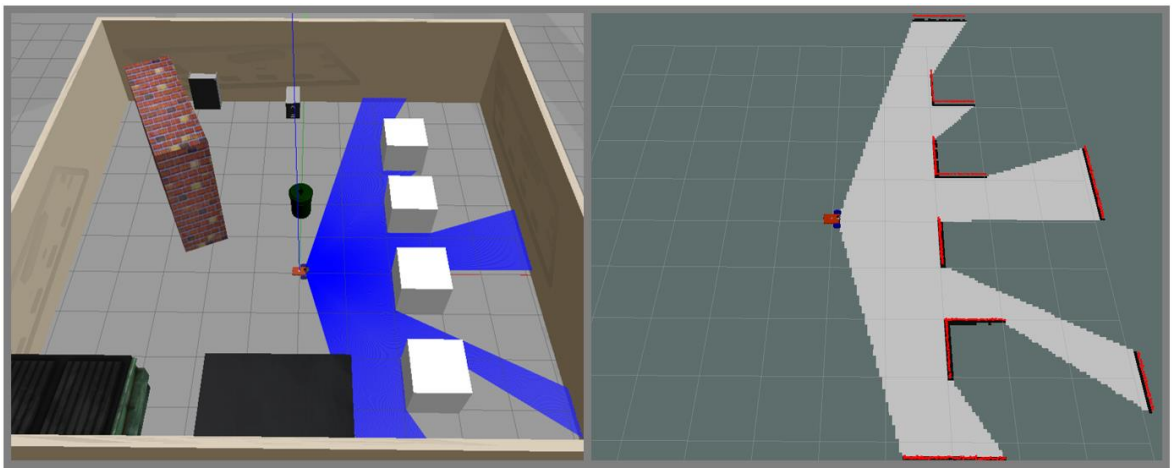


Figure 3-9 – Output of Gmapping SLAM Simulated in Gazebo & Visualized in RViz

Illustrated below in *Figure 3-10*, the transfer tree **tf** of the robot is also updated to include the ACES robots' **odometry** and a **map** published by the SLAM system.

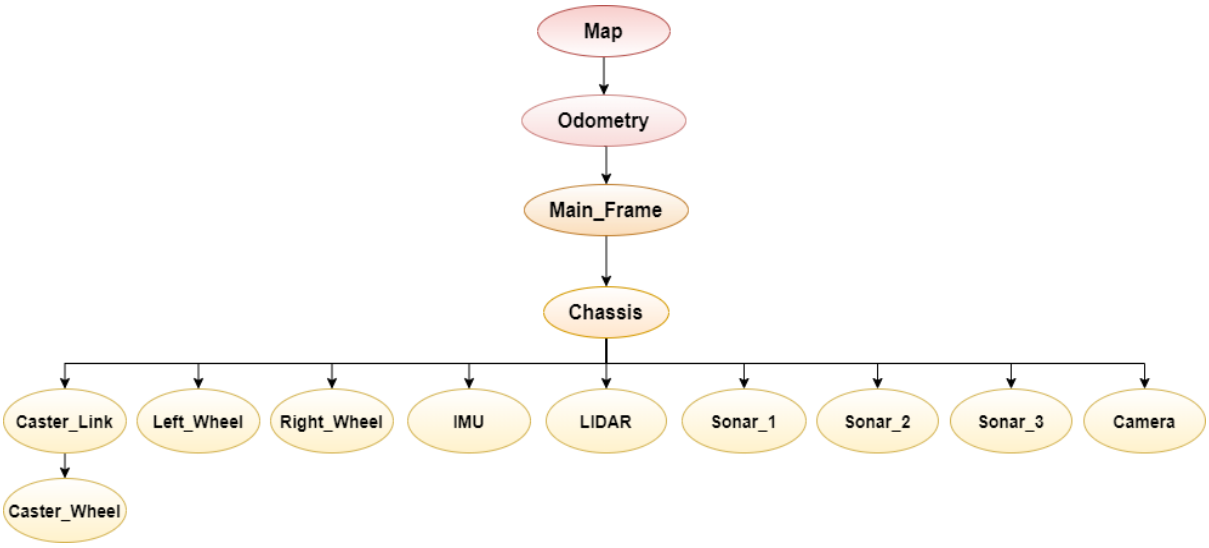


Figure 3-10 – Updated Transfer Tree of ACES Robot

3.5.3 3D SLAM - ORBSLAM

Visual-inertial and RGB-D 3D SLAM implementations offer the strongest balance between cost and performance. However, algorithms such as ORBSLAM3 [26] can perform visual-inertial SLAM using an RGB-D camera coupled to inertial measurement units. To facilitate this, a Kinect RGB-D sensor and an IMU were implemented in the simulated ACES robots.

Of the available visual-inertial and RGB-D SLAM algorithms, ORBSLAM3 boasts the highest accuracy and robustness [24] [26], as well as the capability to process sensor data from monocular, stereo, and RGB-D cameras. This method is highly versatile and able to cope with any changes to robot hardware. Although no ROS implementation of ORBSLAM3 currently exists there are several ROS wrappers available for ORBSLAM2 [83] [84]. The engineers have therefore attempted to adapt an ORBSLAM2 ROS wrapper to interact with the ORBSLAM3 code [85], also justified in that full adaption of this package constitutes a significant and useful addition to the growing ROS community. However, as the ORBSLAM3 code is currently in beta, several bugs and issues are still present at the time of testing. As the code has been developed to work with legacy versions of OpenCV and Python, all ORBSLAM3 libraries have required patching to utilise Python 3 and OpenCV 4. Once updated, it transpires that a known bug related to mismanagement of timestamps is the cause of sudden and catastrophic system failure following initialisation. Attempts to overcome this issue by the engineers have proved fruitless, and after dedicating a significant amount of development time trying to resolve this issue, the functionality of ORBSLAM3 in the ACES robots remains incomplete. Instead, ORBSLAM2 [86] has been implemented, with ORBSLAM3 recommended as an area of interest for further development with the ACES system in the future.

ORBSLAM2 uses the ORB (Oriented FAST, Rotated BREIF) [87] feature extractor to detect key points. ORB is significantly more efficient than SIFT and SURF and has similar accuracy. ORB is rotation, scale, and viewpoint invariant and robust against image noise and blur. These qualities make ORB an efficient and accurate feature extractor for visual SLAM. The output of the ORBSLAM2 GUI processing the fr1/desk sequence of the TUM RGB-D dataset may be seen in *Figure 3-11*. Matched ORB features are shown highlighted in green. The features highlighted are fairly sparse and are limited to the edges and corners of objects.

Figure 3-11 – ORB Features, extracted from TUM RGB-D Dataset sequence, processed by ORBSLAM2

ORBSLAM2 utilises a keyframe-based system to minimise processing requirements and improve overall efficiency. This system allows for tracking, local mapping, and loop closure to be performed simultaneously, as tracking is performed on every frame, but local mapping and loop closure is performed only on keyframes. A diagram illustrating ORBSLAM2 processing flow is included in *Figure 3-12*. During the Tracking phase, ORB features are extracted from every frame. These features are then compared to those extracted from the previous frame, with camera motion estimated based on the movement of matched features between frames. Motion estimates are then improved through bundle adjustment. The current frame features are then compared to the co-visibility map of all previous keyframes, and camera motion and pose is further optimised via bundle adjustment. If the current frame meets certain criteria based on feature quantity and quality, then it can be made a keyframe and its features added to the co-visibility map.

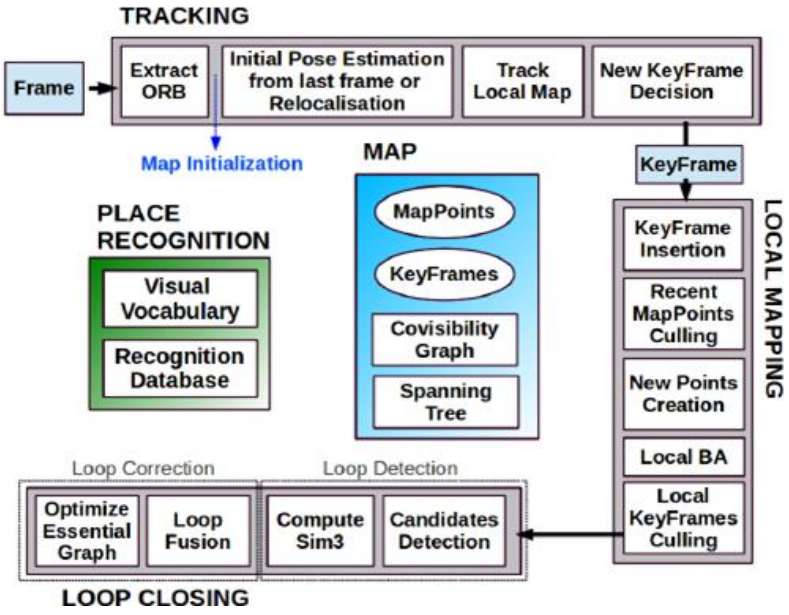


Figure 3-12 – ORBSLAM2 System Diagram

The Local Mapping phase processes the most recently produced keyframe and compares its features to the co-visibility map. When features are matched between key frames, the estimated camera motion between frames is used to calculate the distance of the feature from the camera. In this manner, a local map is produced incorporating features in 3D space as well as camera pose. To ensure efficiency and accuracy, features that do not match between keyframes are deleted, as are keyframes with a low overall matching rate.

The Loop Closing phase then processes the most recent keyframe to perform place recognition. If the features in the current keyframe match very heavily with those of a previous keyframe then it is assumed that the camera is in the same location. When a loop has detected the translation of the features between the current and matching key frame is calculated to estimate the actual camera position. Using this method, the accumulated error may be calculated, and the map recursively adjusted via pose graph estimation to reduce error across the loop.

The ORBSLAM2 ROS wrapper used in the ACES system is developed and maintained by the Applied AI Initiative [84]. This wrapper implementation allows the ORBSLAM2 algorithm and libraries to be accessed as ROS nodes with associated ROS services. The RGB-D node in this wrapper subscribes to an RGB image topic, a depth image topic, and a 'camera_info' topic, publishing point clouds in real-time. This ROS wrapper has been highlighted as particularly desirable due to its compatibility with Python 3 and OpenCV 4, and so no manual upgrading was necessary to ensure compatibility with the rest of the ACES system. As well as this, the camera configuration parameters such as camera resolution, distortion, focal length, etc could be retrieved from a 'camera_info' topic such as those published by default within *Gazebo*. Retrieving camera parameters from 'camera_info' topics on system initialisation, rather than hard-coding these parameters into the node configuration allows for increased flexibility and versatility concerning hardware changes. Using this method, only the camera parameters within *Gazebo* need to be changed, and the ORBSLAM system adapts dynamically. These features reduced implementation complexity and the risk of human error during system configuration.

3.5.3.1 3D TO 2D CONVERSION USING OCTOMAP SERVER

As the navigation and exploration system used in the ACES design relies on occupancy grids, an **OctoMap** server node is used to produce a down-projected occupancy grid from the SLAM point cloud.

OctoMap is an open-source framework for producing volumetric 3D maps [88]. OctoMap Server [89] is a pre-developed ROS package that allows the OctoMap C++ library to be used via ROS. The OctoMap Server node is configured to accept point clouds published in the standardised PointcCoud2 ROS message format. The node then uses the OctoMap library to convert the point cloud to a volumetric cube-based map. This is desirable as OctoMaps are more visually intelligible and in some ways more useful than point cloud maps as they are denser and contain information about explored, unexplored and occupied areas. A comparison between a point cloud and OctoMap is provided in *Figure 3-13*. The OctoMap server node is configured to publish a 2D occupancy grid map via a down projection of the OctoMap within a defined range of heights defined by the z-axis magnitude. This occupancy grid is then usable by the navigation and exploration systems within ACES.

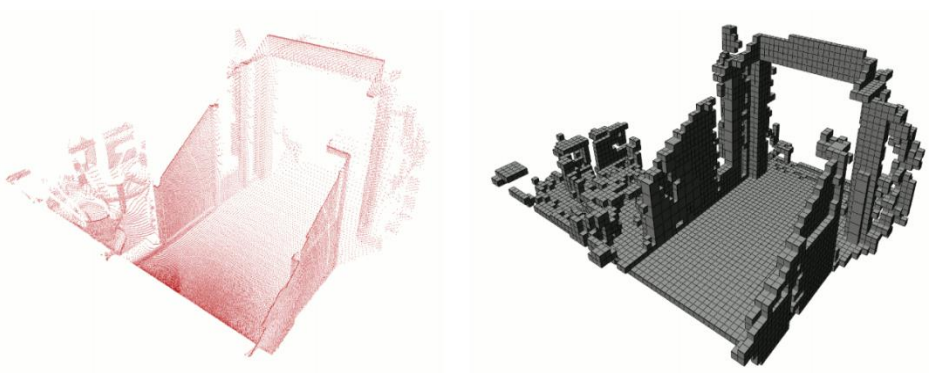


Figure 3-13 – Point cloud (left) and OctoMap (right) representation of a Doorway [88]

The OctoMap library produces a volumetric 3D map utilising Octrees [90] [91]. Octrees are a hierarchical tree structure for dividing 3D space. Each node in the hierarchy represents a cubic volume of space called a voxel. With each layer of the Octree, voxels are recursively divided into eight equality sized smaller voxels [88]. Subdivision continues until a minimum voxel size is reached. This process may be seen in *Figure 3-14*. This method provides efficient mapping, as only voxels which are potentially occupied need to be sub-divided. As well as this, the probabilistic method of occupancy recording allows the map to be quickly updated to account for new sensor data.

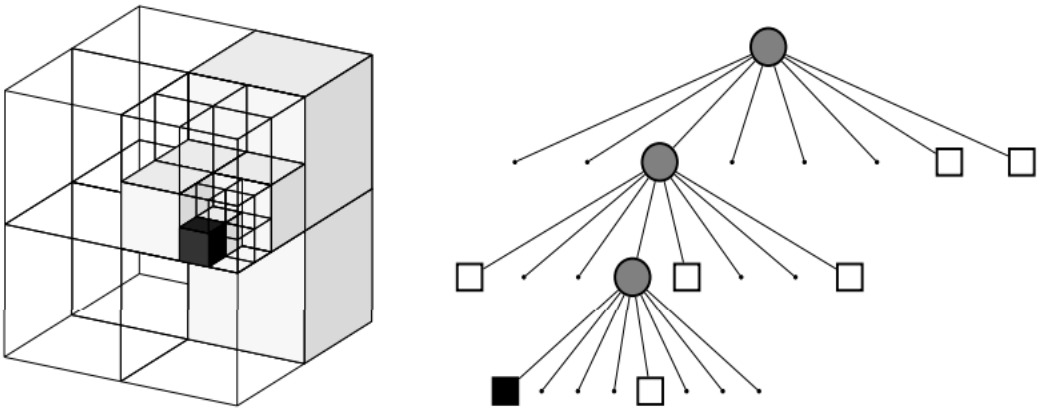


Figure 3-14 – Octree Hierarchy and Volumetric Visualisation [88]

3.6 AUTONOMOUS NAVIGATION DEVELOPMENT AND CONFIGURATION

The main aim of autonomous navigation is to move a robot from the start position to the goal position without making any collision with the environment. Within ROS, an open-source package called **navigation** is included with an implementation of several navigation-related algorithms such as **SLAM, A *(star), Dijkstra,** and **AMCL** can be directly installed for use by the ACES system to enable autonomous navigation.

3.6.1 THE NAVIGATION STACK

The ROS **navigation** package or **navigation stack** receives information from robot odometry data via sensors such as wheel encoders, IMUs, and GPS, along with sensor data streams such as laser scanner data and a 3D point cloud from sensors such as the Kinect sensor. The end goal position is determined by the user through manual entry [55] or provided by an autonomous exploration package as required by the project objectives.

Accordingly, the differential drive plugin implemented for the ACES robot is publishing the odometry data of the robot, the LIDAR sensor is providing the sensor data for obstacle avoidance, and the visualizing software *RViz* is used to provide the end goal position to the robot. Once the end goal is set, the navigation stack sends velocity commands or movement commands to the **cmd_vel** topic of the ACES robot, which translates these commands to turn the right and left actuating wheels until the robot has reached the desired end goal. To successfully configure the ACES robot to use the navigation stack, the structure shown in *Figure 3-15* is considered.

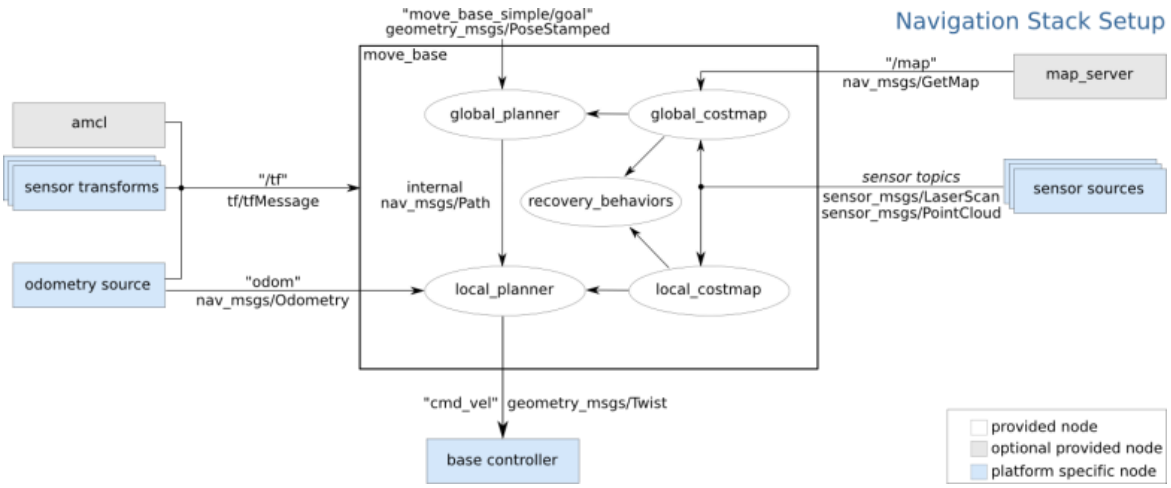


Figure 3-15 – Layout of ROS Navigation Stack [92]

3.6.2 MOVE_BASE COMMANDS

At the core of the navigation stack is the **move_base** node. The main purpose of the node is to provide movement instructions that direct the base of a robot to the end goal position within a user-specified tolerance. This node can also perform recovery behaviours when the robot perceives itself as stuck. For **move_base** to work properly, a layered structure of global and a local costmap is required. Both costmaps are overlaid onto the main map produced by the SLAM system as 2D layered grids and used to store information about obstacles. The global costmap is used for creating long-term plans over the entire environment by using global planners. Whereas the local costmap uses local planners to produce movement plans for the robot that avoid detected obstacles. Effective configuration of the **move_base** node for the ACES robot requires setting up each of the following parameters.

3.6.2.1 COMMON COSTMAP

The **move_base** node uses the **common costmap** to guide the local and global costmaps to the appropriate sensor topics for updates. The **common costmap** also specifies other parameters such as the obstacle detection range, the robot's footprint, the names of the observation sources and associated topics, the plugins needed for obstacle detection and avoidance, and the inflation radius which indicates the maximum distance from obstacles. These parameters are listed in *Figure 3-16* below:

```
obstacle_range: 2.5
raytrace_range: 3.0
publish_voxel_map: false
transform_tolerance: 0.5
meter_scoring: true
footprint: [[-0.12, -0.075], [-0.12, 0.075], [0.12, 0.09], [0.12, -0.09]]
footprint_padding: 0.1
plugins:
- {name: obstacles_layer, type: "costmap_2d::ObstacleLayer"}
- {name: inflater_layer, type: "costmap_2d::InflationLayer"}
obstacles_layer:
  observation_sources: scan
  scan: {sensor_frame: hokuyo, data_type: LaserScan, topic: laser/scan,
         marking: true, clearing: true, obstacle_range: 4.0, raytrace_range: 3.5}
  track_unknown_space: true
inflater_layer:
  enabled:              true
  inflation_radius:     0.01
  cost_scaling_factor:  0.1
```

Figure 3-16 – Configuration of Common Costmap

3.6.2.2 LOCAL & GLOBAL COSTMAPS

The key difference between local and global costmaps is the **global_frame** parameter which defines which coordinate frame the costmap should run in. The **global_frame** for both costmaps was set to the **map** frame for the ACES robot, indicating that both costmaps will be placed on the map created by the SLAM system. Furthermore, both costmaps necessitate the identification of the **robot_base_frame**: this parameter specifies the coordinate frame that the costmap should use for the robot's base. As the robot's base frame is named **Main_Frame**, the **robot_base_frame** parameter has been labelled accordingly.

The **rolling_window** setting in the local costmap is set to true and set to false in the global costmap settings, indicating that the local costmap will remain centred around the robot as it traverses the simulated environment, while the global costmap will remain static. Configuration of the costmaps is demonstrated in *Figure 3-17*.

```
local_costmap:
   global_frame: map
   robot_base_frame: Main_Frame
   update_frequency: 20.0
   publish_frequency: 5.0
   width: 10.0
   height: 10.0
   resolution: 0.05
   rolling_window: true
```

```
global_costmap:
   global_frame: map
   robot_base_frame: Main_Frame
   update_frequency: 20.0
   publish_frequency: 5.0
   width: 40.0
   height: 40.0
   resolution: 0.05
   rolling_window: false
```

Figure 3-17 – Configuration of Local & Global Costmap

3.6.2.3 GLOBAL & LOCAL PLANNERS

From the limited options available, **Navfn** [93] has been selected as the global planner for implementation in the ACES system. This planner employs Dijkstra's algorithm to determine a global path with the lowest cost between start and end points, representing the shortest distance between the two points for maximum efficiency.

The **Trajectory planner** [94] was selected as the local planner. The planner generates kinematic trajectories for the robot to follow to get from one position to the next. Using the local costmap the rolling window around the robot computes a cost value for each of the map grids represented within the area of this rolling window. This value represents the costs of traversing the grid cells, i.e., deciding if the grids represented by the global path are free of obstacles. The value function is also used to determine which movement commands are sent to the robot's base frame (the **cmd_vel** topic).

3.6.2.4 AMCL

A probabilistic localization technique known as **amcl** [95] is used to aid the robot's localization process. This technique outputs position estimates using the 2D map produced by the SLAM system, LIDAR scans, and transform messages **tf**. The accuracy of localization improves as the robot moves around within the map, as seen by a cluster of misaligned red arrows in *Figure 3-18 (a)* to only a few directed red arrows in *Figure 3-18 (b)*

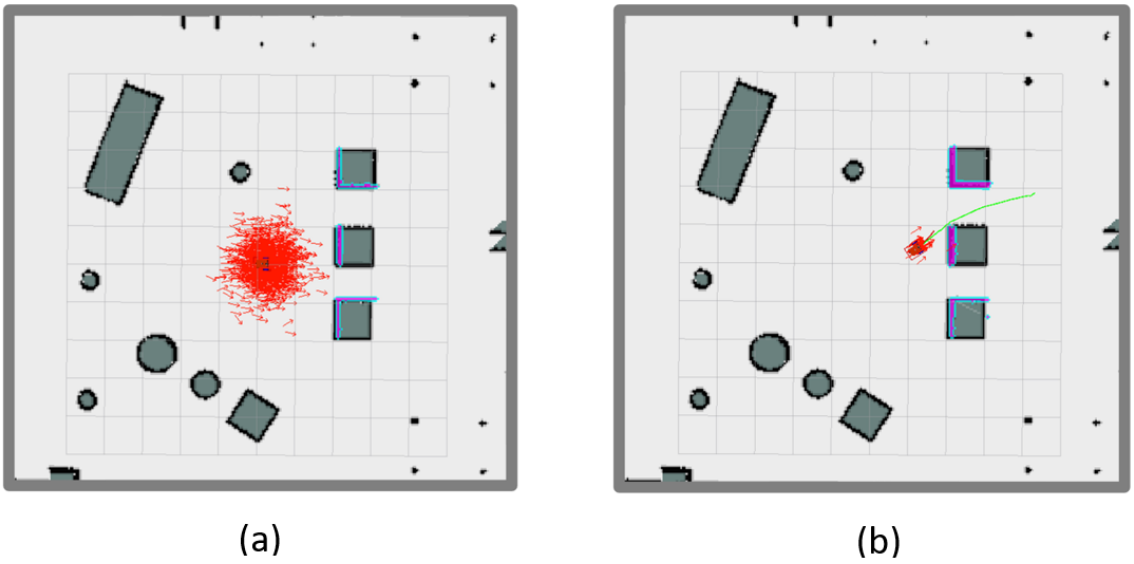


Figure 3-18 – AMCL Operation (a) Initial Pose Estimates (b) Pose Estimates After Moving

The ACES robot was launched in a known map, and the output visualized in *Figure 3-19* was achieved. Upon receiving an end goal, a global plan (shown in green) is created for the robot from its starting position to the end goal position and using the local plan (shown in red), the robot follows the global plan, detecting and avoiding obstacles for safe navigation.

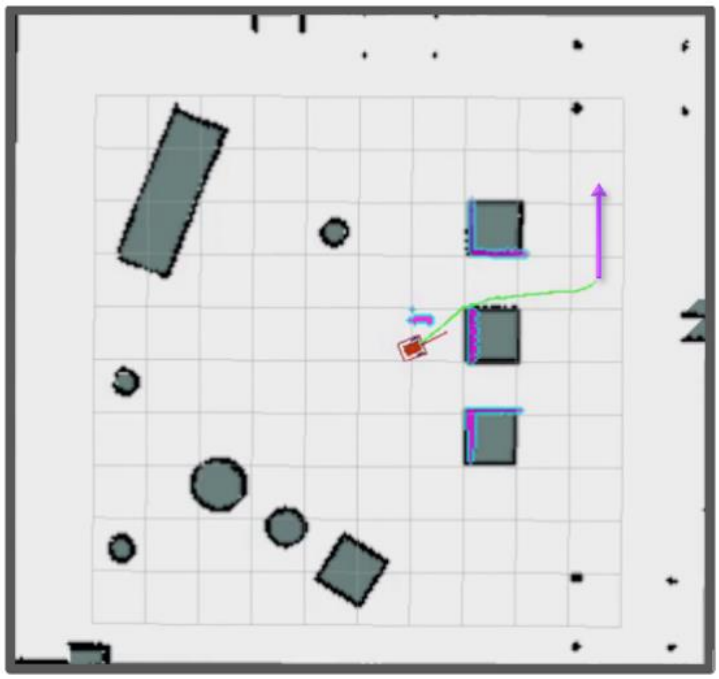


Figure 3-19 – Visualizing the Operation of the Navigation Stack in RViz

3.7 AUTONOMOUS EXPLORATION DEVELOPMENT AND CONFIGURATION

Autonomous exploration aims to discover an individual robot's most desired location goal in a series that offers as much information about the environment as possible. As described in the literature review, frontier-based exploration distinguishes open space from the uncharted territory by designating uncharted territory as free or occupied as robots approach the frontiers and pass through them into new territory. This process is repeated until all frontiers have been explored, requiring the existence of occupied cells in the free area.

To determine the exploration of unknown areas in ROS, open-source package **explore_lite** is used. The package includes the **explore** node that uses the ACES robot's SLAM system to generate occupancy grid frontiers and publishes the target positions to the **move_base** node. Instead of requiring the user to define an end target, the **explore_lite** algorithm searches for and selects unexplored frontiers in the environment and updates the **move_base** node subscribed to by the path planner to submit movement commands to the robot's base.

Configuration of explore node for ACES robot may be seen in *Figure 3-20*.

```
<!-- Explore Lite Node -->
<node pkg="explore_lite" type="explore" respawn="true" name="explore" output="screen">
  <param name="robot_base_frame" value="Main_Frame"/>
  <param name="costmap_topic" value="map"/>
  <param name="costmap_updates_topic" value="map_updates"/>
  <param name="visualize" value="true"/>
  <param name="planner_frequency" value="0.33"/>
  <param name="progress_timeout" value="30.0"/>
  <param name="potential_scale" value="3.0"/>
  <param name="orientation_scale" value="0.0"/>
  <param name="gain_scale" value="1.0"/>
  <param name="transform_tolerance" value="0.3"/>
  <param name="min_frontier_size" value="0.75"/>
</node>
```

Figure 3-20 – Configuration of Explore Lite Package

By specifying the **robot_base_frame** and **costmap_topic**, the explore node distinguishes the position of the ACES robot on the map and uses the occupancy grid produced by the map to locate frontiers. Upon locating a frontier on the map supplied by the SLAM system, the ACES robot is sent movement commands. Based on the movement commands, once the robot has reached the allocated frontier, the map is updated by the SLAM system and is fed back to the **explore_lite** package for it to find new frontiers. This process keeps repeating, until no frontiers are found, or a map has been fully completed.

The successful implementation of **explore_lite** within the ACES system is illustrated in *Figure 3-21* on the following page, demonstrating the operation of the ACES robot exploring an unknown environment.

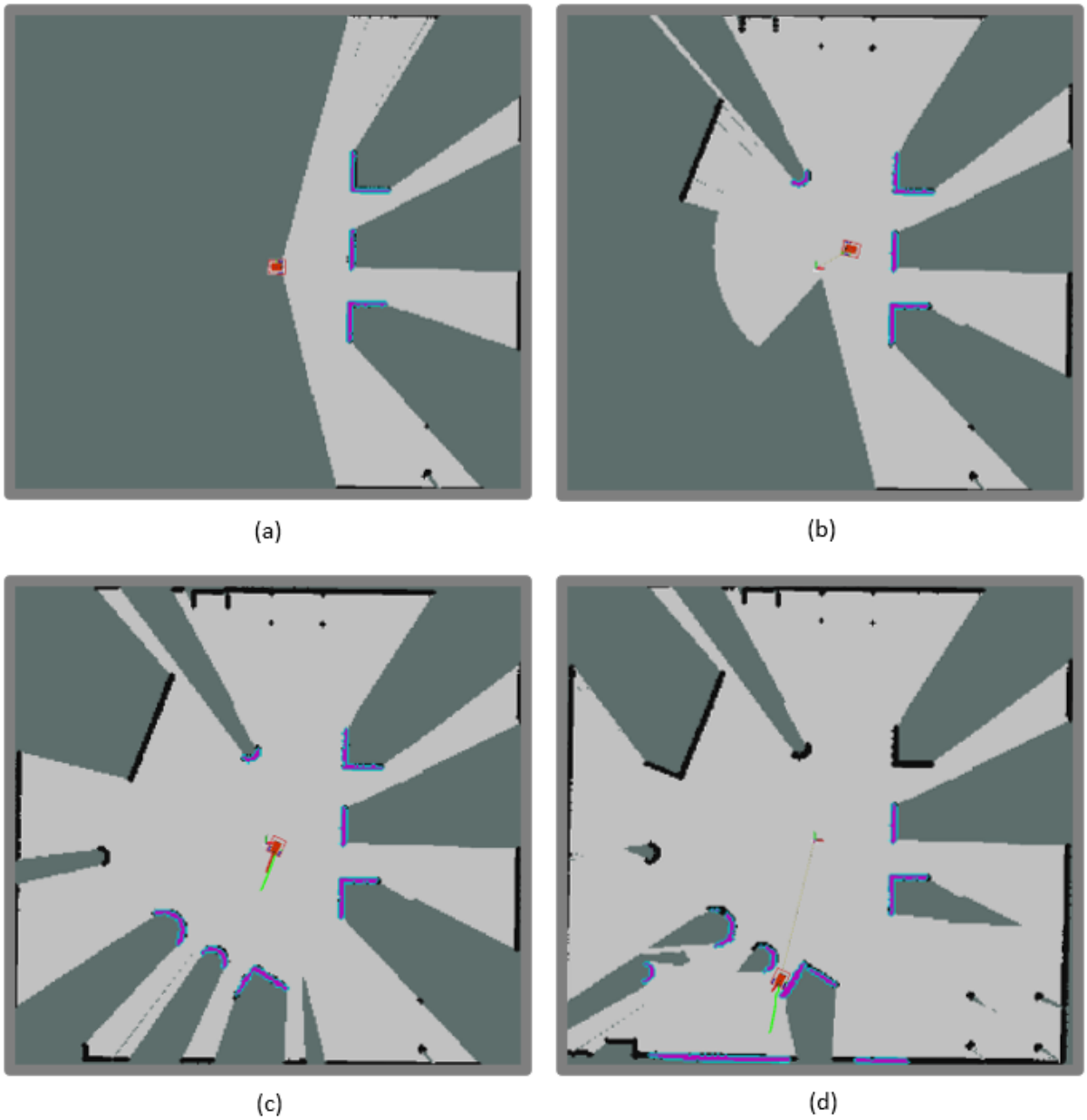


Figure 3-21 – Operation of explore_lite with ACES robot: (a) Initial Position (b) 2nd Position (c) 3rd Position (d) 4th Position

In the first position, the simulated ACES robot is launched in an unknown test map and searches for new frontiers. Upon detection, the unit's navigation process sets a route to the nearest suitable frontier as reflected by the second position. The map is updated, and the same process repeats as shown by the robot's third and fourth positions, until no remaining unknown areas remain.

3.8 MULTI-ROBOT DEVELOPMENT AND CONFIGURATION

To simulate the operation of multiple robots in *Gazebo*, the description, topics, nodes, and frames of the ACES robot must be shared among **n** robots, where **n** can be any rational number.

Two concepts - **namespace** and **tf prefix** - are implemented to accomplish this. Both concepts are necessary to ensure that the multi-robot configuration function properly. Thus, each robot must operate in its namespace within a shared simulation environment. In the current configuration of the ACES robot, nodes, topics, and frames produced by the robot resemble the layout shown in *Figure 3-22(a).* This layout works with one robot, however, if another robot is introduced into the simulation, a transfer tree conflict will be issued.

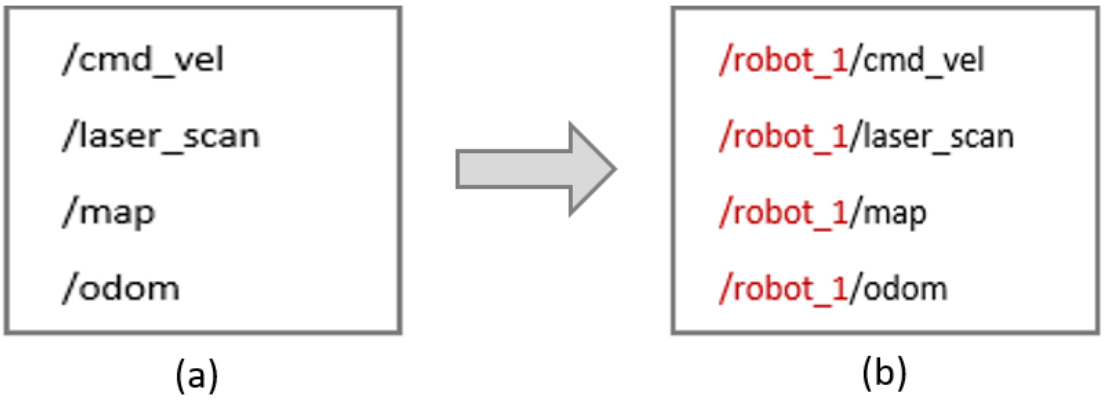


Figure 3-22 – (a) Example Topics, Nodes & Frames of ACES Robot (b) Example Namespace Implementation

To resolve this, the concept of a namespace is implemented as illustrated in *Figure 3-22(b)*. Using this implementation will keep the name of the topics, nodes, and frames the same, but will introduce a namespace for each of the robots in front of these parameters. Once the namespace is introduced, the transfer tree conflict is resolved and the robots can share the ACES robot description, along with its nodes, topics, and frames.

3.8.1 CONFIGURATION METHOD

The process of configuring the **namespaces** and **tf_prefix** has required the creation of a series of launch files that enable multiple robots to be added into the *Gazebo* simulation, with each robot using separate namespaces and a conflict-free transfer tree.

The first launch file introduces two arguments, specifying the **robot_name**, and the second one specifies the initial pose with **init_pose**, determining the spawning coordinates of the ACES robots within the simulated environment.

To share the description of the ACES robot, a '**/**' is added when calling the ACES robot URDF file. The use of '/' within this setup fully qualifies the URDF description file to be shared. The **robot_state_publisher** and the **joint_state_publisher** nodes are initialized, publishing the relationship between different frames, nodes, topics, joints, etc. The ACES robot's description declaration launch file may be seen in *Figure 3-23*.

```
<launch>

  <arg name="robot_name"/>
  <arg name="init_pose"/>

  <node name="spawn_model" pkg="gazebo_ros" type="spawn_model"
        args="$(arg init_pose) -urdf -param /robot_description -model $(arg robot_name)"
        respawn="false" output="screen" />

  <!-- ROBOT STATE -->
  <node pkg="robot_state_publisher" type="robot_state_publisher" name="robot_state_publisher">
    <param name="publish_frequency" type="double" value="50.0" />
  </node>

  <!-- JOINT STATE -->
  <node name="joint_state_publisher" pkg="joint_state_publisher" type="joint_state_publisher">
    <param name="use_gui"            value="False"/>
    <param name="robot_description"  textfile= "$(find jet_description)/urdf/multi_robot.xacro" />
  </node>

</launch>
```

Figure 3-23 – ACES Robot Description Declaration for Multi-Robot Configuration

To launch multiple robots, the robot description specified above is called and shared with each of the robots within their individual **<group>** tags, in which the robot **namespace**, **robot_name**, **init_pose,** and **tf_prefix** is identified.

This methodology of creating separate namespaces was applied to the SLAM, navigation, and exploration packages, enabling multiple robots to launch exploring an unknown environment and creating a map of that environment.

This may be seen in *Figure 3-24* on the following page.

```
<!-- ..........................BEGIN ROBOT 1.....................................-->
<group ns="robot_1">
  <param name="tf_prefix" value="robot_1" />
  <include file="$(find multi_robot)/launch/jet_setup.launch" >
    <arg name="init_pose" value="-x -9 -y 8 -z 0 -R 0 -P 0 -Y 0" />
    <arg name="robot_name"  value="robot_1" />
  </include>
  <!-- Gmapping Node -->
  <node pkg="gmapping" type="slam_gmapping" name="slam_gmapping" output="screen">
    <param name="base_frame" value="/robot_1/Main_Frame"/>
    <param name="odom_frame" value="/robot_1/odom"/>
    <param name="map_frame"  value="/robot_1/map"/>
    <remap from="scan" to="/robot_1/laser/scan"/>
    <remap from="map" to="/robot_1/map" />
    <rosparam command="load" file="$(find jet_slam)/config/gmapping_params.yaml" />
  </node>
  <!-- MoveBase Node -->
  <include file="$(find multi_robot)/config/move_base_setup.launch">
    <arg name="namespace" value="robot_1"/>
  </include>
  <!-- Explore-Lite Node -->
  <include file="$(find multi_robot)/config/exploration_setup.launch">
    <arg name="namespace" value="robot_1"/>
  </include>
</group>
```

Figure 3-24 – Setup of namespace & tf_prefix for Multi-Robot Simulation

Once properly integrated, simulation with three robots was launched into *Gazebo,* and upon calling 'rostopic list' in the *Ubuntu* terminal, the output shown in *Figure 3-25* is seen.

Appendix D may be referred to for more detail.

/robot_1/cmd_vel	/robot_2/cmd_vel	/robot_3/cmd_vel
/robot_1/map	/robot_2/map	/robot_3/map
/robot_1/odom	/robot_2/odom	/robot_3/odom
/robot_1/laser/scan	/robot_2/laser/scan	/robot_3/laser/scan

Figure 3-25 – Nodes, Topics & Frames for Three Robots Simulated in Gazebo

3.9 MAP-MERGER DEVELOPMENT AND CONFIGURATION

Further development has been split into 2D and 3D ACES systems, with the 2D system primarily using the 2D gmapping SLAM described in *Section 3.5.2* which stitches together the occupancy grids provided by each robot to generate one main map. The 3D system works by combining 3D point clouds produced by ORBSLAM2 system stated in *Section 3.5.3*. Using a 3D map merger node, the point clouds from multiple robots are combined to create one main point cloud. The combined point cloud is transformed into a 3D map using OctoMapping.

3.9.1 2D MAP-MERGER

The 2D ACES system shown in *Figure 3-26* uses the hardware, SLAM, exploration, and navigation systems discussed in *Sections 3.1* through *3.8* of this report. Configured as a multi-robot system a Map Merger node is included to perform global map-merging of the individual robot maps. The maps produced are 2D occupancy grids.

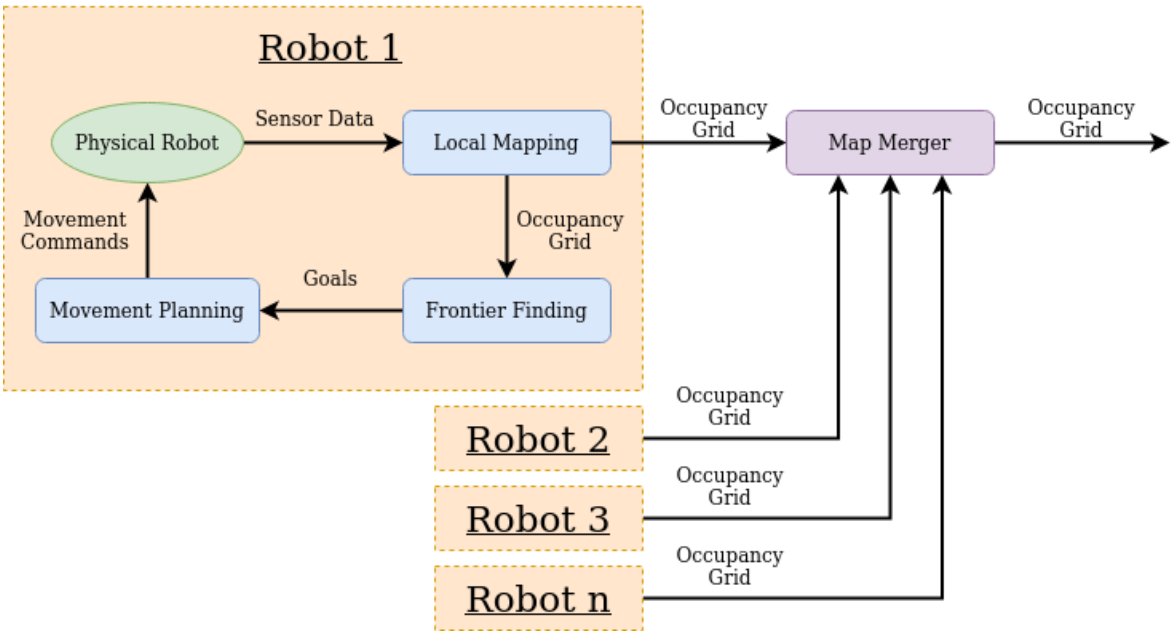


Figure 3-26 – Layout of 2D ACES System

The 2D Map Merger node [96] allows 2D occupancy grids to be stitched together to produce a global map using pair-wise matching and feature detection algorithms.

The algorithm parameters are shown in *Figure 3-27* below:

```
<!-- Map Merge -->
<node pkg="multirobot_map_merge" type="map_merge"
      respawn="false" name="map_merge" output="screen">
  <param name="robot_map_topic" value="map"/>
  <param name="robot_map_updates_topic" value="map_updates"/>
  <param name="robot_namespace" value="robot_"/>
  <param name="merged_map_topic" value="map"/>
  <param name="world_frame" value="/map"/>
  <param name="known_init_poses" value="false"/>
  <param name="merging_rate" value="4.0"/>
  <param name="discovery_rate" value="0.05"/>
  <param name="estimation_rate" value="0.5"/>
  <param name="publish_rate" value="1.0"/>
  <param name="estimation_confidence" value="0.25"/>
</node>
```

Figure 3-27 – 2D Map Merging Configuration for Multi-Robot Simulation

3.9.1.1 STITCHING ALGORITHM

The stitching pipeline is designed using OpenCV image processing tools. Occupancy grids are 'stitched' together and the transformation between them is estimated. A range of 0-100 is assigned to each cell in the occupancy grids, indicating how likely it is for an obstacle to be present, and -1 is used to determine an unknown probability. This representation is also used by ROS and can be easily mapped as a grayscale image, which is why the computer vision approach was chosen.

3.9.1.2 FEATURE DETECTION

The 2D map merger node uses SIFT features, as it is a common method of feature detection when applying the stitching algorithm. As the map merger node uses occupancy grids, downsizing of the images is not required. This is because occupancy grids are inherently less pixel-heavy than camera images. Since there are fewer features in the picture, feature matching on occupancy grids is usually rather accurate and effective. The algorithm only needs to change the robots' starting positions so that the stitching frequency can be reduced. This reduces the calculation time, allowing the algorithm to reuse previous transformations without reducing the accuracy. As a result, the estimate transforms only shift during the initial state; however, if an overlap occurs, the transforms can be calculated with greater precision.

3.9.1.3 FINAL TRANSFORMATION

The algorithm's final step calculates the transform of each robot's locally generated map in relation to the global reference frame. This allows the algorithm to choose one of the local maps as the global reference frame into which other local maps are combined. However, since there may be multiple paths to multiple maps, this approach does not completely solve the problem, and the algorithm may take additional steps. To break up the multiple loops, one of these steps is to build a maximum spanning tree. After that, the map's edges are weighted with inliers that prefer stronger matches, and spanning trees are used to find the final transform and choose it as the reference to the remaining maps.

3.9.2 3D MAP-MERGER

The navigation and exploration nodes in the 3D ACES system are the same as in the 2D system; however, the local mapping node in the 3D SLAM system outputs point clouds in the PointCloud2 message format. Therefore, the 3D system's map-merger method differs from

the 2D system, as different techniques are needed to align point clouds versus occupancy grids.

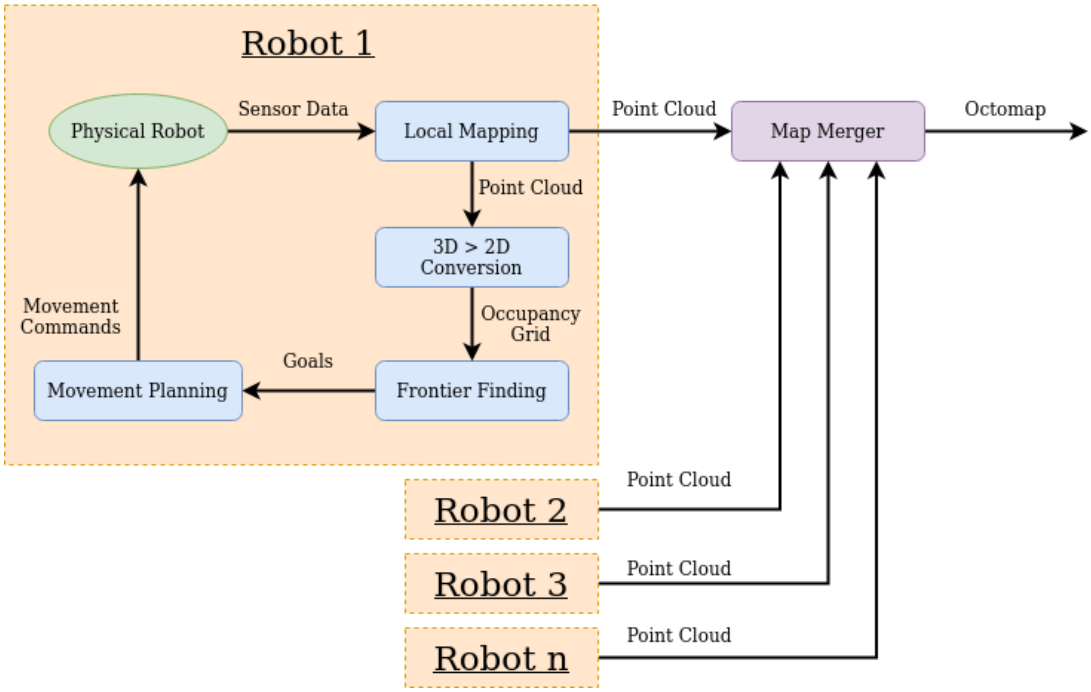


Figure 3-28 – Layout of 3D ACES System

In the 3D ACES system, seen in *Figure 3-28*, a single map merger node is used to process point clouds from all robots and merge them into a single global map. This global map is further processed via an OctoMap server to make the resultant global map more visually intelligible and usable.

The algorithm parameters are shown in Figure 3-29 below:

```
<node pkg="map_merge_3d" type="map_merge_node"
      respawn="false" name="map_merge_3d" output="screen">
  <param name="robot_map_topic" value="map"/>
  <param name="robot_namespace" value=""/>
  <param name="merged_map_topic" value="map"/>
  <param name="world_frame" value="world"/>
  <param name="compositing_rate" value="0.3"/>
  <param name="discovery_rate" value="0.05"/>
  <param name="estimation_rate" value="0.01"/>
  <param name="publish_tf" value="true"/>
</node>
```

Figure 3-29 – 3D Map Merging Configuration for Multi-Robot Simulation

Once the map-merging processing had been completed a visual output of the map is presented in *RViz*. Using a single master configuration means that the system becomes

centralised, therefore local communications are needed to send the individual local maps from the ACES units to the merging node.

3.9.2.1 MAP MERGER ALGORITHM

The Map Merger algorithm is based on a featured matching approach where the algorithm exclusively uses point cloud data representing the maps as an input. Input data also includes RGB information from the point clouds. The algorithm performs a pair-wise transformation using only the geographic data and the RGB. This is achieved through down-sampling and filtering of outliers before an estimation can be made for the point cloud.

Using SIFT detection, the key points are detected and extracted, reducing the overall points required for processing. Using the descriptor method, the key points between the maps being merged are assigned descriptors. Once the descriptors are assigned, the key points between the maps are then matched allowing for estimation of initial rigid transformations. With the matching process completed, refinement of the transformation is carried out, utilising all the points in the filtered point cloud. Using the minimisation method on the filtered point clouds further refines the transform.

For the map merging algorithm to be capable of stitching together more than two maps, a merging graph must be considered. The merging graph has nodes that correspond to individual robot maps with the edges of the generated map are represented by a pair-wise transformation estimate. To construct the map merging graph for the required number of robots launched, the pair-wise transforms must also be estimated.

The resulting map can be either dense or sparse: this is down to the map missing some edges which is expected as some of the transforms cannot be estimated. Once the graph has been constructed the global map can be produced. To compute the global map, the nodes must undergo a spatial configuration allowing for a transformation of each map from the global reference frame to the map that is constant with the pairwise transform.

3.9.2.2 MERGE NODE

The **map_merge_3D** node [97] handles the merging of the point clouds produced from the robot's SLAM system. Once the SLAM system produces point cloud data the data is sent to the map merger node. This node estimates the transformations between the maps using the

map merging algorithm. The algorithm works with only the point clouds and operates using feature matching. Once the transformations are estimated the maps are stitched together to create a global map. The ROS node requires significant computational power to merge multiple maps especially if they are complex or use high resolution. Therefore, the map_merge_3D is of asynchronous architecture, allowing for the merged map to be updated at a high rate. The map-merger node does not need to produce new estimations every time the map is updated as it can use previous estimates to update the map. handles the merging of the point clouds produced from the robot's SLAM system. Once the SLAM system produces point cloud data the data is sent to the map merger node. This node estimates the transformations between the maps using the map merging algorithm. The algorithm works with only the point clouds and operates using feature matching. Once the transformations are estimated the maps are stitched together to create a global map. The ROS node requires significant computational power to merge multiple maps especially if they are complex or use high resolution. Therefore, the **map_merge_3D** is of asynchronous architecture, allowing for the merged map to be updated at a high rate. The map-merger node does not need to produce new estimations every time the map is updated as it can use previous estimates to update the map.

The map merging node automatically scans the ROS graph to find available map topics. This allows for the map merger to automatically handle an arbitrary number of robots, allowing for n robots to be added to the multi-robot layer.

3.10 FINAL DELIVERABLE: ACES PACKAGE ON GITHUB

The final deliverable of the ACES project is the ACES software distributed via the team's GitHub repository [82]. Users are introduced to the project, its structure and operation, as well as installation and usage instructions, through the main readme file. Examples of the ACES simulation outputs are also presented to allow users to verify their results.

Users are directed to clone the ACES repository into the source folder of their catkin workspace and run the ACES install script [98] to facilitate and automate the installation of the ACES system and simulations in Ubuntu 18.04 OS. This installation script installs the primary ROS nodes: orb_slam_2_rgbd [84], octomap_server [89], explore_lite [68], move_base [68], and map_merge_node [97], as well as all system dependencies.

To ensure that all third-party ROS packages remain fully up to date these packages are not directly included in the ACES repository. Instead, these are installed individually from their respective sources when running the installation script. This approach ensures that any updates to the incorporated ROS packages are automatically included in the ACES system. This also keeps the ACES repository as lightweight as possible with relatively few files and folders, as demonstrated in figure 3-30.

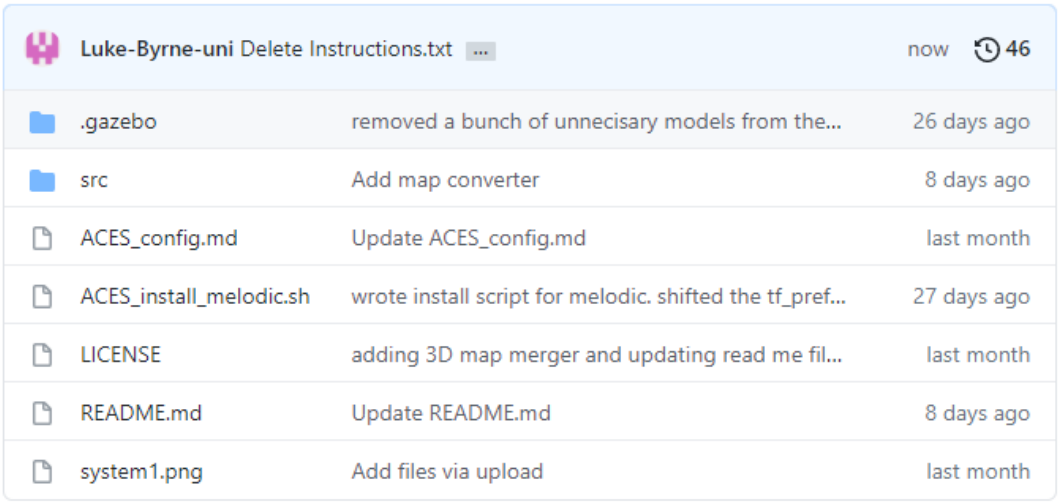


Figure 3-30 – ACES GitHub Files/Folders

Due to this, the main python scripts used to process experimental results make up the bulk of the repository by file size, as may be seen in figure 3-31. An exhaustive list and description of all configurable parameters is provided in the ACES_config.md referred to in the main readme. This document details the methods by which all component ROS nodes may be configured, as well as the methods for executing all available ROS services.

Figure 3-30 – ACES GitHub Files/Folders

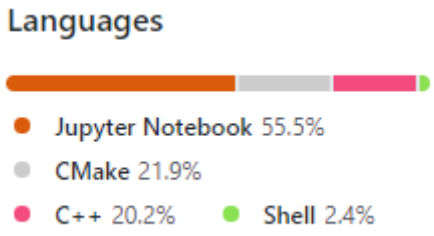


Figure 3-31 – Project Programming Languages Breakdown

Chapter 4 DESIGN OF EXPERIMENTS

4.1 DESIGN OF EXPERIMENTS

Although it was originally planned for each experiment to be entirely factorial in nature, time and processing constraints made it difficult to achieve that goal while restricting the depth and complexity of the experiments. Nonetheless, to minimise labour requirements and increase reliability, the majority of testing was automated using bash scripts, with tests run several times where possible to reduce the effect of uncontrollable external influences and improve accuracy.

4.1.1 OBJECTIVES OF THE EXPERIMENTS

To validate the system, ensure functionality, and investigate ACES strengths and weaknesses, aims and objectives of the testing were defined:

- Validate ACES functionality.
- Assess the accuracy of the mapping.
- Evaluate the scalability of the system.
- Measure the effects of different velocities on ACES performance.

4.1.2 TESTING ENVIRONMENT AND ENVIRONMENTAL CONTROL

To ensure results of all tests are valid and comparable, care is taken to ensure a static testing environment. Within the scope of this report, a "testing environment" refers to the computer system which was running the simulations – specifications are included in *table 4-1.*

Table 4-1 – Simulation Hardware Specification

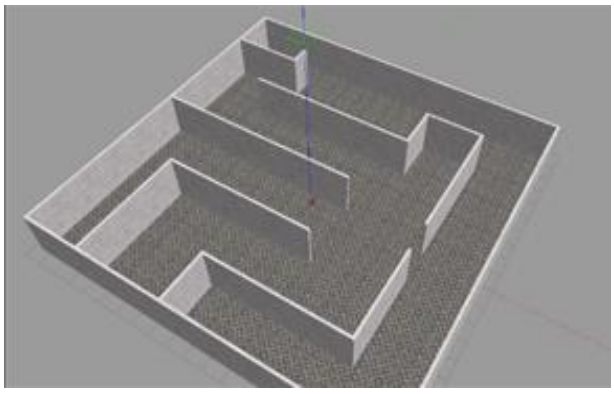
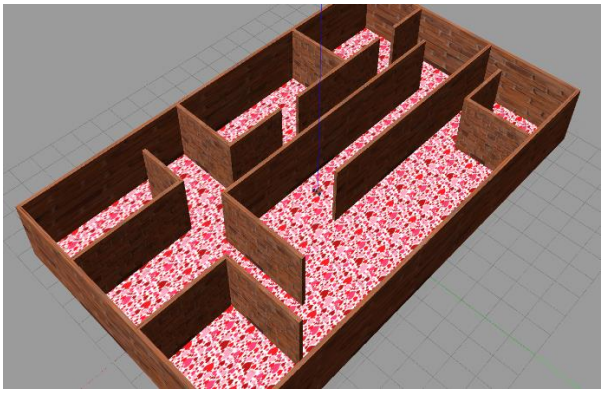
Testing Environment Specifications		
Operating System Configuration	Ubuntu on Virtual Machine	Native Ubuntu
CPU	i5-8265U	i7-970
Number of Cores	4	6
Frequency (GHz)	1.8	3.2
RAM (GB)	8	16
Virtual Machine Version	VMWare Workstation version 16	N/A
Ubuntu Version	18.04	18.04

Attempts were made to ensure stable performance by disabling CPU frequency scaling and associated power management functions. No attempt was made to control ambient or CPU temperature, as this would have little impact, and could be averaged out over multiple test runs.

4.1.3 Simulated Gazebo Testing Environments

To assess the impact of simulated environmental factors, several test worlds were designed in *Blender* configured for use within *Gazebo*. These environments included variations in attributes such as wall textures, the density of obstacles, surface reflectivity, dimensions, and total surface to explore. From the pool of worlds designed during the development of the simulation, two were chosen as test maps for the verification of the ACES functionality, labelled Map 1, and Map 2. Map 1 has been developed as less complex control, with generous manoeuvring space and flat surfaces. Map 2 is designed with more numerous, smaller rooms and narrower corridors and doorways to navigate. The test area also has brick walls that are textured in the simulation using bump maps. A summary of the two simulation environments' attributes is provided in *Table 4-2*.

Table 4-2 – Simulated Testing Environments

Map 1	Map 2
	
Dimensions	
19m x 18.75m	19m x 11.25m
Features	
Wide corridors Wide thoroughfares No textures/reflective attributes Monocolour	Narrow corridors & small rooms Narrow thoroughfares Simulated brick wall texture Multiple RGB levels

Map 1 has been used in testing for formal validation of system functionality. While both maps were used for initial assessment of 2D map-merging capabilities, due to extended development impacting available testing time only Map 1 was used for experiments conducted in assessing scalability and mapping accuracy of the system, as well as the effect of velocity on performance.

4.1.4 SHORTCOMINGS OF SIMULATED TESTS

While simulation is the best possible choice for comprehensive testing in this situation, it does have some drawbacks. A physics simulator like *Gazebo*, for example, is incapable of accurately modelling interference between different robot sensors. In a real-world scenario, having multiple robots, each with multiple sonar sensors, in the same area may result in erroneous sensor readings due to interference. Within the framework of these simulations, factors such as light levels, terrain consistency, and the robot's mechanical responsiveness are also ideal.

Finally, the ROS based system is designed to operate using numerous processors running different sections of the software independently – this is expected to drastically limit the scalability of the system due to the functionality of multiple simulated robots being conducted using a single PC for these experiments.

4.1.5 ADDITIONAL SOFTWARE USED IN EXPERIMENTS

4.1.5.1 3D TO 2D MAP CONVERTER

The accuracy of the mapping has been evaluated by comparing the produced OctoMaps with the representations of the virtual environments used to test the robots. Initially, those reference maps were obtained by the implementation of a node converting Gazebo worlds to voxels representation, before transforming them into OctoMaps for comparison with the output of the 3D Map-merger.

After migration to gmapping SLAM, this node was modified to transform *.world* files to a 2D occupancy grid representation output as *.pgm* and *.yaml* files. This operation calls a plugin installed in the *.world* file, performing wavefront exploration along the occupancy grid identifying all the objects and explorable area, and publishing the results to map2d ROS topic.

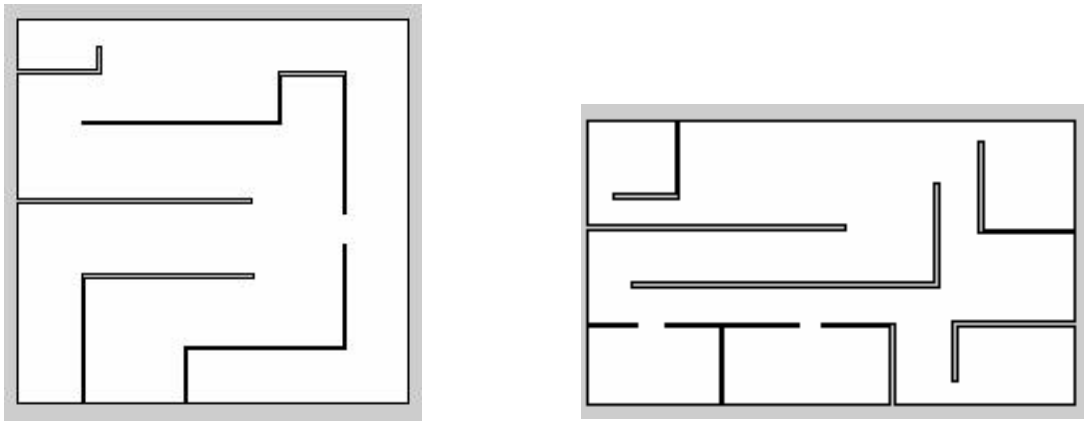


Figure 4-1 – Occupancy grid reference generated from simulated environments Map 1 and Map 2

To contrast reference grid and map merger output, it was attempted to implement automatic occupancy grid comparison. However, the latest version available of this node was outdated in terms of ROS version compatibility, so the team opted to develop an image processing solution to extract the desired data.

4.1.5.2 IMAGE PROCESSING FOR MAP ACCURACY ANALYSIS

The team used *Photoshop* for resizing 2D reference maps and maps generated by the ACES system for image processing and analysis. The software was also used to overlap the reference and generated occupancy grids for a clear visual representation of the successfully mapped area within the reference environment resulting from experiments.

To acquire quantitative data representing the accuracy of the system's mapping output, the engineers developed a Python script using Google Colab Notebook. *Figure 4-2* presents a flow diagram of the 2D map comparison program. Functions included with the OpenCV Image processing library are used to convert 2D sample maps and map-merger output images to greyscale, before thresholding to provide a binary representation of mapped and unmapped areas of the simulated floorspace.

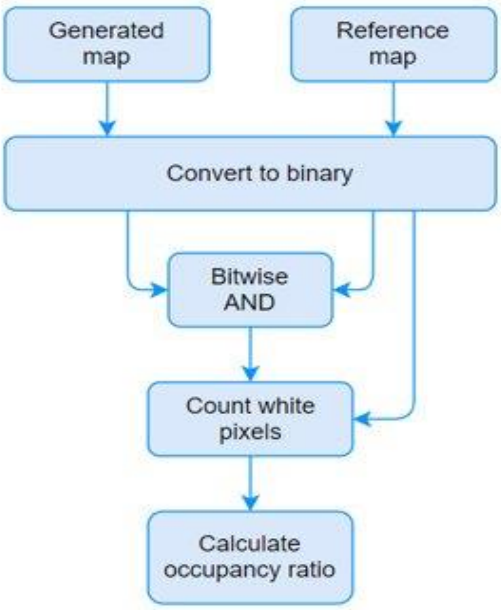


Figure 4-2 – OpenCV Data Extractor Program Flow Diagram

The number of white pixels in each map image is extracted using the countNonZero function – this pixel count is proportional to the mapped floorspace cells in the binary map. To remove white pixels resultant from partial detections of walls, a bitwise AND operation is carried out on the binary representations of the generated map with the reference map as shown in figure 4-3:

Figure 4-3 – Comparison of generated map vs bitwise-AND map

An accuracy measurement is determined by comparing the desired number of white pixels in the reference image to those in a map provided by the ACES system, with the program calculating a percentage of a map that has been successfully explored. The program also facilitates the efficient generation of figure plots using the *matplotlib* library and is included alongside the delivered open-source ACES package as a *.ipynb* Notebook file.

4.2 PROPOSED EXPERIMENTS

Experiments have been designed to investigate the testing objectives identified and set out by the engineers. First, a robust verification of the ACES system functionality is proposed confirming the operation of the software components. This is followed by an assessment of system scalability and map-merging accuracy leading into experimentation analysing the effect of maximum robot velocity on the accuracy of output map.

4.2.1 FUNCTIONALITY VERIFICATION

The system's main areas have been individually tested to ensure that each node had been implemented appropriately and is performing correctly. Verification tests were repeated three times by the engineers before confirming the validation of the ACES software.

4.2.1.1 MOBILITY AND NAVIGATION VERIFICATION

Basic mobility tests were performed on a single robot to test its manoeuvrability and correct movement. Teleoperation package **teleop_twist_keyboard** was installed, granting the testing engineer manual control of the robot's movement. After ensuring correct mobility, the navigation and frontier exploration nodes were activated, and the robot's navigation was tested with destinations set by publishing goals using *RViz*. Tests are listed in *Table 4-4*:

Table 4-3 – Mobility and Navigation Verification Experiments and Expected Outcome

Experiment – Mobility Validation using Teleoperation	Expected outcome
Move forward/backwards	Successful
Rotational movement	Successful
Move on a straight path with no deviation	No deviation
Manually navigate obstacles from point A to point B	Successful
Experiment – Navigation Validation using *RViz*	**Expected outcome**
Move forward/Backwards	Successful
Rotational movement	Successful
Move to a distant point	Successful
Navigate to a point behind an obstacle	Successful

4.2.1.2 2D GMAPPING SLAM VERIFICATION

2D SLAM validation consisted of the mapping of the test virtual environments by the robot. Mapping was performed initially on a static robot and subsequently, it was proceeded to perform gmapping SLAM in parallel with navigation and exploration. Results were analysed to determine to what extent the system was having a satisfactory output.

Table 4-4 – Expected Outcomes of SLAM gmapping experiments

Experiment	Expected outcome (Map1)	Expected outcome (Map2)
Map with a static robot	Successful	Successful
Map with a robot in movement	Successful	Successful
Evaluate the suitability of the produced 2D maps	Suitable	Suitable

4.2.1.3 3D ORBSLAM SLAM MAPPING VERIFICATION

Experiments verifying ORBSLAM included the validation of ORBSLAM3 and ORBSLAM2 in its monocular and RGB-D versions. The tests involved the processing of real-world images provided by TUM RGB-D dataset. After confirming correct performance with the dataset images the systems are challenged with the two testing environments, with map2 being texturized as a contrast to map1 lack of textures.

Table 4-5 – Expected Outcomes of ORBSLAM Experiments

Experiment	Expected Outcome (TUM Dataset)	Expected outcome (Map1)	Expected outcome (Map2)
Map with a static robot	-	Successful	Successful
Map with a robot in movement	Successful	Successful	Successful
Evaluate the suitability of the produced OctoMaps	-	Suitable	Suitable

4.2.1.4 MAP-MERGER VERIFICATION

Before incorporating the 3D map merger node into the ACES system, it has been tested with a *.bag* file dataset to ensure its correct functionality. The selected dataset is Collaborative SLAM Dataset, [47] a collection of point clouds from four different environments containing several different sequences to be merged. The team chose this version because it uses a Velodyne Lidar, [99] a type of 3D Lidar that produces point clouds. This does not require the use of a SLAM system to process the acquired data, and therefore allowing the engineers to isolate the map merger node for experimentation.

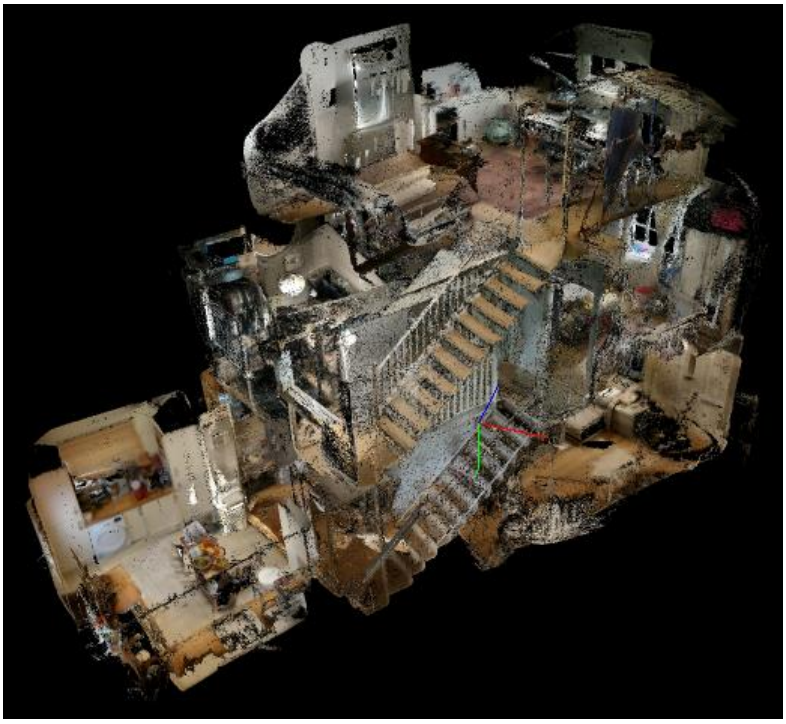


Figure 4-4 Collaborative SLAM dataset example [47]

With the validation completed, the map merger has been integrated into the ACES system, with the combination of the point clouds generated by ORB SLAM creating a global map. The output of the map merger was visualized with *RViz*.

The testing process for the 2D map merger is simpler than the method used for 3D. This is explained by the need for the sensor data to be processed by a SLAM system to convert it into an occupancy grid, and thus be suitable for stitching. For this reason, and due to the late stage of development at the time of the testing, the map merger node was directly implemented on the ACES system and tested by stitching the maps generated by the 3 robots.

Table 4-6 – Expected Outcomes of Map-merging Experiments

Experiment	Expected outcome (3D ORB-SLAM)	Expected outcome (2D gmapping SLAM)
Merge collaborative SLAM dataset maps	Successful	Not applicable
Merge robot's output maps	Successful	Successful
Assess the correctness of the merged maps	Correct	Correct

4.2.2 SCALABILITY ASSESSMENT

To test the potential scalability of the system, the engineers have designed the following experiment to stress test how robust the map-merging system is when operating with an increased number of robots. To evaluate the processing power limitations of the system in terms of scalability, it is decided to run ACES in both testing environments.

ACES as a single robot system, and as a multi-robot of 3 and 5 units are simulated on both terminals, evaluating the CPU performance, and registering the results.

The expected results for this experiment are displayed in *Table 4-8* below:

Table 4-7 – Expected Outcomes of Scalability Assessment

Experiment	Expected results [Ubuntu on Virtual Machine]	Expected results [natively run Ubuntu]
Single robot	Correct performance	Correct performance
Three robots	Correct performance	Correct performance
Five Robots	Limited performance	Correct performance

4.2.3 MAP-MERGER ACCURACY ASSESSMENT

The team has aimed to assess the map-merging accuracy of the ACES system, comparing a single-robot and a multi-robot system, contrasting the result, and evaluating to what extent a multi-robot system can prove advantageous to mapping an environment.

Assessing the accuracy of the system involves comparing the generated maps with the worlds the robot was trying to map and identify to what extent ACES had successfully mapped the relevant areas. Before carrying out the experiments, it is required to convert the virtual environments used in testing into occupancy grids, using the converter node mentioned previously. The experiment consists of operating the ACES system as a one, three, or five robot team for a set time of 1500 seconds. The occupancy grid generated by the map merger is then processed with the developed accuracy analysis program and compared to the reference map to calculate a percentage of the successful mapped area.

4.2.4 EVALUATION OF VELOCITY EFFECTS

The effects of velocity on the system's behaviour and mapping accuracy have been identified as an area of interest by the team. To evaluate the impact on ACES performance and contrast the results between single and multi-robot, one, two and three-unit teams of ACES robots have been simulated to allow comparison. Configuration of the system has limited the maximum velocity of the robots, maintaining the minimum velocity at 0.1 m/s. The maximum velocities used in this experiment are 0.1, 0.3, 0.5, and 1 m/s. The system simulation is activated for 1500 seconds, a time that has been estimated sufficient to achieve a satisfactory percentage of completion. At this time limit, the generated map is saved using **map-saver**, a ROS Service that allows dynamically generated maps to be saved [100]. The results are analysed using image processing, with photoshop for clear visual analysis, with map accuracy analysis software used to acquire quantitative data.

Chapter 5 RESULTS AND DISCUSSION

5.1 VERIFICATION OF ACES SYSTEM FUNCTIONALITY

This section provides the results of the proposed ACES verification testing procedures. Visualised results of these validation trials are presented with *Gazebo* and *RViz* GUI screen shots included in Appendix F.

5.1.1 NAVIGATION AND MOBILITY VERIFICATION

The tests have been executed according to plan, with the single robot performing all the operations required in the map 2 simulation environment with no malfunctions to report. Results of mobility experiments are summarised in *Table 5-1*.

Table 5-1 – Results of Mobility Verification

Experiment	Outcome
Move forward/Backwards	Successful
Rotational movement	Successful
Move on a straight line with no deviation	No deviation present
Manually navigate from point A to point B	Successful

Upon obtaining successful results for the mobility validation, the testing engineer proceeded to validate the robot's navigation. The robot successfully met all the objectives and navigation of the robot was validated allowing for further tests as shown in Table 5-2.

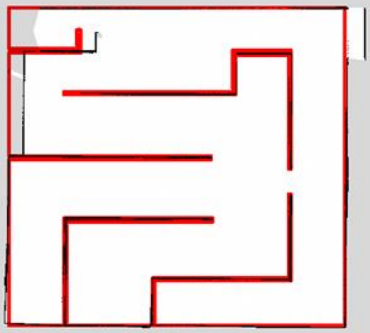
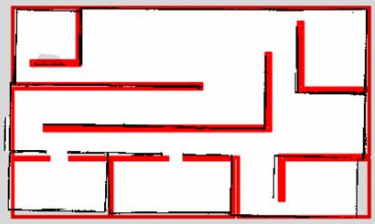
Table 5-2 – Results of Autonomous Navigation Verification

Experiment	Outcome
Move forward/Backwards	Successful
Rotational movement	Successful
Move to a distant point	Successful
Navigate to a point behind an obstacle	Successful

5.1.2 2D GMAPPING SLAM MAPPING VERIFICATION

Results of 2D gmapping package operation verification are provided in *Table 5-3*. As proposed, the mapping simulation was run stationary and in motion for 1500 seconds.

Table 5-3 – Results of gmapping Verification

Experiment	Outcome (Map1)	Contrasted Occupancy Grid (Map1)
Map with a static robot	Successful	
Map with a robot in movement	Successful	
Evaluate the suitability of the produced maps	Suitable	
Experiment	**Outcome (Map2)**	**Contrasted Occupancy Grid (Map2)**
Map with a static robot	Successful	
Map with a robot in movement	Successful	
Evaluate the suitability of the produced maps	Suitable	

2D SLAM offers an adequate performance when performing stationary mapping, proceeding to operate along with exploration functionality. Generated occupancy grids after 1500 seconds have been analysed using image processing, revealing satisfactory performance from 2D mapping in both testing environments.

5.1.3 3D ORBSLAM SLAM MAPPING VERIFICATION

As discussed in *Section 3.5.3* of this report, an initial attempt has been made to implement the ORBSLAM3 visual-inertial RGB-D algorithm. However, due to a known bug in the

ORSLAM3 code, verification has failed in all cases. For this reason, ORBSLAM2 has been implemented instead, using the Applied AI Initiative ROS wrapper [84]. The Kinect RGB-D sensor included in the ACES robot design consists of a virtual stereo system of a tightly coupled monocular RGB-D camera and an infra-red depth sensor. As such, the outputs of the Kinect sensor may be used by both RGB-D and monocular SLAM systems. Both the RGB-D and monocular ORBSLAM2 nodes of the ROS wrapper were tested during verification, with the results of these trials included in *Tables 5-4* through *5-6*.

Table 5-4 – Results of ORBSLAM3 RGB-D Verification

Experiment	Outcome (TUM Dataset)	Outcome (Map1)	Outcome (Map2)
Map with a static robot	-	Unsuccessful	Unsuccessful
Map with a robot in movement	Unsuccessful	Unsuccessful	Unsuccessful
Evaluate the suitability of the produced OctoMaps	-	Not suitable	Not suitable

Table 5-5 – Results of ORBSLAM2 RGB-D Verification

Experiment	Outcome (TUM Dataset)	Outcome (Map1)	Outcome (Map2)
Map with a static robot	-	Unsuccessful	Low Quality
Map with a robot in movement	Successful	Unsuccessful	Low Quality
Evaluate the suitability of the produced OctoMaps	-	Not suitable	Not suitable

Table 5-6 – Results of ORBSLAM2 Monocular Verification

Experiment	Outcome (TUM Dataset)	Outcome (Map1)	Outcome (Map2)
Map with a static robot	-	Unsuccessful	Low Quality
Map with a robot in movement	Successful	Unsuccessful	Low Quality
Evaluate the suitability of the produced OctoMaps	-	Not suitable	Not suitable

The results of testing the ORBSLAM3 RGB-D algorithm, shown in *Table 5-4*, reveal that the algorithm fails in all cases. This is due to segmentation and timestamp mismanagement in the ORBSLAM3 code with resulting faults causing catastrophic failure of the ORBSLAM3 system soon after initialisation. For this reason, no usable data could be retrieved from the ORBSLAM3 system in any case. However, the code developers are aware of this fault and are working to remove this bug. Implementation of the ORBSLAM3 visual-inertial RGB-D algorithm remains a stretch goal of the project, as this would constitute a significant improvement over ORBSLAM2.

Both the RGB-D and monocular nodes of the ORBSLAM2 ROS wrapper have been tested, and these results may be seen in *Table 5-5* and *5-6* respectively. In both cases, testing on sequences of the TUM RGB-D dataset produced reasonable results. These results were assessed via simple visualisation using *RViz*. It was observed that both systems initialise as expected and produce reasonably similar point clouds and pose estimates.

Neither the RGB-D nor the monocular ORBSLAM2 nodes can initialise properly in the untextured map (map 1) when used as part of the ACES system. As ORBSLAM is a visual SLAM algorithm, the failure to initialise is likely due to the lack of visual information in the *Gazebo* world. The monochromatic surfaces and lack of environmental objects likely limited the number of extractable ORB features. It is not unfeasible that robots would be required to operate in similarly monotone environments as map 1: for example, unfurnished offices, or cave systems. Based on the verification results, it is reasonable to conclude that the use of a visual SLAM system such as ORBSLAM would likely be inappropriate in these environments. However, real-world testing is necessary to confirm for certain. Due to the failure to initialise, no point clouds or pose estimates were produced by the RGB-D and monocular implementations for map 1. Due to this failure, no data is passed to the OctoMap Server to process, and thus no occupancy grids are produced. Without viable occupancy grids, the navigation and exploration systems in the ACES design cannot function, and therefore autonomous movement is not possible.

When testing the monocular node of the ORBSLAM2 wrapper in the textured map 2, the algorithm initialises properly and produces reasonable pose estimates and point cloud maps. However, scale drift is evident and within a few minutes of testing the maps produced are entirely inaccurate. This is true of both the stationary and mobile testing results. It can be

observed that this scale drift is exacerbated significantly during the mobile test, with errors becoming larger and more frequent during turning operations. As well as this, many spurious map points are produced in the point clouds. In the stationary test, it is seen that the robot's pose varies with time, likely owed to the inconsistent map. As the system initialises properly and produces a point cloud and pose estimates, processing by the OctoMap server is possible. However, due to the significant errors present in both the point cloud and pose, the use of the resulting occupancy grid within the exploration and navigation systems in ACES is deemed not possible.

When testing the RGB-D node of the ORBSLAM2 wrapper in the textured map, the algorithm again initialises properly and produces reasonable point clouds and pose estimates. Compared to the monocular tests, the RGB-D node is significantly more robust in overcoming scale-drift within the *Gazebo* simulation. However, erroneous map points are still produced, and in the stationary test, the pose is again seen to be inconsistent. Although the resultant maps are of slightly higher quality than the monocular implementation, the presence of spurious map points and inconsistent pose render the resultant occupancy grids unusable for navigation and exploration. An *RViz* visualisation of the stationary RGB-D results may be seen in figure 5-1. In this figure, the points in the point cloud are shown in green & red, the pose estimate is shown as a red arrow, and the down-projected occupancy grid is shown as black, white, and grey squares on a plane. It may be seen that the pose arrow is pointing in the wrong direction, and there is a significant amount of noise in the point cloud and occupancy grid. It is important to note that the success of the ORBSLAM2 nodes in processing the real-world camera feeds of the TUM RGB-D dataset likely indicates that the failure of these nodes in the simulated *Gazebo* environments is due to the nature of the simulation itself. Although *Gazebo* is a capable physics simulator, it is possible that *Gazebo* is not a viable choice for testing visual SLAM systems. This may be the reason that previous researchers have chosen to use high-fidelity game engines such as Unreal Engine when testing visual SLAM systems [36].

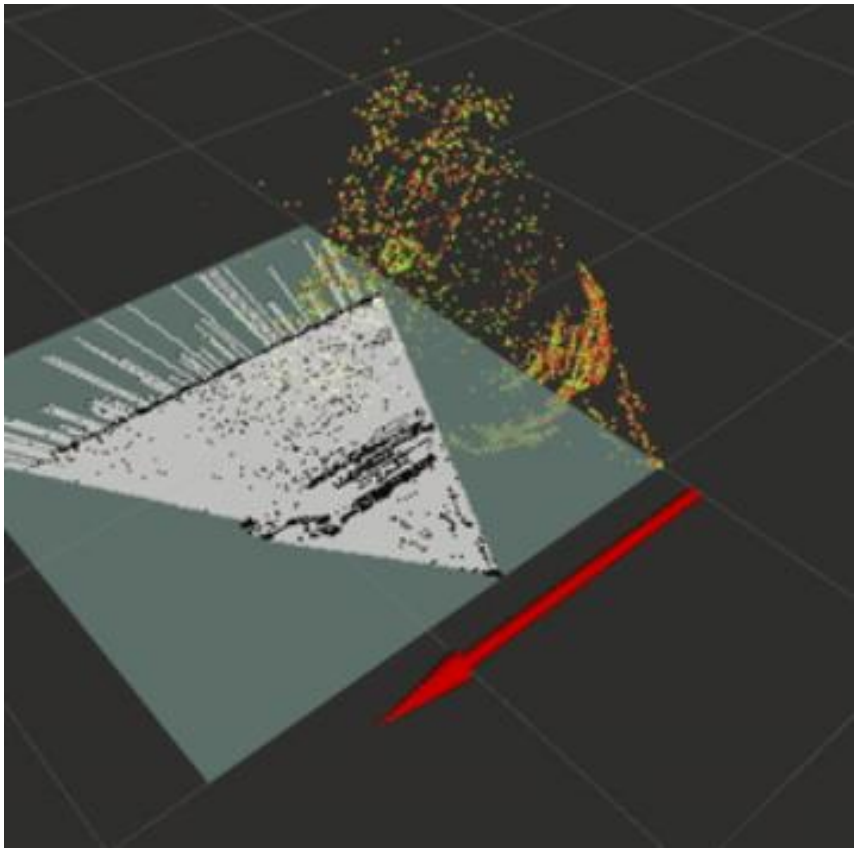


Figure 5-1 – 3D RGB-D ORBSLAM Map 2 Stationary results shown in RViz

Due to restrictions imposed by the CoVID-19 pandemic testing of the ACES system outside of simulations has been impossible, and so this possibly can neither be confirmed nor dismissed.

While ORBSLAM2 has been projected by the engineers as likely being the optimal method of mapping with the ACES system, the unusable results obtained during verification have forced the team to continue experimentation using only the 2D ACES system with LIDAR and gmapping.

5.1.4 MAP-MERGER VERIFICATION

To validate the correct functionality of the 3D map merger node, it was tested with a Collaborative SLAM Dataset before being included in the ACES System. The testing was successful, and the node merged the point clouds introduced as shown in *Figure 5-3* below:

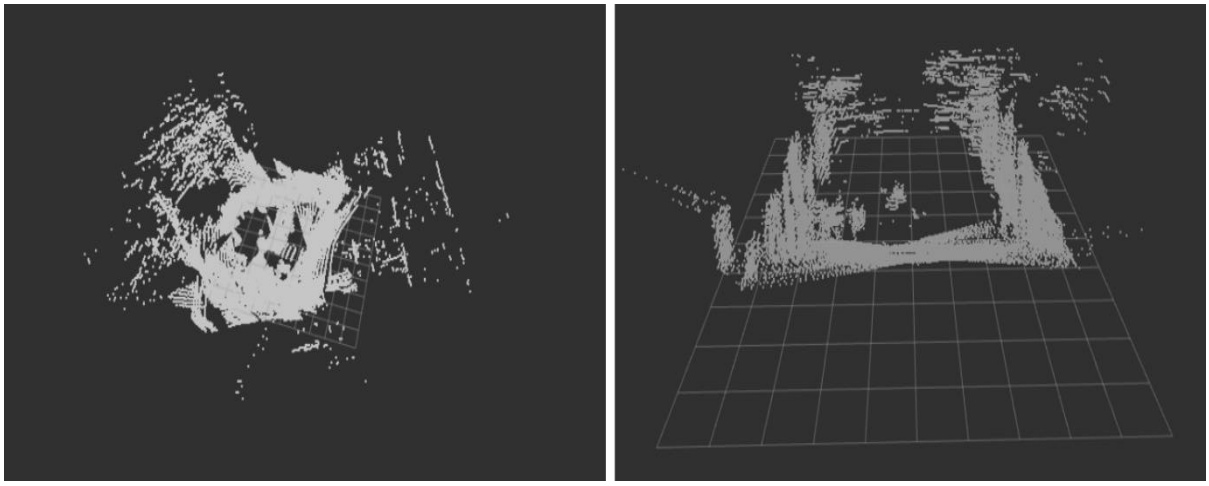


Figure 5-2 – Map merger verification results shown in RViz

The image on the left shows the point clouds from Collaborative SLAM Dataset as they are first received by the map merger. The node then processes the data and allocates the map points in their correct location, generating the global map displayed on the right.

Table 5-7 provides the results of the map-merger verification testing stage – as previously reported, the single robot 3D ORB SLAM was malfunctioning by the completion of the development phase, thus further planned verification of this feature could not be completed. Map-merging based on the validated 2D gmapping input delivered satisfactory results, finalizing verification of the individual ACES system components.

Table 5-7 – Results of Map-merger Verification

Experiment	3D map merger results	2D map merger results
Merge collaborative SLAM dataset maps	Successful	Not applicable
Merge robot's output maps	Unavailable	Successful
Assess the correctness of the merged maps	Unavailable	Correct

5.2 RESULTS OF SCALABILITY ASSESSMENT

The experiments measuring scalability were done by simulating the system in both a computer running Ubuntu natively and one operating through a Virtual Machine, with a gradually increasing number of robots. The CPU performance was evaluated and registered. The CPU resource usage was measured with the *System Monitor* application within Linux, highlighting that the CPU was saturated with usage rates of 100%.

Table 5-8 – Results of Scalability Assessment

Experiment	Ubuntu on Virtual Machine	Natively run Ubuntu
Single Robot	Correct Performance	Correct Performance
Three Robots	Malfunctioning	Correct performance
Five Robots	Malfunctioning	Malfunctioning
Two Robots	Correct Performance	Correct Performance

ACES as a single robot was successfully simulated on both terminals, offering optimal performance in both cases.

When simulating the system as a 3-robot network on the Virtual Machine Ubuntu, it was observed that one of the robots was operating correctly, while the other two presented malfunctions. The movement of the malfunctioning units was not continuous, with the robots experiencing difficulties producing a local or global plan. Further to this, the LIDAR sensor was giving erroneous measurements, generating an inaccurate map.

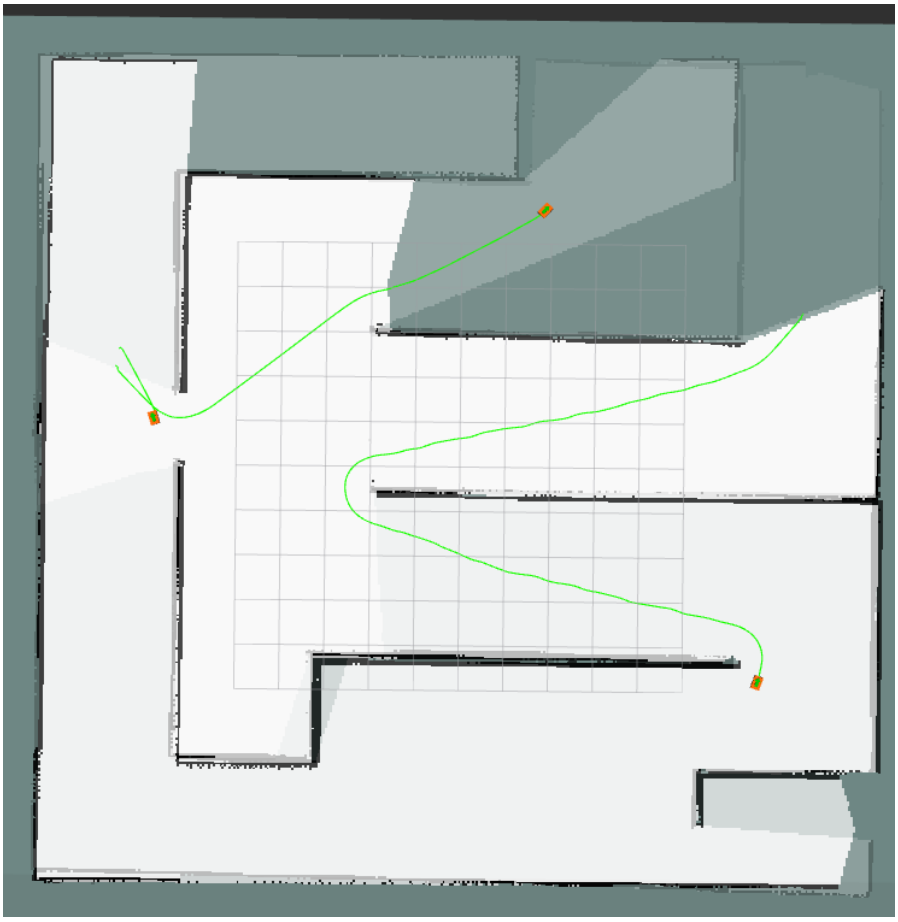


Figure 5-3 – Correct Performance with 3 robots using native Ubuntu

The same version of the robots has then been simulated on the natively run Ubuntu computer, demonstrating a satisfactory operation as shown in *figure 5-3*. After the results given by the 3 robots system on the Virtual Machine, it was considered unrealistic to attempt to simulate the 5 robots ACES in that environment. It has therefore proceeded to be run in the native Ubuntu machine. Similarly, as the experienced with 3 units in the Virtual Machine, some of the robots were experiencing malfunctions related to path planning and mapping. *Gazebo* and *RViz* graphics were disabled to save processing power, but the system did not present sensible improvements.

While the team has succeeded in launching a 5-robot ACES team, as demonstrated in *figure 5-4,* it was evident that it would not be possible to obtain valid test results from the 5-robot system due to processing power limitation. In response to these findings, it was decided to modify the sets of robots used in the experiments to one, two, and three units and repeat the test.

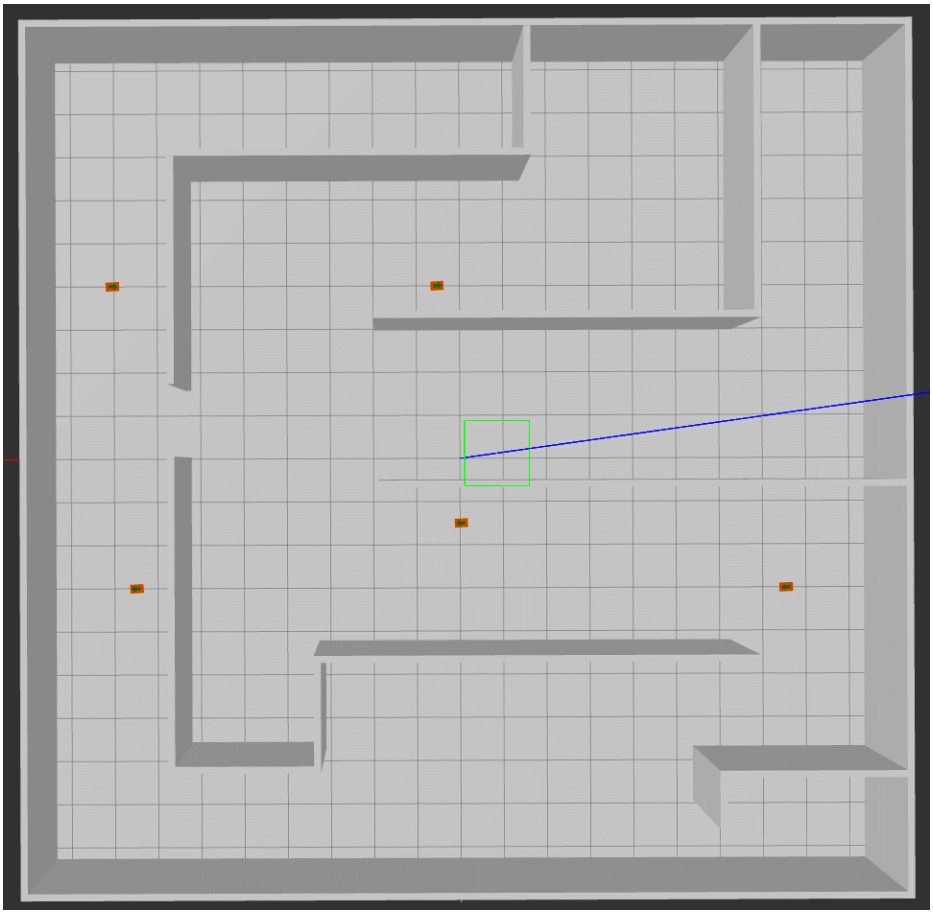


Figure 5-4 – Unsuccessfully launching 5 ACES Robots in Gazebo using native Ubuntu

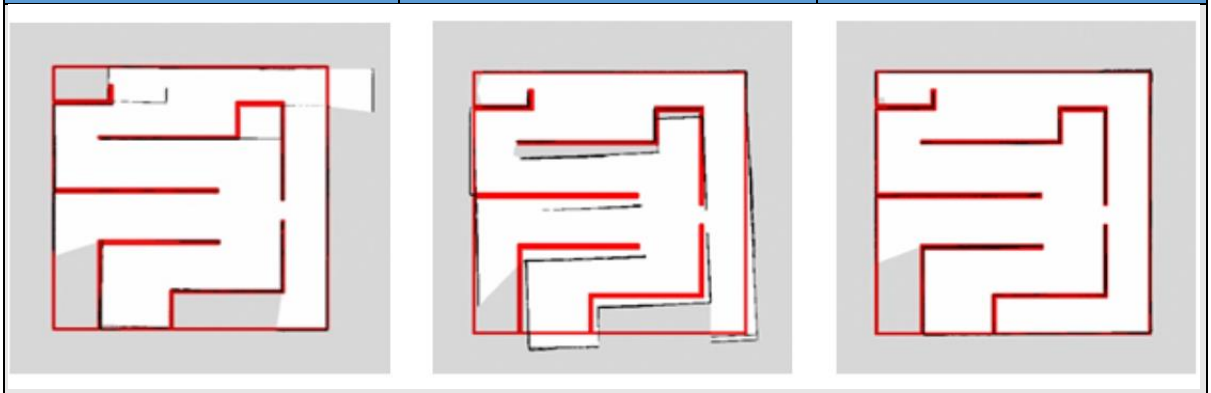
5.3 RESULTS OF MAP-MERGER ACCURACY EXPERIMENTS

Comparing the performance of a single-robot and a multi-robot system allowed evaluating the benefits from scaling the operational units within the network.

This experiment was designed to explore and map the area first with a single robot, and then with multi-robot systems of three and five bots, for better identification of the effects of a larger number of operational units. However, the scalability assessment revealed that it was not possible to obtain valid results from the simulation of 5 robots, so results were carried out as a single robot and with multi-robot networks of 2 and 3 units at a maximum speed of 0.3 m/s.

Results are displayed in *Table 5-9*. It can be observed that ACES with one and two robots present inaccuracies in the mapping, misplacing the walls and limits of the test room. In contrast with this, 3 ACES units offer improved accuracy, correctly identifying every internal and perimetral wall within the room.

Table 5-9 – Map-merger Accuracy Experiment Results

Output of map-merging completion (white space) overlaid with Sample Map (red outline) – Parameters: 0.3m/s max velocity			
Testing Environment	Native Ubuntu	Test Map	Map1
One Robot	Two Robots	Three Robots	
			
Accuracy (%) of map output relative to 2D Sample Map 1			
One Robot	Two Robots	Three Robots	
83.39	79.24	95.54	

After extracting the quantitative data with OpenCV and calculating the percentage of an explored map, it was seen that while one and two robots were offering a similar performance around the region of 80% completion, it was when the system was scaled up to three units was when a sensible improvement was appreciated. Three robots demonstrated the benefits of a multi-robot system by completing 95.54% of the mapped area.

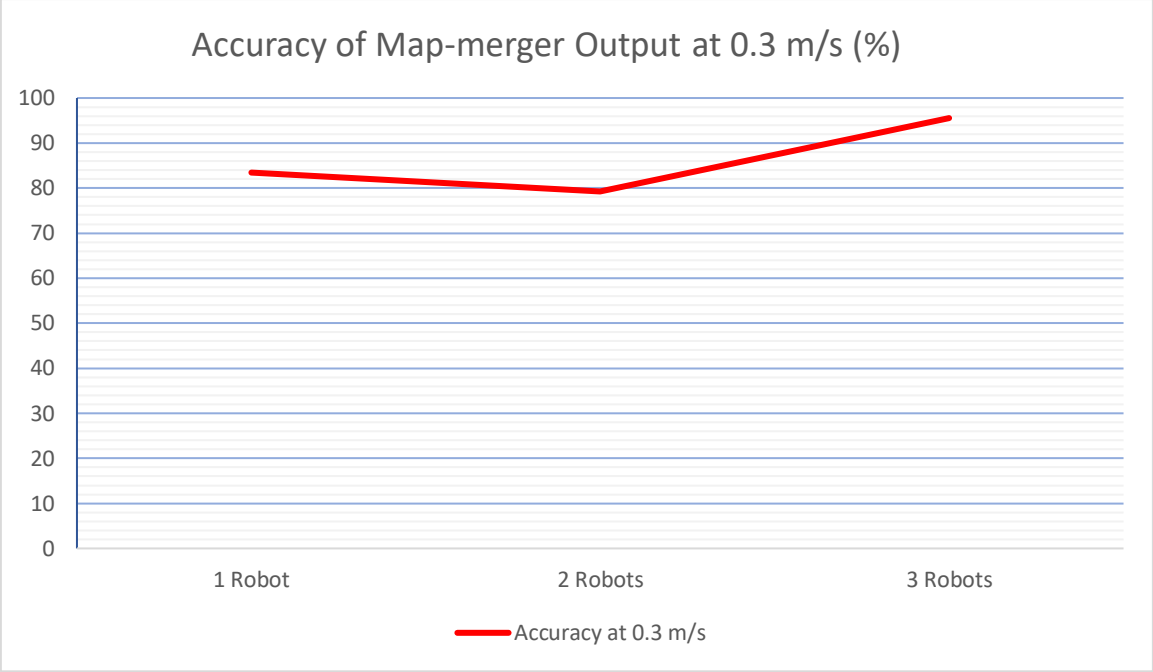


Figure 5-5 – Accuracy of map-merger output at 0.3 m/s (Map1)

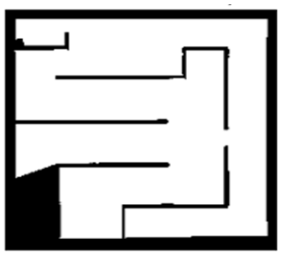
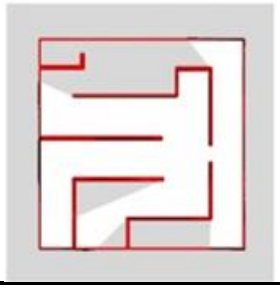
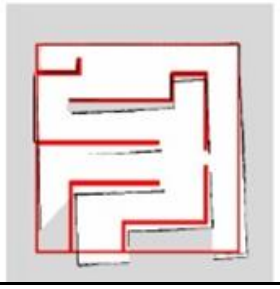
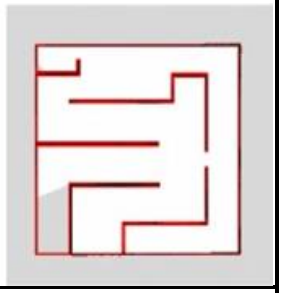
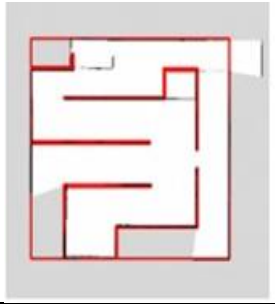
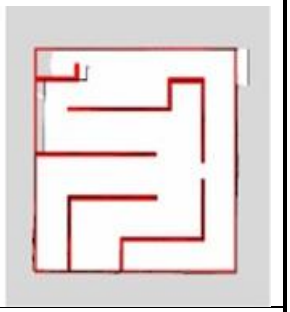
While the ACES team consisting of 3 robots mapping accuracy indicated satisfactory performance, the team believes that if the simulation with 5 robots would have been possible, the improvement with respect to the single robot and two-unit multi-robot would have increased accuracy further, confirming the benefit of employing a multi-robot system over a single-robot system.

5.4 Results of Varying Velocity on ACES Mapping Accuracy

Robots in teams of 1, 2, and 3 were configured to run at a maximum velocity of 0.1, 0.3, 0.5, and 1 m/s for 1500 seconds as proposed. Experimentation was carried out using the Native Ubuntu testing environment. The custom program for determining map accuracy successfully analysed the output from the map merger, providing quantitative data from the experiments. The results obtained were registered using image processing and are displayed in *table 5-10* with higher resolution output images included in Appendix G.

Table 5-10 – Analysis of Maximum Robot Velocity on Map-Merger Accuracy

Results of varying robot velocity – accuracy gauged after 1500 seconds simulation time			
Testing Environment	Native Ubuntu	Test Map	Map1

Max Velocity 0.1m/s	**One Robot**	**Two Robots**	**Three Robots**
Image processing Output			
Overlaid Map			
Mapping Accuracy (%)	67.75	80.23	91.21
Max Velocity 0.3 m/s	**One Robot**	**Two Robots**	**Three Robots**
Image processing Output			
Overlaid Map			
Mapping Accuracy (%)	83.39	79.24	95.54

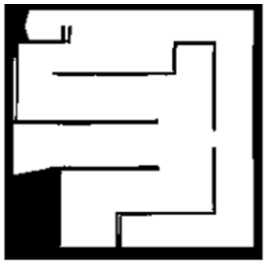
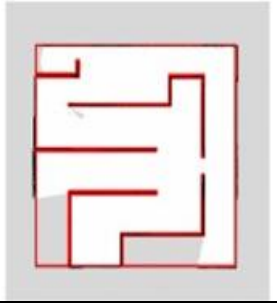
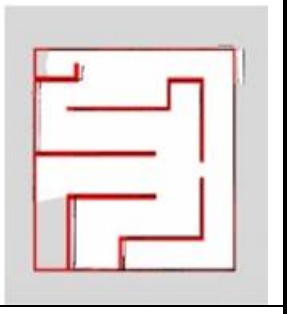
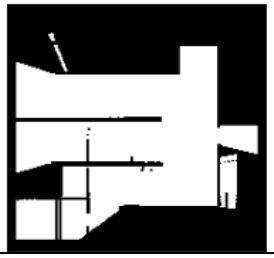
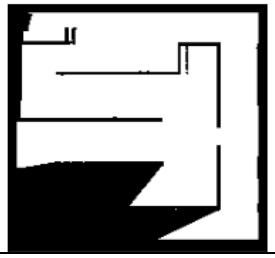
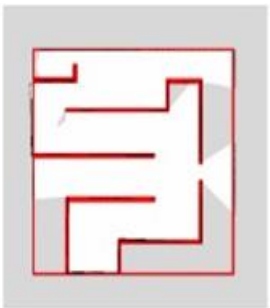
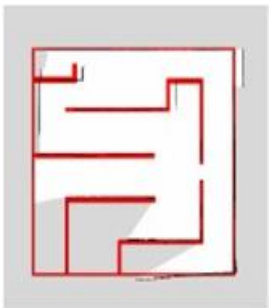
Max Velocity 0.5 m/s	One Robot	Two Robots	Three Robots
Image processing Output			
Overlaid Map			
Mapping Accuracy (%)	90.74	85.42	91.12
Max Velocity 1.0 m/s	**One Robot**	**Two Robots**	**Three Robots**
Image processing Output			
Overlaid Map			
Mapping Accuracy (%)	56.25	74.35	77.50

Figure 5-6 illustrated that the best overall performance was achieved by 3 robots ACES. This set-up achieved the highest map completion at all velocities, recording the best three results among every robot and every velocity, with 91.12% and 91.21% at 0.5 and 0.1 m/s, and peaking with a 95.54% of the mapped area at 0.3 m/s. These experiments also showed that velocities from 0.3 to 0.5 m/s optimize ACES performance, with every set of robots improving its results compared to the other velocities, achieving over 80 % of the mapped area in all cases except one. Given these results, the velocity range from 0.3 to 0.5 m/s was considered the most suitable operating range.

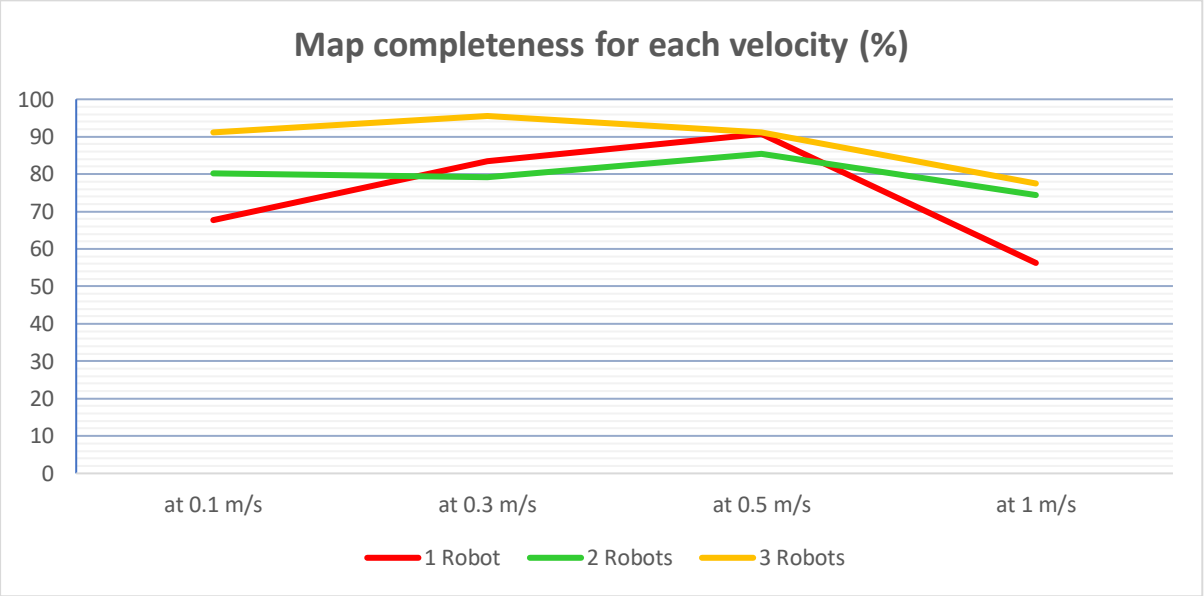


Figure 5-6 – Effects of velocity on mapping results

The single robot system has achieved a higher mapping percentage than the 2-unit multi-robot team, improving results for both velocities within the optimum operating range. However, the single robot's poor performance at 0.1 and 1 m/s, or in other words, at higher or lower velocities than the operating range, showcases that multi-robot systems have a better tolerance to work at velocities outside the most suitable range. The 3-unit multi-robot system offers a satisfactory performance at 0.1 m/s, while the 2-unit ACES team maintains a mapping accuracy percentage above 80%. However, when speeds are increased up to 1 m/s, all systems experience a decrease in productivity, scoring their individual lowest results.

Chapter 6 REFLECTION

This section of the report aims to reflect on the outcomes of the ACES system development and testing in retrospect of meeting the objectives initially laid out by the engineers, providing an evaluation of strengths and weaknesses, and providing recommendations for future development.

6.1 STRENGTHS

Due to the exclusive development of the ACES system within a simulated virtual environment, the team has been able to explore applications of ROS and associated solutions involving the implementation of hardware impractical for physical implementation due to equipment and budgetary limitations. In addition, simulation allows the generation of large, customizable testing environments that would be challenging to recreate for testing a real-world hardware implementation.

The team's primary goal of designing and implementing a ROS based multi-robot system capable of performing exploration and collaborative SLAM-based mapping has been achieved, as evidenced by the successful verification testing of the ACES system using the 2D gmapping configuration. The strategy of developing the ACES system for use with 2D and 3D SLAM inputs has ensured that the team has met the goals required of the project objectives. Despite ORBSLAM capabilities being unavailable for experimental testing with the complete ACES system, extensive development and experimentation were carried out by the engineers. It is believed that the verification of ORBSLAM with the real-world sequences of the TUM RGB-D dataset justifies further experimentation with real-world hardware. The failures of the ORBSLAM node were likely due not to the implementation or configuration of the node, but to the simulated environment itself.

Robust testing and experiments on the system have been carried out, with the use of two static simulation environments (Virtual Machine and Native Ubuntu), with consideration given to ensuring stable performance. Each node in the ACES system used for experimentation has been systematically validated, verifying successful implementation ensuring planned testing has been carried out reliably within the timeframe of the project. It has been demonstrated that 2D SLAM offers reliable and accurate maps of the tested

environments. While the performance of the system is less satisfactory when operated using a single robot or two-unit multi-robot team, it has achieved demonstrably improved mapping performances when utilising a 3-robot configuration.

The results of the velocity experimentation carried out by the engineers indicate that the ACES system offers optimal map-merging performance when operated in the range of 0.3 – 0.5 m/s. Functioning at these velocities, the highest overall mapping performance and the highest average completing percentages for each set-up were achieved.

The system has been packaged and presented as an open-source deliverable available on the team's GitHub page [82], providing step-by-step installation and operation instructions. The team is proud to have contributed to the ROS community, making available ACES software as a scalable and adaptable open-source solution for collaborative exploration and mapping, using a team of robots to improve efficiency and accuracy. It is hoped that the work presented here may be adapted and built upon by others to facilitate further development of low-cost robotic solutions.

The team was able to harness skills acquired from undergraduate studies at Glasgow Caledonian University frequently throughout the project – experience in using Creo Parametric 3D modelling software for mechanical engineering studies and in previous design projects has proved advantageous in rapidly developing robot and environmental models, while modules in intelligent robotics and machine vision have proved invaluable in providing a solid fundamental understanding of programming differential drive robots using the Robot Operating System.

6.2 WEAKNESSES

Because of continued uncertainty over access to university facilities, the team has focused on a simulated solution to the problem of collaborative robot exploration. Simulations are carried out under ideal conditions that are unlikely to be replicated when the system is tested on actual hardware – this limits validation of the system's suitability for application in the real world.

Developing the ACES system using *Gazebo* simulation software has allowed for the implementation of a range of sensors providing data for map building, however, the team

must further consider sensor choice for exploration, environmental conditions, cost of components, and robustness for practical and specialised use.

While the open-source Robot Operating System is advantageous in its availability and boasts an extensive library of functions, the team has been challenged with compatibility issues implementing nodes in more recent versions of ROS leading to delays in implementing all planned functionality. Extended implementation time has been provided in the project timeline to compensate for delays due to debugging to ensure the system is suitable for validation and results of planned experiments can be obtained for analysis, however the extent of data capture was limited due to time lost to this allowance.

It was projected to use the results from ACES with 5 robots to further demonstrate the performance improvements associated with the scaling of operational unit quantity. While these experiments have not been possible for execution due to processing constraints, the results obtained from the 3-robot system have been found to register a consistently higher accuracy across all velocities tested compared to experiments conducted using smaller teams of robots. Processing power limitations have also proved a critical constraint on the depth of the experiments realized. However, these scalability limits are understood to be directly associated with the simulated condition of the experiments. With a physical implementation of the system where each unit conducts decentralised processing on a Jetson Nano CPU, processing limits associated with simulation here would not be present on account of each robot running only its relevant nodes locally. The map-merger master unit remains a centralised node, thus being the only node where performance should be affected by a bottleneck in CPU performance.

As the navigation and exploration systems used in the ACES robots rely on 2D occupancy grids with no z-levels, the ACES robots are incapable of mapping environments with multiple overlapping areas: for example, an apartment block. This is a significant limitation in many environments. Researchers have suggested the use of elevation maps to account for this [101]. However, no ROS nodes capable of performing exploration and navigation via elevation maps were identified during this project. As such, this remains an interesting area for further development.

The project being limited to implementation within a simulation also affected the verification of ORBSLAM features. Although significant development and testing were carried out on ORB SLAM 3D mapping, the team has been unable to implement this feature for experimentation within the time scale of the project. Based on the satisfactory performance offered when processing a dataset with real-world images, the team understands that ACES would not experience the malfunctions present while mapping a virtual environment. However, due to the lack of a physical implementation of the system and despite the promising dataset test results, the team could not validate the correct performance of ORB SLAM in a real-world environment.

No testing of the ACES system within a dynamic environment was feasible during this project due to the relative simplicity of simulation offered by *Gazebo*. Instead, two static environments (maps 1 and 2) were simulated, with the only dynamic, moving, objects being the robots within these environments. Although many real-world environments are static, the ability of a robotic mapping system to cope with movement within the environment is critical to evaluate. As this was not reasonable within *Gazebo*, no comments have been made on the suitability of the ACES system within dynamic environments.

The 2D SLAM was implemented and validated in the simulated environment, verifying its suitability to operate in the real world. However, the LIDAR sensor used to gather data for mapping and navigation is effective but expensive to deploy and cumbersome. Further development of the 3D ORB SLAM system would facilitate the use of the significantly more cost-effective Kinect sensor and is thus recommended.

6.3 SUMMARY OF RECOMMENDED FUTURE DEVELOPMENT

The research, development, and testing ACES has highlighted areas of the system that require improvement or that would be interesting to explore in further detail in future work.

To enhance the durability of the system and improve multirobot navigation by preventing collisions against walls or other robots, it would be necessary to implement an obstacle avoidance system. The required hardware comprised of an array of ultrasonic sensors were considered during research and included in the simulated design of the robots, however, no software solution was developed.

The team considered it interesting to investigate further on the malfunctions experienced using ORBSLAM in mapping the virtual environment. Based on the correct functionality using TUM RGB-D Dataset the team believes that further development should be dedicated to identifying the elements that are preventing the system from performing correctly while used in a simulation, and further develop ORBSLAM to make it compatible with ACES within a virtual environment for testing and to enable further development of system features.

As well as this, the pose estimates of the SLAM systems, both 2D and 3D would benefit from the inclusion of other sensors. As presented in section 2.6.2.2 of this report there is currently a trend in the literature towards the incorporation of sonar, wheel encoder, and IMU measurements via extended Kalman filters. This approach has been shown to increase overall SLAM accuracy, and possibly most importantly, increase SLAM robustness in the case of dynamic environments.

The next stage of development of the ACES system involves the physical implementation of ACES using hardware. The migration of a simulated system to an electro-mechanical network would grant the possibility to validate the system in a real-world environment and elevate the complexity of the experiments. To provide further validation of ACES, it would be beneficial to verify the correct functionality of every node within a real-world environment. As discussed before, the processing power limitations encountered during the experimentation of this project would not be present in the physical implementation of ACES. With the map merger as the only centralised node, it would be an area of interest to evaluate the maximum number of robots that can be simultaneously integrated within ACES operation.

A proposed upgrade from the current ACES version is the substitution of the differential robots by Unmanned Aerial Vehicles (UAV). Improved mapping accuracy and suitability to operate in difficult access or navigation environments are some of the advantages derived from the robots not being restrained to operate at ground level.

Chapter 7 EVALUATION OF TEAM PERFORMANCE

This section of the report aims to discuss the effectiveness of the team in approaching and tackling the ACES project, focusing on strengths and weaknesses of the planning, teamwork, development strategies, and data gathering accomplished by the group from an organizational perspective.

Primarily, the team has devoted ample time to plan the project, maintaining a detailed timeline to ensure work can be achieved in time so as not to negatively impact development downstream. This planning has been cemented and clearly illustrated through the creation and ongoing updating of the project Gantt chart – an important lesson learned from previous projects and implemented here is the detailed breaking down of tasks. This has ensured that team members have had specific goals to achieve, clear deadlines, and an overview of simultaneous development by colleagues. Due to this rigorous planning, adequate time was devoted to all areas of the project and all critical goals were met within the proposed timeline. This has also ensured that responsibility and workload have been shared fairly among the group ensuring efficient development and participation from all engineers in all key areas. The final version of the project Gantt chart has been included in *Appendix I* of the report.

Planning has been supported through the organisation of regularly scheduled formal group meetings, as well as proactively organizing and engaging in online discussions with Dr Mata, the project supervisor. Contribution to meetings has been strong from each individual and all discussions have been recorded in minutes and logged for reference.

Before deciding on an application for a simulated ROS-based project, the engineers have familiarised themselves with the software packages critical for the development through a small project to enhance a simulation of a single manually controlled Jetson robot in *Gazebo*. The group has here demonstrated excellent teamwork and problem-solving skills with engineers working in sub-teams on elements of the simulation such as sensor design, environment design, and robot control. This work has proved invaluable in determining an achievable scope, with the engineers very quickly reaching a consensus regarding objectives for system functionality. The success in undertaking this challenge has ensured that all engineers embarked on development with software properly configured, and with the

enhanced Jetson robot providing a solid base on which to design the more complex ACES system.

The initial project trajectory has been summarized in a kick-off presentation, solidifying the project aims and objectives for the team at an early stage in the project cycle and providing experience in collaborating effectively on an assignment. This has been built upon with the generation of an interim report, helping the group summarise the work achieved at the halfway stage of the project, consolidate research findings, defining clear goals for continuing implementation and testing of the proposed system.

Another strength displayed by the team during the early development of the project is a structured, collaborative approach to researching and reviewing the literature. This has involved a broad sweep of literature from reliable sources (IEEE Xplore, ArXiv, Glasgow Caledonian Library) with relevant material logged via *OneDrive*, proceeding with a more thorough analysis, and grading of relevant material within the project scope using a four-tier system. This approach has proved valuable in identifying useful resources for project development and avoided inefficiency through avoiding individuals tackling the same books and journal articles.

The most prominent challenge facing the team during the project has been the need to adapt to limited access to university facilities due to COVID-19 restrictions. As well as restricting experiments with hardware in the laboratory, this has forced the team to exclusively work remotely from home on all aspects of the project. The engineers have demonstrated a strong ability to adapt to these measures, employing a range of solutions for communication, programming, and reporting, as well as successfully setting ambitious project goals within the more restrictive scope imposed by these limitations.

Discord has served as the primary communication method between group members allowing the engineers to organize discussions regarding different areas of development into dedicated channels and maintain regular communication on project matters. This has been especially useful for efficient sharing of digital materials, as well as allowing clear communication during sections of the timeline requiring discussion of simultaneous development activities.

As well as providing means of hosting the project package, GitHub was used by the engineers for sharing of programming and version control during the development of the ACES system, facilitating the uploading and downloading of updated workspaces. This was found to provide efficient access to the most recent version of the simulation for continuing development, as well as a record of changes made throughout the project. The team's shared OneDrive folder was used extensively throughout the project to provide a log of formal meeting proceedings and as a means of sharing access to report documentation for collaborative working. While some challenges were experienced during the reporting phase due to connectivity issues the group's excellent organisation and communication skills have ensured a detailed account of the project timeline has been delivered, maximizing available time for development and experimentation.

The team has provided a summary of individual contribution to the project in *Appendix H*.

Chapter 8 CLOSING REMARKS

This report has presented the activities undertaken by the team in developing the Autonomous Collaborative Exploration Swarm (ACES) system, meeting the objectives laid out during the planning of the ROS-based Jetson robot solution required of this Masters project. An autonomous system for collaborative robotic map building and exploration of unknown indoor environments using ground-based units was built. The team conducted testing on the ACES system's accuracy, scalability, and response to environmental factors using software simulation, presenting a discussion of results.

All code and resources developed for the ACES system have been made available on the project GitHub [82]. This includes thorough documentation on the use and configuration of ACES, as well as an installation script to fully automate the setup of the ACES ROS nodes and *Gazebo* simulations. Example outputs for verification are also provided. It is hoped that this will represent a significant contribution to the growing ROS community and facilitate the development of future research projects.

Project planning, practical experimentation with software packages, and literary research carried out to date have been included in the report, accompanied by a description of development work carried out in the implementation of the system in *Gazebo* simulation. Future work is carefully considered, focusing on further development required for completion of the system and ensuring an experimentation strategy is prepared.

Analysis of the team's progress has been discussed, highlighting strong planning and teamwork used in familiarization of tools required for the project, as well as individual contributions made by the project's engineers. Challenges facing the team such as potential compatibility issues have been addressed in the report, with future work planned accordingly.

REFERENCES

[1] A. Mahtani, L. Sánchez, E. Fernández and A. Martinez, “ROS Architecture and Concepts,” Packt, 28 December 2016. [Online]. Available: https://hub.packtpub.com/ros-architecture-and-concepts/. [Accessed 13 April 2021].

[2] ROS, “ROS Documentation -- Introduction,” [Online]. Available: http://wiki.ros.org/ROS/Introduction. [Accessed 13 April 2021].

[3] ROS, “ROS Documentation -- Concepts,” [Online]. Available: http://wiki.ros.org/ROS/Concepts. [Accessed 13 April 2021].

[4] L. J. Rouse, S. J. Bergeron and T. M. Harris, “Participating in the Geospatial Web: Collaborative Mapping, Social Networks and Participatory GIS,” in *The Geospatial Web*, London, Springer, 2009, pp. 153 - 158.

[5] Humanitarian OpenStreetmap, “Hotosm.org,” Humanitarian OpenStreetmap Team, 2020. [Online]. Available: https://www.hotosm.org/. [Accessed 8 12 2020].

[6] Cision, “Transport for London enables a greener transportation future with Remix's collaborative mapping platform,” Cision, 14 10 2020. [Online]. Available: https://www.prnewswire.com/news-releases/transport-for-london-enables-a-greener-transportation-future-with-remixs-collaborative-mapping-platform-301151558.html.

[7] J. M. Klopp, “Busandcoach,” 7 2015. [Online]. Available: http://www.busandcoach.travel/download/digital_matatus.pdf. [Accessed 8 12 2020].

[8] C. C. Freifeld, R. Chunara, S. R. Mekaru, E. H. Chan, T. Kass-Hout, A. A. Iacucci and J. S. Brownstein, “Participatory epidemiology : Use of mobile phones for community-based health reporting.,” *PLoS Medicine,* vol. 7, no. 12, 2010.

[9] S. Fleischer, G. Payne, T. Kuhar, A. H. Jr, S. Malone, J. Whalen, G. Dively, D. Johnson, J. A. Hebberger, J. Ingerson-Mahar, K. Holmstrom and D. Miller., “Helicoverpa zea Trends from the Northeast: Suggestions Towards Collaborative Mapping of Migration and Pyrethroid Susceptibility,” in *Plant health progress*, Minneapolis, USA, 2018.

[10] I. Navarro and F. Matía, “An Introduction to Swarm Robotics,” *International Scholarly Research Notices,* vol. 2013, p. 10, 2013.

[11] G. Beni, “From Swarm Intelligence to Swarm Robotics,” in *Swarm Robotics, SAB 2004 International*, Santa Monica, USA, 2004.

[12] A. Singhal, P. Pallav, N. Kejriwal, S. Choudhury, S. Kumar and R. Sinha, “Managing a fleet of autonomous mobile robots (AMR) using cloud robotics platform,” in *European Conference on Mobile Robots (ECMR)*, Paris, France, 2017.

[13] B. Kehoe, S. Patil, P. Abbeel and K. Goldberg, “A Survey of Research on Cloud Robotics and Automation,” *IEEE Transactions on Automation Science and Engineering,* vol. 12, no. 2, pp. 398-409, 2015.

[14] G. Mohanarajah, D. Hunziker, R. D'Andrea and M. Waibel, “Rapyuta: A Cloud Robotics Platform,” *IEEE Transactions on Automation Science and Engineering,* vol. 12, no. 2, pp. 481-493, 2015.

[15] A. Ford, C. Raiciu, M. Handley, S. Barre and J. Iyengar, “Architectural guidelines for multipath tcp development,” London, 2011.

[16] G. Hu, W. P. Tay and Y. Wen, “Cloud robotics: architecture, challenges and applications,” *IEEE Network,* vol. 26, no. 3, pp. 21-28, 2012.

[17] T. Cieslewski, S. Choudhary and D. Scaramuzza, “Data-Efficient Decentralized Visual SLAM,” in *IEE International Conference*, 2018.

[18] W. Hess, D. Kohler, H. Rapp and D. Andor, “Cartogrepher Documentation,” 2016. [Online]. Available: https://google-cartographer.readthedocs.io/en/latest/.

[19] M. Korkmaz, A. Durdu and Y. Tusun, “Sensor Comparison for a Real-Time SLAM Application,” *International Journal of Information and Electronics Engineering,* vol. 8, pp. 1-4, 2018.

[20] T. Chong, X. Tang, C. Leng, M. Yogeswaran, O. Ng and Y. Chong, “Sensor Technologies and Simultaneous Localization and Mapping (SLAM),” in *2015 IEEE International Symposiumon Robotics and Intelligent Sensors*, 2015.

[21] X. Chen, H. Zhang, H. Lu, J. Xiao, Q. Qiu and Y. Li, “Robust SLAM system based on monocular vision and LiDAR for robotic urban search and rescue,” in *IEEE International Workshop on Safety, Security, and Rescue Robotics (SSRR)*, Shanghai, China, 2017.

[22] S. Kohlbrecher, O. v. Stryk, J. Meyer and U. Klingauf, “A flexible and scalable SLAM system with full 3D motion estimation,” in *IEEE International Workshop on Safety, Security, and Rescue Robotics (SSRR)*, Kyoto, Japan, 2011.

[23] G. Grisetti, C. Stachniss and W. Burgard, “Improved Techniques for Grid Mapping With Rao-Blackwellized Particle Filters,” *IEEE Transactions on Robotics,* vol. 23, pp. 34-46, 2007.

[24] C. Chen, H. Zhu, M. Li and S. You, “A Review of Visual-Inertial Simultaneous Localizationand Mapping from Filtering-Based andOptimization-Based Perspectives,” *Robotics,* 2018.

[25] D. S. Oliver Wasenmüller, “Comparison of Kinect V1 and V2 Depth Images in Terms of Accuracy and Precision,” in *Lecture notes in Computer Science*, 2016.

[26] C. Campos, R. Elvira, J. J. G. Rodríguez, J. M. M. Montiel and J. D. Tardós, “ORB-SLAM3: An Accurate Open-Source Library for Visual, Visual-Inertial and Multi-Map SLAM,” *arXiv pre-print,* 2020.

[27] K. Wang, C. Gou, Y. Duan, Y. Lin, X. Zheng and F.-Y. Wang, “Generative Adversarial Networks:Introduction and Outlook,” *IEEE/CAA Journal of Automatica Sinica,* vol. 4, no. 4, 2017.

[28] V. L. Alexander Vakhitov, “Learnable Line Segment Descriptor for Visual SLAM,” *IEEE Access,* vol. 7, 2019.

[29] T. S. A. J. L. Dinh-Cuong Hoang, “High-quality Instance-aware Semantic 3D Map Using RGB-D Camera,” Centre for Applied Autonomous Sensor Systems (AASS); Orebro University., 2019.

[30] J. G. K. K. S. N. J. K. Chao Liu, “Neural RGB->D Sensing: Depth and Uncertainty from a Video Camera,” in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[31] G. Dissanayake, S. Huang, Z. Wang and R. Ranasinghe, “A review of recent developments in simultaneous localization and mapping,” in *International Conference on Industrial and Information Systems*, 2011.

[32] W. X. K. H. L. K. Lan Hu, “Deep-SLAM++: Object-level RGBD SLAM based on class-specific deep shape priors,” in *arXiv preprint*, 2019.

[33] Q. Z. M. P. L. L.-T. Torsten Sattler, “Understanding the Limitations of CNN-based Absolute Camera Pose Regression,” in *Conference on Computer Vision and Pattern Recognition*, 2019.

[34] M. Quan, S. Piao, M. Tan and S.-S. Huang, “Tightly-coupled Monocular Visual-odometric SLAM using Wheels and a MEMS Gyroscope,” *IEEE Access,* vol. 7, 2019.

[35] L. Zhao, Z. Liu, J. Chen, W. Cai, W. Wang and L. Zeng, “A Compatible Framework for RGB-D SLAM in Dynamic Scenes,” *IEEE Access,* vol. 7, pp. 75604 - 75614, 2019.

[36] B. W. James Garforth, “Visual Appearance Analysis of Forest Scenes for Monocular SLAM,” *arXiv pre-print,* 2019.

[37] J. H. S. Liu, “Robust simultaneous localization and mapping in low-light environment,” *Computer Animation and Virtual Worlds,* vol. 30, no. 3, 2019.

[38] J. S. Michael Strecke, “EM-Fusion: Dynamic Object-Level SLAM with Probabilistic Data Association,” in *IEEE/CVF International Conference on Computer Vision 2019*, 2019.

[39] A. Kasar, “Benchmarking and Comparing Popular Visual SLAM Algorithms,” Department of Electronics and Telecommunication Engineering, Pune Institute of Computer Technology, 2018.

[40] B. K. P. Horn, “Closed-form solution of absolute orientation using unit quaternions,” *Optical Society of America,* vol. 4, no. 4, 1987.

[41] J. Sturm, N. Engelhard, F. Endres, W. Burgard and D. Cremers, “A benchmark for the evaluation of RGB-D SLAM systems,” in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2012.

[42] B. S. C. D. M. R. G. G. C. S. A. K. Rainer Kümmerle, “On Measuring the Accuracy of SLAM Algorithms,” *Autonomous Robots,* vol. 27, 2009.

[43] A. Geiger, P. Lenz, C. Stiller and R. Urtasun, “Vision meets Robotics: The KITTI Dataset,” *International Journal of Robotics Research,* 2013.

[44] J. Jeong, Y. Cho, Y.-S. Shin, H. Roh and A. Kim, “Complex urban dataset with multi-level sensors from highly diverse urban environments,” *The International Journal of Robotics Research,* 2019.

[45] J. R. M. R. M. E. Gaurav Pandey, “Ford Campus Vision and Lidar Data Set,” Ford Motor Company Research, Dearborn, MI, 2009.

[46] T. Peynot, S. Scheding and S. Terho, “The Marulan Data Sets: Multi-Sensor Perception in Natural Environment with Challenging Conditions,” *International Journal of Robotics Research,* vol. 29, pp. 1602-1607, 2010.

[47] S. Golodetz, T. Cavallari, N. A. Lord, V. A. Prisacariu, D. W. Murray and P. H. S. Torr, “Collaborative Large-Scale Dense 3D Reconstruction with Online Inter-Agent Pose Optimisation,” *TVCG (ISMAR Special Issue),* 2018.

[48] N. Chebrolu, P. Lottes, A. Schaefer, W. Winterhalter, W. Burgard and C. Stachniss, “Agricultural robot dataset for plant classification, localization and mapping on sugar beet fields,” *The International Journal of Robotics Research,* 2017.

[49] J. R. Ruiz-Sarmiento, C. Galindo and J. Gonzalez-Jimenez, “Robot@Home, a Robotic Dataset for Semantic Mapping of Home Environments,” *International Journal of Robotics Research,* 2017.

[50] O. Wasenmuller, M. Meyer and D. Stricker, “CoRBS: Comprehensive RGB-D Benchmark for SLAM using Kinect v2,” in *IEEE Winter Conference on Applications of Computer Vision*, 2016.

[51] J. Sturm, N. Engelhard, F. Endres, W. Burgard and D. Cremers, “A Benchmark for the Evaluation of RGB-D SLAM Systems,” in *International Conference on Intelligent Robot Systems*, 2012.

[52] A. Handa, T. Whelan, J. McDonald and A. Davison, “A Benchmark for RGB-D Visual Odometry, 3D Reconstruction and SLAM,” in *IEEE Intl. Conf. on Robotics and Automation*, 2014.

[53] E. D. C. C. W. L. D. T. Sajad Saeedi, “Characterizing Visual Localization and Mapping Datasets,” in *International Conference on Robotics and Automation*, 2019.

[54] J. McCormac, A. Handa, S. Leutenegger and A. J.Davison, “SceneNet RGB-D: Can 5M Synthetic Images Beat Generic ImageNet Pre-training on Indoor Segmentation?,” *ICCV,* 2017.

[55] W. Li, S. Saeedi, J. McCormac, R. Clark, D. Tzoumanikas, Q. Ye, Y. Huang, R. Tang and S. Leutenegger, “InteriorNet: Mega-scale Multi-sensor Photo-realistic Indoor Scenes Dataset,” in *British Machine Vision Conference*, 2018.

[56] A. Bokovoy, K. Muraviev and K. Yakovlev, “Map-merging Algorithms for Visual SLAM: Feasibility Study and Empirical Evaluation,” in *Russian Conference on Artificial Intelligence*, 2020.

[57] J. Horner, “map_merge_3d ROS wiki,” 20 June 2018. [Online]. Available: http://wiki.ros.org/map_merge_3d. [Accessed 15 December 2020].

[58] P.-Y. Lajoie, B. Ramtoula, Y. Chang, L. Carlone and G. Beltrame, “DOOR-SLAM: Distributed, Online, and Outlier Resilient SLAM for Robotic Teams,” arXiv, 2019.

[59] M. Chamanbaz, D. Mateo, B. M. Zoss, G. Tokic, E. Wilhelm, R. Bouffanais and D. K. P. Yue, “Swarm-Enabling Technology for Multi-Robot Systems,” *Frontiers,* 2017.

[60] M. Garzon, J. Valente, J. J. Roldán, D. Garzón-Ramos, J. de León, A. Barrientosand and J. del Cerro, Robot Operating System (ROS) - The Complete Reference, vol. 2, A. Koubaa, Ed., AG: Springer, 2017, p. 452.

[61] R. Ramachandran, Z. Kakish and S. Berman, “Information correlated L'evy walk exploration and distributed mapping using a swarm of robots,” University of Southern California, LA, 2019.

[62] M. Kegeleirs, “Random Walk Exploration For Swarm Mapping,” IRIDIA, (Institut de Recherches Interdisciplinaireset de D´eveloppements en Intelligence Artificielle), Brussels, 2019.

[63] G. Arpino, K. Morris, S. Nagavalli and K. Sycara, “Using Information Invariants to Compare Swarm Algorithms and General Multi-Robot Algorithms,” in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, Brisbane, 2018.

[64] S. Lavalle, “Rapidly-Exploring Random Trees: A New Tool for Path Planning,” Department of Computer Science, Iowa State University, Ames, 1998.

[65] H. Umari, “ROS documentation - rrt_exploration,” ROS, 2018. [Online]. Available: http://wiki.ros.org/rrt_exploration. [Accessed 2020].

[66] J. Horner, “ROS documentation - explore_lite,” ROS, 2017. [Online]. Available: http://wiki.ros.org/explore_lite. [Accessed 2021].

[67] S. Koenig and C. ,. H. W. Tovey, “Greedy Mapping of Terrain,” in *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2001.

[68] C. DuHadway and B. Pitzer, “ROS documentation - explore,” ROS, 2010. [Online]. Available: http://wiki.ros.org/explore. [Accessed 2021].

[69] P. Bovbel, “ROS documentation - frontier_exploration,” ROS, 2017. [Online]. Available: http://wiki.ros.org/frontier_exploration. [Accessed 2021].

[70] D. Claes and K. Tuyls, “Multi robot collision avoidance in a shared workspace,” *Autonomous Robots,* vol. 42, no. 8, pp. 1749-1770, 2018.

[71] S. Bultmann, L. Matignon and O. Simonin, “Multi-Robot Navigation and CooperativeMapping in a Circular Topology,” INSA, Lyon, 2017.

[72] D. Balkcom and M. Mason, “Time Optimal Trajectories for Bounded Velocity Differential Drive Robots,”,” in *IEEE International Conference on Robotics and Automation (ICRA)*, San Francisco, USA, 2000.

[73] M. W. Choi, J. S. Park, B. S. Lee and M. H. Lee, “The performance of independent wheels steering vehicle(4WS) applied Ackerman geometry,” in *Control, Automation and Systems*, Seoul, Korea, 2008.

[74] S. Tang, X. Li and Y. Zhang, “Effect of Body Rolling of Skid-Steering Wheeled Vehicle on Steering Characteristics,” in *IEEE Conference and Expo Transportation Electrification Asia-Pacific (ITEC Asia-Pacific)*, Beijing, China, 2014.

[75] L. H. Prayudhi, A. Widyotriatmo and K.-S. Hong, “Wall following control algorithm for a car-like wheeled-mobile robot with differential-wheels drive,” in *15th International Conference on Control, Automation and Systems (ICCAS)*, Busan, South Korea, 2015.

[76] Robodyne SRL, “TurtleBot3 Datasheet,” 2020. [Online]. Available: https://www.robodyne-services.com/turtlebot-3/. [Accessed 2020].

[77] Nvidia, “Nvidia.com,” Nvidia, 2020. [Online]. Available: https://developer.nvidia.com/embedded/jetson-nano-developer-kit. [Accessed 9 12 2020].

[78] STMicroelectronics, “STM32 32-bit Arm Cortex MCU Documentation,” 2021. [Online]. Available: https://www.st.com/en/microcontrollers-microprocessors/stm32-32-bit-arm-cortex-mcus.html. [Accessed 2021].

[79] Robotis, “Dynamixel Servomotor Data Sheet,” 2021. [Online]. Available: https://emanual.robotis.com/docs/en/platform/turtlebot3/appendixes/. [Accessed 2021].

[80] Waveshare, “AC8265 Wireless NIC for Jetson Nano Documentation,” 2020. [Online]. Available: https://www.waveshare.com/wireless-ac8265.htm. [Accessed 2020].

[81] ROS, “ROS Documentation,” ROS, 2020. [Online]. Available: https://www.ros.org/. [Accessed 12 12 2020].

[82] M. Alvarez, L. Byrne, J. Hardie, K. Shabkhez and R. Walker, “ACES Github,” 2020. [Online]. Available: https://github.com/Luke-Byrne/ACES.

[83] ravyburn, “ORB-SLAM2_ROS Github,” [Online]. Available: https://github.com/rayvburn/ORB-SLAM2_ROS. [Accessed 2021].

[84] A. A. Initiative, “ORB_SLAM_2_ROS Github,” [Online]. Available: https://github.com/appliedAI-Initiative/orb_slam_2_ros. [Accessed 2021].

[85] U.-S. Lab, “ORB_SLAM3 Github,” [Online]. Available: https://github.com/UZ-SLAMLab/ORB_SLAM3. [Accessed 2021].

[86] R. Mur-Artal and J. D. Tardos, “ORB-SLAM2: an Open-Source SLAM System for Monocular, Stereo and RGB-D Cameras,” *IEEE Transactions on Robotics,* vol. 33, no. 5, pp. 1255-1262, 2017.

[87] E. Rublee, V. Rabaud, K. Konolige and G. Bradski, “ORB: an efficient alternative to SIFT or SURF,” in *International Conference on Computer Vision*, 2011.

[88] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss and W. Burgard, “OctoMap: an efficient probabilistic 3D mapping framework based on octrees,” *Autonomous Robots,* vol. 34, pp. 189-206, 2013.

[89] A. H. Wolfgang Merkt, “OctoMap Server: ROS Wiki,” [Online]. Available: http://wiki.ros.org/octomap_server. [Accessed 12 April 2021].

[90] D. Meagher, “Geometric Modeling Using Octree-Encoding,” *Computer Graphics and Image Processing,* vol. 19, pp. 129-147, 1982.

[91] J. Wilhelms and A. V. Gelder, “Octrees for Faster Isosurface Generation,” *ACM Transactions on Graphics,* vol. 11, no. 3, 1992.

[92] ROS, “Setup and Configuration of the Navigation Stack on a Robot,” 2021. [Online]. Available: http://wiki.ros.org/navigation/Tutorials/RobotSetup. [Accessed 2020].

[93] M. F. David Lu, “ROS Wiki: global_planner,” [Online]. Available: http://wiki.ros.org/global_planner. [Accessed 14 April 2021].

[94] D. L. A. H. E. M.-E. E. P. Michael Ferguson, “ROS Wiki: base_local_planner,” [Online]. Available: http://wiki.ros.org/base_local_planner. [Accessed 14 Aril 2021].

[95] B. Gerkey, “ROS Documentation - amcl,” 2021. [Online]. Available: http://wiki.ros.org/amcl. [Accessed 2020].

[96] J. Horner, “multirobot_map_merge: ROS Wiki,” [Online]. Available: http://wiki.ros.org/multirobot_map_merge. [Accessed 12 04 2021].

[97] J. Horner, “map_merge_3d: ROS Wiki,” [Online]. Available: http://wiki.ros.org/map_merge_3d. [Accessed 12 04 2021].

[98] R. W. M. A. J. H. K. S. Luke Byrne, “ACES Install Script,” 2021. [Online]. Available: https://github.com/Luke-Byrne-uni/ACES/blob/main/ACES_install_melodic.sh.

[99] “Velodynelidar.com,” Velodynelidar, [Online]. Available: https://velodynelidar.com/. [Accessed 2 March 2021].

[100] “ROS Documentation - Map saver,” ROS, [Online]. Available: http://wiki.ros.org/map_server . [Accessed 5 February 2021].

[101] P. Fankhauser, M. Bloesch and M. Hutter, “Probabilistic Terrain Mapping for Mobile Robots with Uncertain Localization,” *IEEE Robotics and Automation Letters,* vol. 3, no. 4, pp. 3019-3026, 2018.

[102] H. Umari and S. Mukhopadhyay, “Autonomous Robotic Exploration Based on Multiple Rapidly-exploring Randomized Trees,” *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS),* pp. 1396-1402, 2017.

[103] D. Ghorpade, A. D. Thakare and S. Doiphode, “Obstacle Detection and Avoidance Algorithm for Autonomous Mobile Robot using 2D LiDAR,” in *International Conference on Computing Communication Control and Automation (ICCUBEA)*, Pune, India, 2017.

[104] D. Jost, “FierceElectronics,” 11 6 2019. [Online]. Available: https://www.fierceelectronics.com/sensors/what-accelerometer. [Accessed 10 12 2020].

[105] B. Mustapha, A. Zayegh and R. K. Begg, “Ultrasonic and Infrared Sensors Performance in a Wireless Obstacle Detection System,” in *International Conference on Artificial Intelligence, Modelling and Simulation (AIMS)*, Kota Kinabalu, Malaysia, 2013.

[106] T. Mohammad, “Using Ultrasonic and Infrared Sensors for Distance Measurement,” in *World Academy of Science, Engineering and Technology*, Hong Kong, 2009.

[107] O. A. Hasan, A. T. Rashid, R. S. Ali and J. Kosha, “A practical performance analysis of low-cost sensors for indoor localization of multi-node systems,” in *Internet Technologies and Applications (ITA)*, Wrexham, UK, 2017.

[108] E.B., “S-Lab,” 3 8 2009. [Online]. Available: https://slab.concordia.ca/arduino/ir-range-finder. [Accessed 12 12 2020].

[109] Components101, “Components101,” 18 10 2017. [Online]. Available: https://components101.com/ultrasonic-sensor-working-pinout-datasheet. [Accessed 12 12 2020].

[110] SparkFun, “Gyroscope Tutorial,” SparkFun, 2020. [Online]. Available: https://learn.sparkfun.com/tutorials/gyroscope/all. [Accessed 9 12 2020].

[111] Gazebosim.org, “ROS Control,” Open Source Robotics Foundation, 2014. [Online]. Available:

http://gazebosim.org/tutorials?tut=ros_control&cat=connect_ros#Aboutros_control. [Accessed 01 12 2020].

[112] E. Erős, M. Dahl, K. Bengtsson, A. Hanna and P. Falkman, "A ROS2 based communication architecture for control in collaborative and intelligent automation systems," in *29th International Conference on Flexible Automation and Intelligent Manufacturing*, Limerick, Ireland, 2019.

[113] Blender, "Blender," Blender, 2020. [Online]. Available: https://www.blender.org/. [Accessed 9 12 2020].

[114] PTC, "PTC.com," PTC, 2020. [Online]. Available: https://www.ptc.com/en/products/creo#. [Accessed 7 12 2020].

[115] Autodesk, "Autodesk.co.uk," Autodesk, 2020. [Online]. Available: https://www.autodesk.co.uk/products/fusion-360/overview. [Accessed 6 12 2020].

[116] ROS.org, "Setup and Configuration of the Navigation Stack on a Robot," Open Source Robotics Foundation, 2020. [Online]. Available: http://wiki.ros.org/navigation/Tutorials/RobotSetup. [Accessed 01 12 2020].

[117] A. Tiderko, "ROS Documentation - master_discovery_fkie," ROS, 2020. [Online]. Available: http://wiki.ros.org/master_discovery_fkie. [Accessed 10 12 2020].

[118] L. Walkter, "ROS documentation - roscore," ROS, 2020. [Online]. Available: http://wiki.ros.org/roscore. [Accessed 12 12 2020].

[119] A. Tiderko, "ROS documentation - master_sync_fkie," ROS, 2020. [Online]. Available: http://wiki.ros.org/master_sync_fkie. [Accessed 11 12 2020].

[120] E. Marder-Eppstein, "ROS documentation - move_base," ROS, 2020. [Online]. Available: http://wiki.ros.org/move_base. [Accessed 2021].

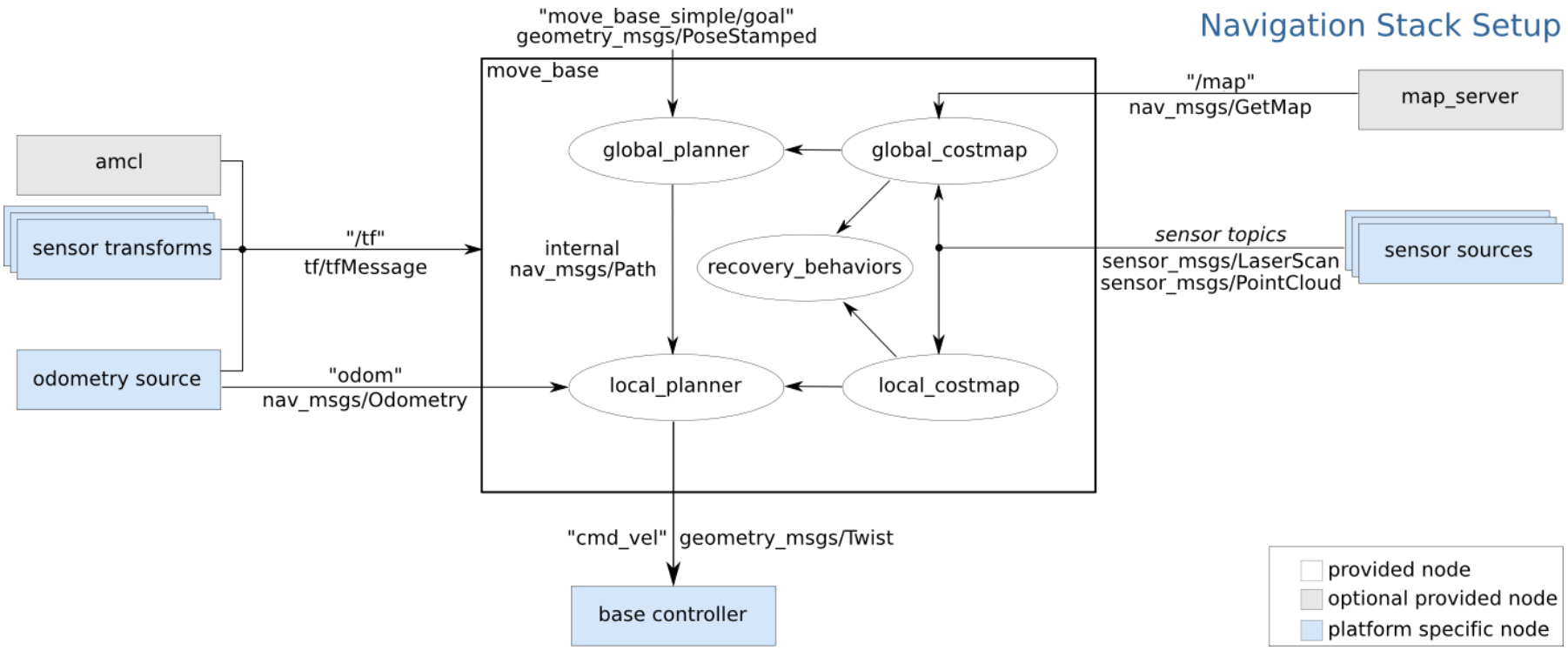
[121] M. Ferguson, D. Lu and A. Hoy, "ROS documentation - navigation," ROS, 2020. [Online]. Available: http://wiki.ros.org/navigation. [Accessed 2021].

[122] A. Hendrix, “ROS documentation - teleop_twist_keyboard,” ROS, 2015. [Online]. Available: http://wiki.ros.org/teleop_twist_keyboard. [Accessed 2020].

[123] ROS, “ROS/Concepts,” 2021. [Online]. Available: http://wiki.ros.org/ROS/Concepts. [Accessed 2021].

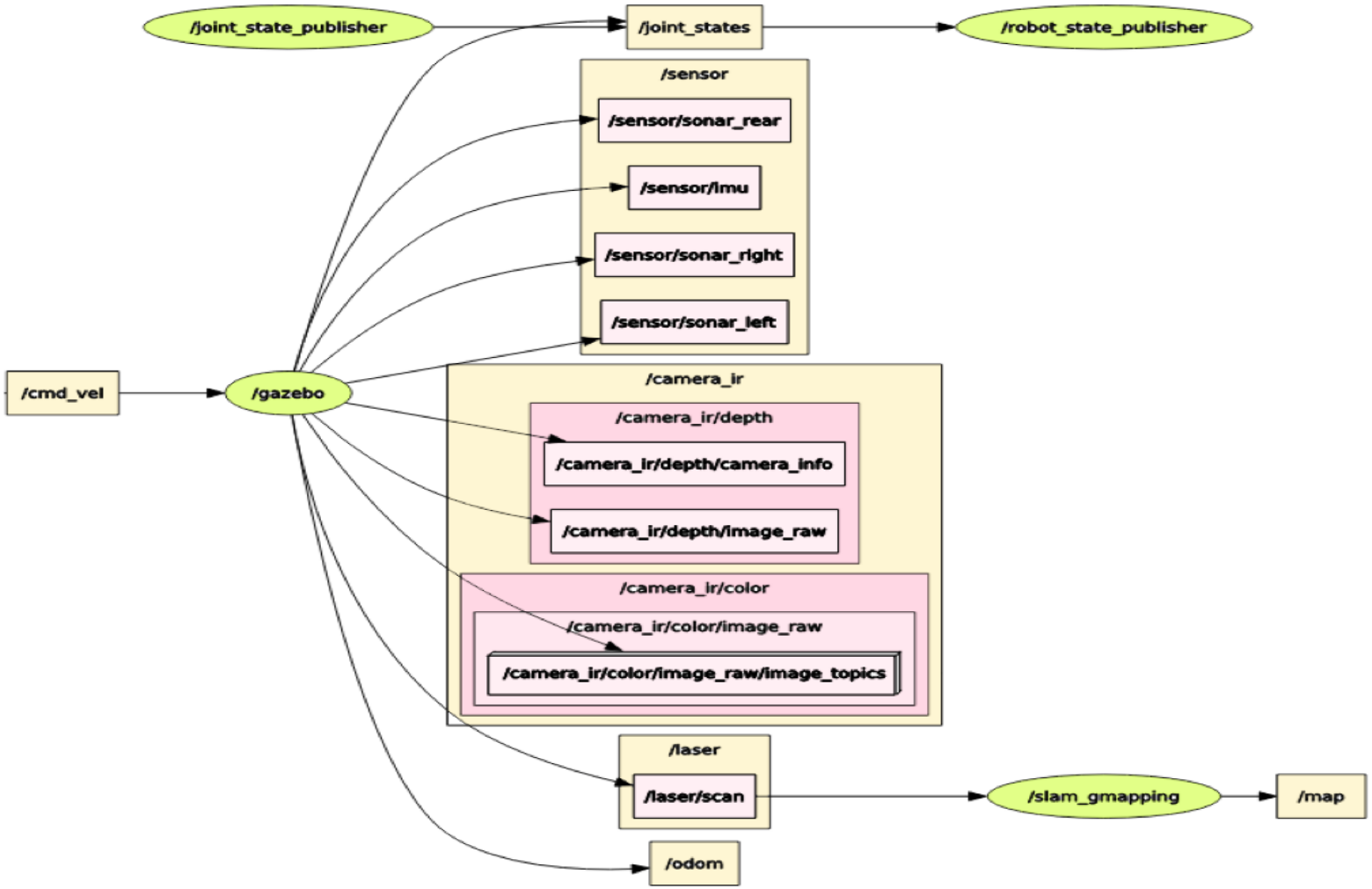
APPENDIX

Appendix A: ROS NAVIGATION STACK SETUP



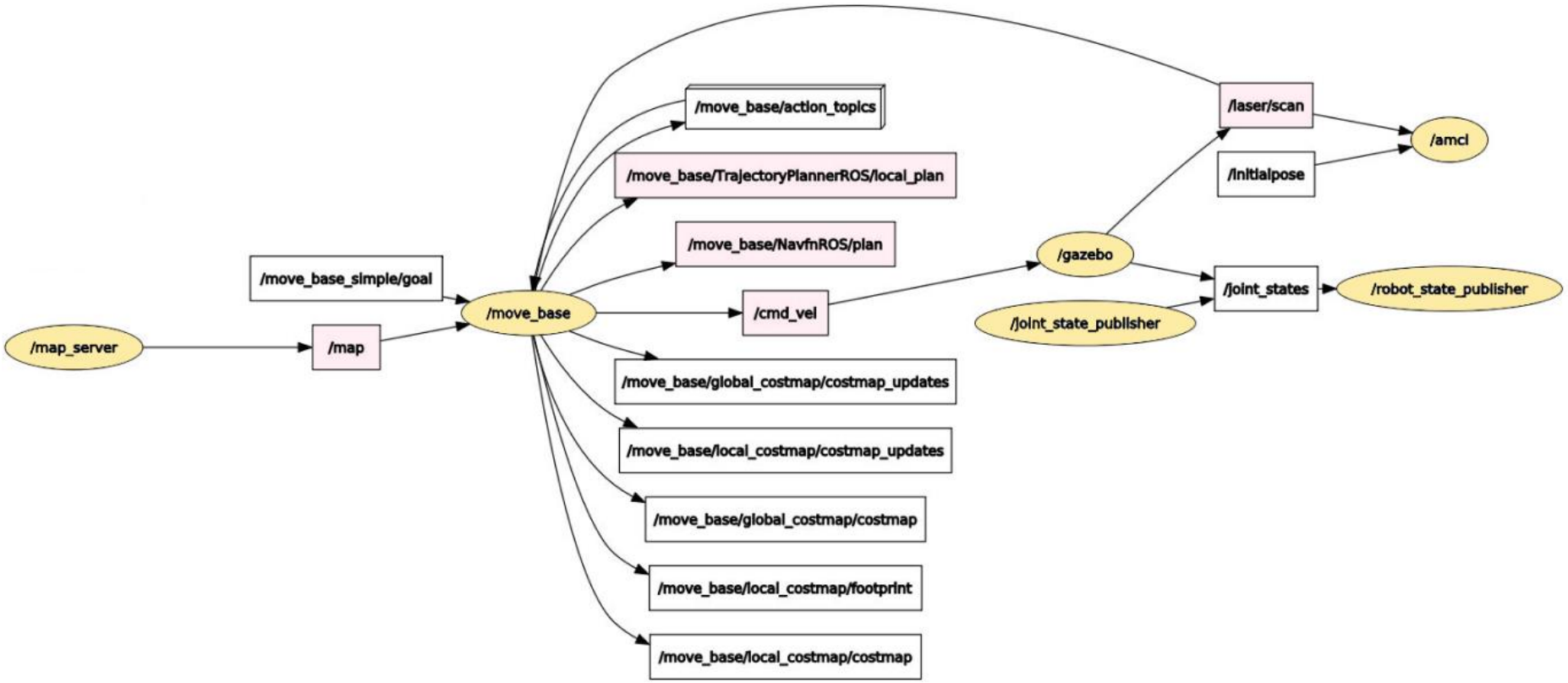
Appendix Figure A-1 – ROS Navigation Stack [97]

Appendix B: TOPICS & NODES FOR ACES ROBOT



Appendix Figure B-1 – ACES Robot Topics and Nodes

Appendix C: TOPICS & NODES FOR AUTONOMOUS NAVIGATION



Appendix Figure C-1 – Topics & Nodes for Autonomous Navigation

Appendix D: TERMINAL VIEW OF MULTI-ROBOT CONFIGURATION

robot_1

```
/robot_1/camera_ir/color/camera_info
/robot_1/camera_ir/color/image_raw
/robot_1/camera_ir/color/image_raw/compressed
/robot_1/camera_ir/color/image_raw/compressed/parameter_descriptions
/robot_1/camera_ir/color/image_raw/compressed/parameter_updates
/robot_1/camera_ir/color/image_raw/compressedDepth
/robot_1/camera_ir/color/image_raw/compressedDepth/parameter_descriptions
/robot_1/camera_ir/color/image_raw/compressedDepth/parameter_updates
/robot_1/camera_ir/color/image_raw/theora
/robot_1/camera_ir/color/image_raw/theora/parameter_descriptions
/robot_1/camera_ir/color/image_raw/theora/parameter_updates
/robot_1/camera_ir/depth/camera_info
/robot_1/camera_ir/depth/image_raw
/robot_1/camera_ir/depth/points
/robot_1/camera_ir/parameter_descriptions
/robot_1/camera_ir/parameter_updates
/robot_1/cmd_vel
/robot_1/joint_states
/robot_1/laser/scan
/robot_1/map
/robot_1/map_updates
/robot_1/move_base/NavfnROS/plan
/robot_1/move_base/TrajectoryPlannerROS/local_plan
/robot_1/odom
/robot_1/sensor/imu
/robot_1/sensor/sonar_left
/robot_1/sensor/sonar_rear
/robot_1/sensor/sonar_right
```

Appendix Figure D-1 – robot_1 Namespace

robot_2

```
/robot_2/camera_ir/color/camera_info
/robot_2/camera_ir/color/image_raw
/robot_2/camera_ir/color/image_raw/compressed
/robot_2/camera_ir/color/image_raw/compressed/parameter_descriptions
/robot_2/camera_ir/color/image_raw/compressed/parameter_updates
/robot_2/camera_ir/color/image_raw/compressedDepth
/robot_2/camera_ir/color/image_raw/compressedDepth/parameter_descriptions
/robot_2/camera_ir/color/image_raw/compressedDepth/parameter_updates
/robot_2/camera_ir/color/image_raw/theora
/robot_2/camera_ir/color/image_raw/theora/parameter_descriptions
/robot_2/camera_ir/color/image_raw/theora/parameter_updates
/robot_2/camera_ir/depth/camera_info
/robot_2/camera_ir/depth/image_raw
/robot_2/camera_ir/depth/points
/robot_2/camera_ir/parameter_descriptions
/robot_2/camera_ir/parameter_updates
/robot_2/joint_states
/robot_2/laser/scan
/robot_2/map
/robot_2/map_updates
/robot_2/move_base/NavfnROS/plan
/robot_2/move_base/TrajectoryPlannerROS/local_plan
/robot_2/sensor/imu
/robot_2/sensor/sonar_left
/robot_2/sensor/sonar_rear
/robot_2/sensor/sonar_right
```

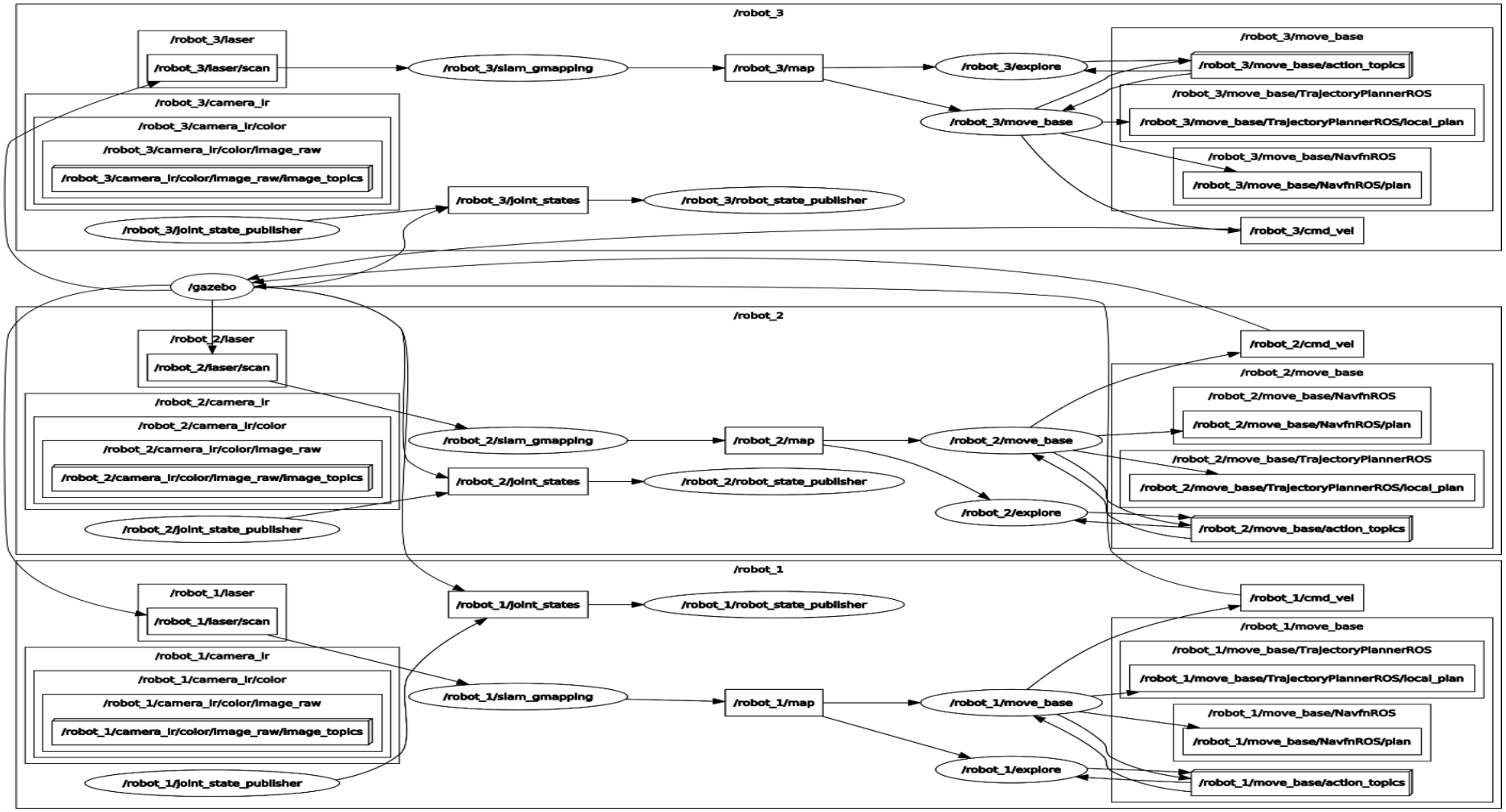
Appendix Figure D-2 – robot_2 Namespace

robot_3

```
/robot_3/camera_ir/color/camera_info
/robot_3/camera_ir/color/image_raw
/robot_3/camera_ir/color/image_raw/compressed
/robot_3/camera_ir/color/image_raw/compressed/parameter_descriptions
/robot_3/camera_ir/color/image_raw/compressed/parameter_updates
/robot_3/camera_ir/color/image_raw/compressedDepth
/robot_3/camera_ir/color/image_raw/compressedDepth/parameter_descriptions
/robot_3/camera_ir/color/image_raw/compressedDepth/parameter_updates
/robot_3/camera_ir/color/image_raw/theora
/robot_3/camera_ir/color/image_raw/theora/parameter_descriptions
/robot_3/camera_ir/color/image_raw/theora/parameter_updates
/robot_3/camera_ir/depth/camera_info
/robot_3/camera_ir/depth/image_raw
/robot_3/camera_ir/depth/points
/robot_3/camera_ir/parameter_descriptions
/robot_3/camera_ir/parameter_updates
/robot_3/cmd_vel
/robot_3/joint_states
/robot_3/laser/scan
/robot_3/map
/robot_3/map_updates
/robot_3/move_base/NavfnROS/plan
/robot_3/move_base/TrajectoryPlannerROS/local_plan
/robot_3/odom
/robot_3/sensor/imu
/robot_3/sensor/sonar_left
/robot_3/sensor/sonar_rear
/robot_3/sensor/sonar_right
```

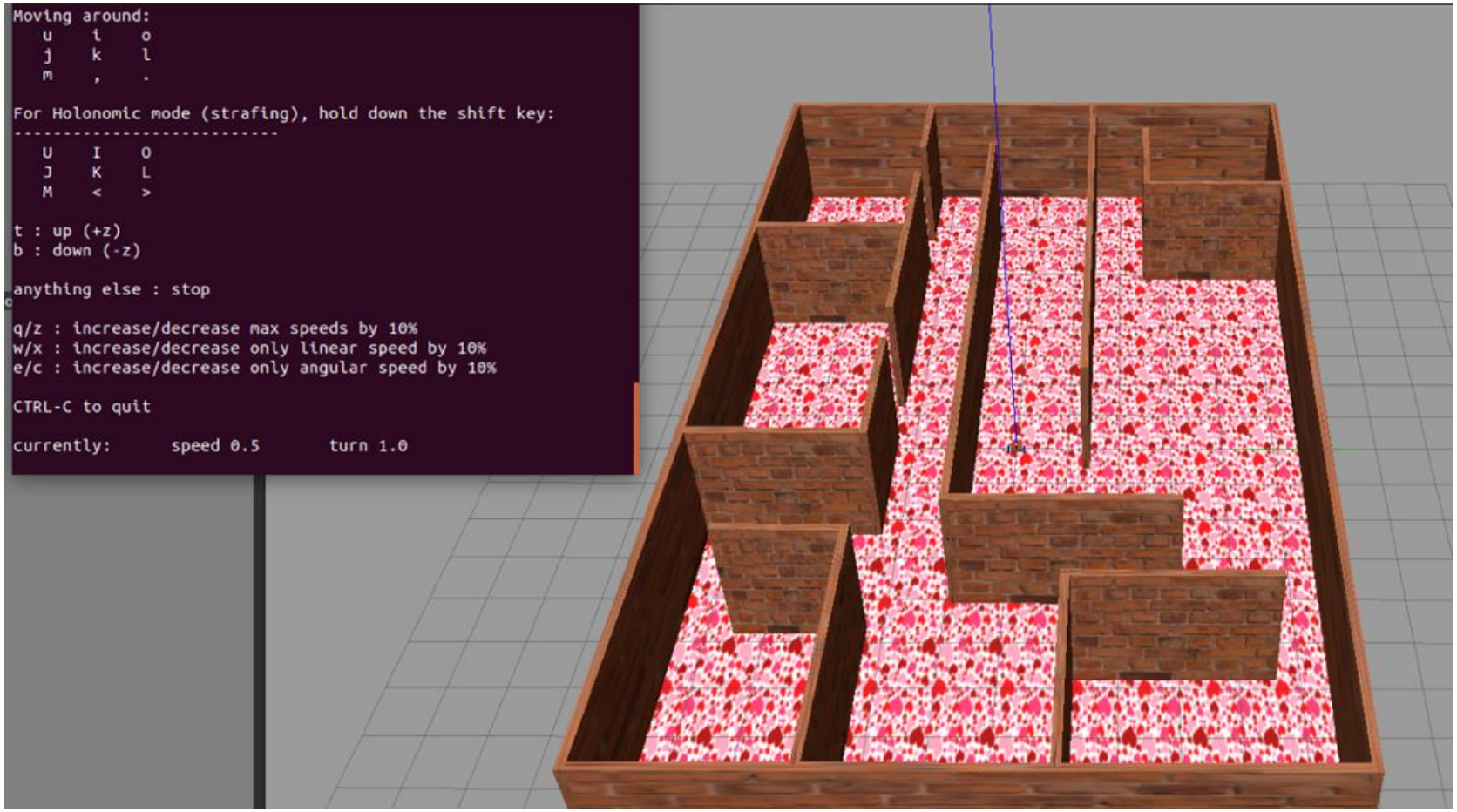
Appendix Figure D-3 – robot_3 Namespace

Appendix E: **2D ACES SYSTEM ARCHITECTURE**

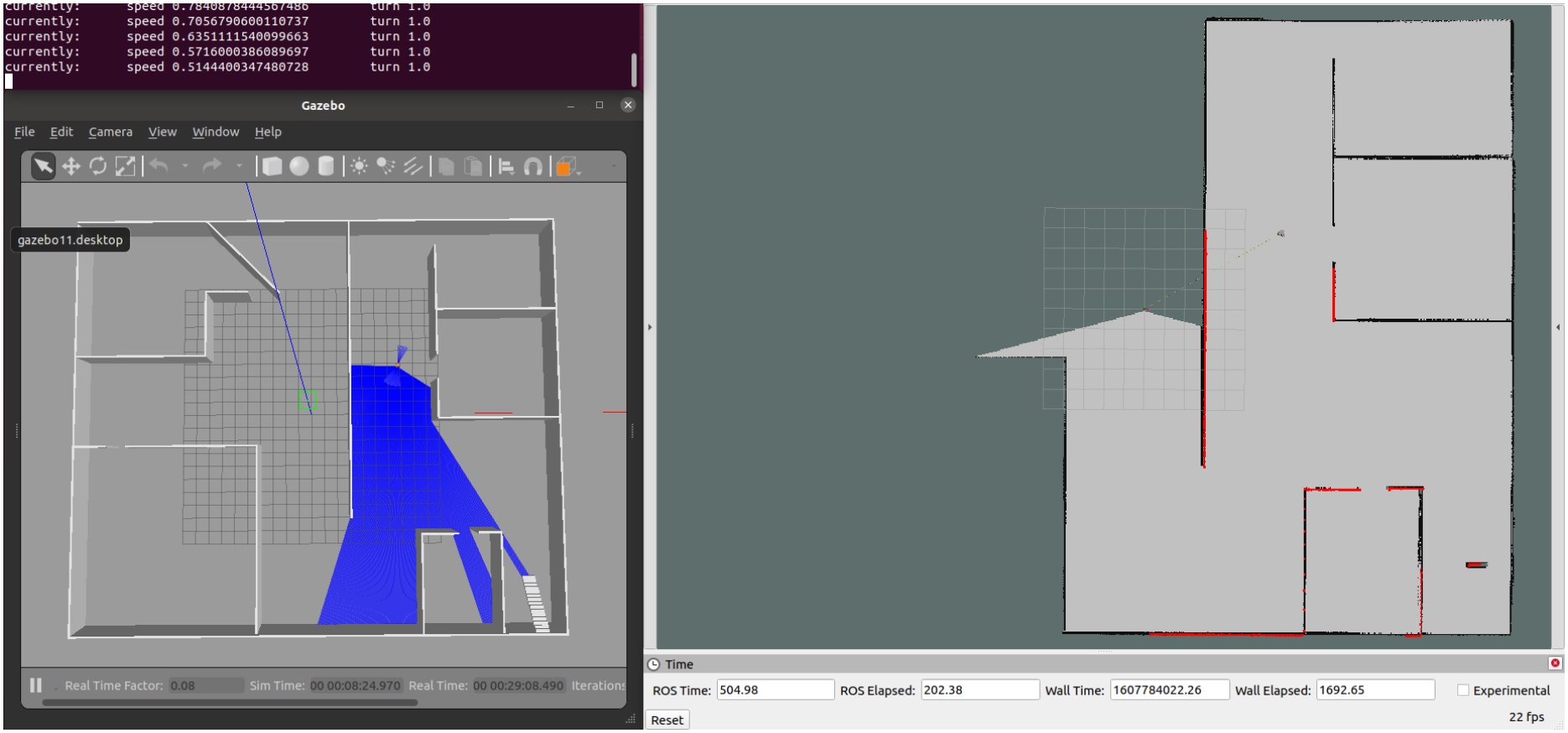


Appendix Figure E-1 – Architecture of 2D ACES System

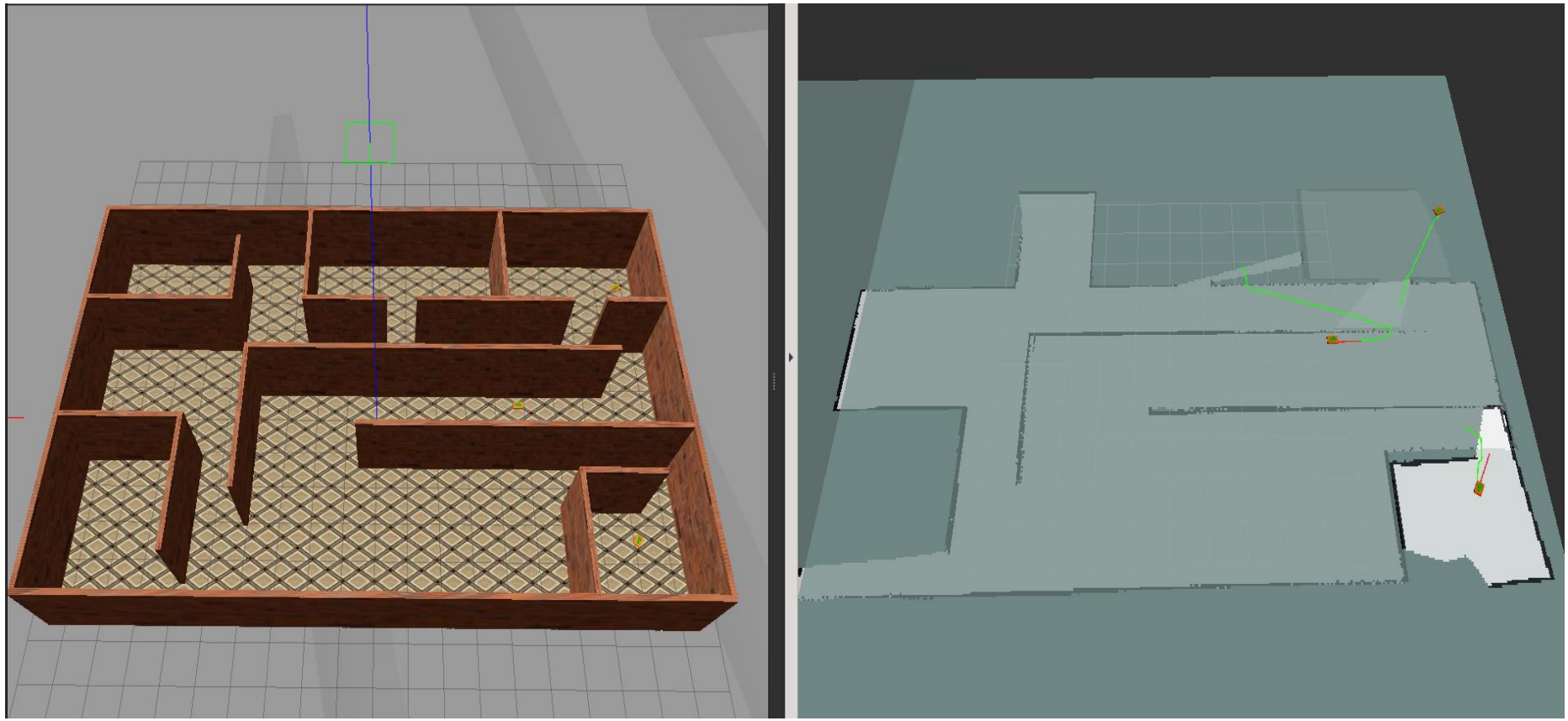
Appendix F: SYSTEM VALIDATION – GAZEBO SIMULATION & VISUALIZING IN RVIZ



Appendix Figure F-1 – Manually Controlled Mobility Validation - Map 2



Appendix Figure F-2 – Map Creation visualized in Gazebo & Rviz



Appendix Figure F-3 – Multi-Robot Exploration visualized in Gazebo & RViz

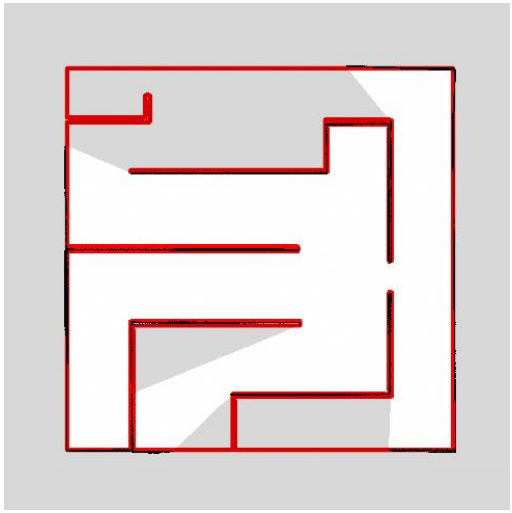
Appendix G: **2D RESULTS**

ACCURACY VS VELOCITY

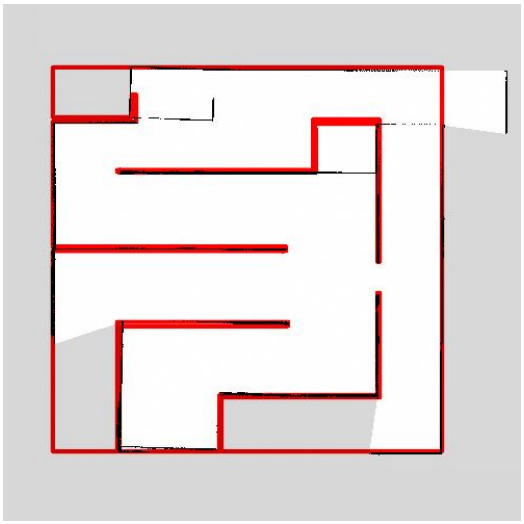
The real map is shown in red, overlayed on the map produced during testing.

Map 1

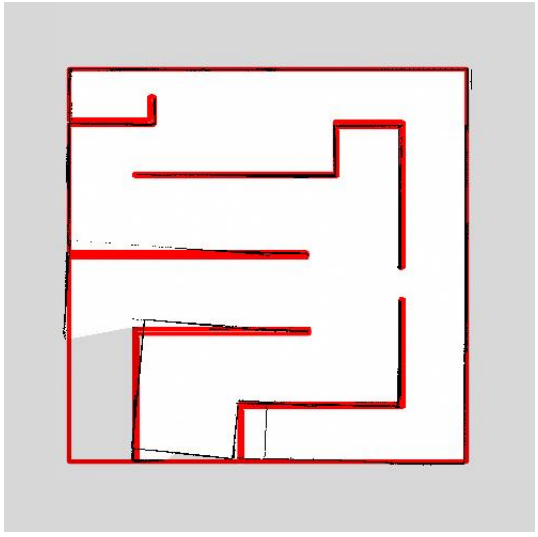
Office – One Robot – 0.1 m/s



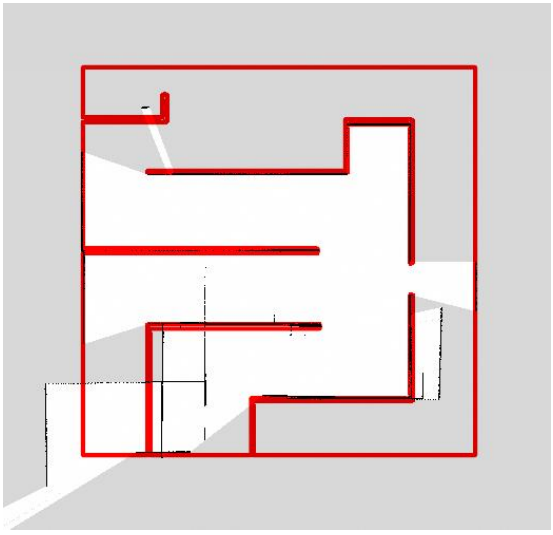
Office – One Robot – 0.3 m/s



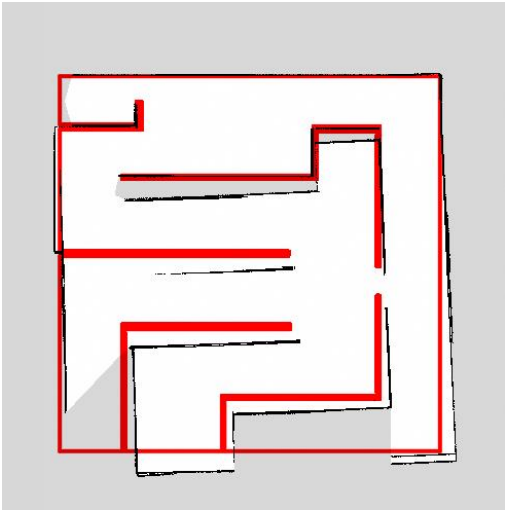
Office – One Robot – 0.5 m/s



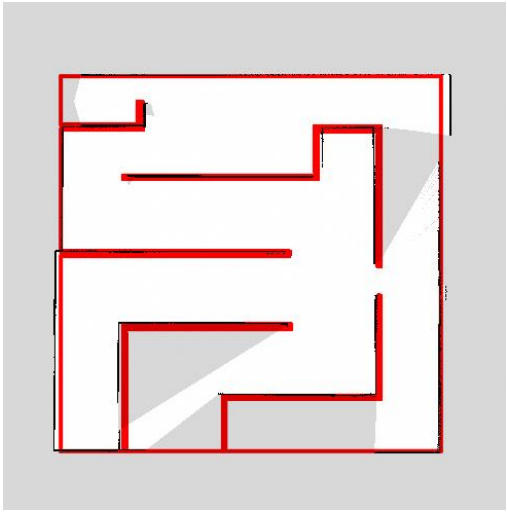
Office – One Robot – 1.0 m/s



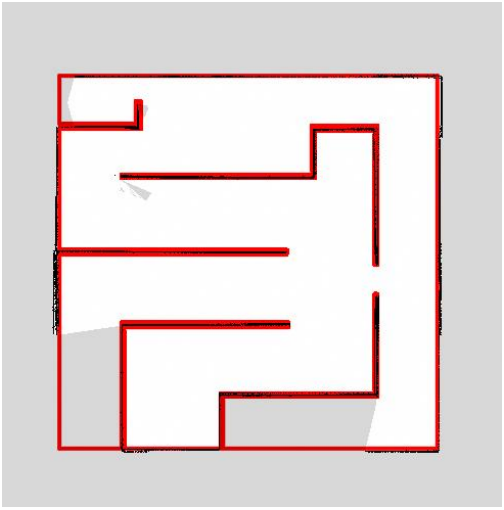
Office – Two Robots – 0.1 m/s



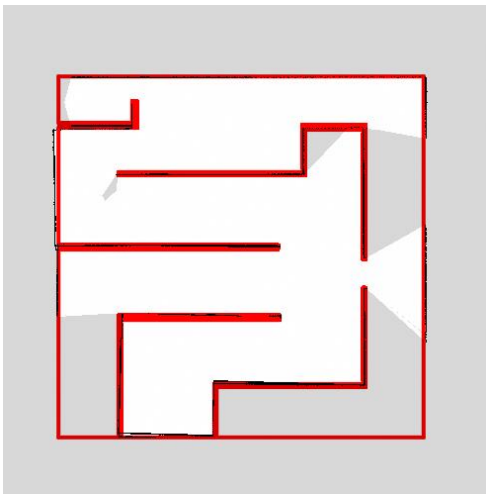
Office – Two Robots – 0.3 m/s



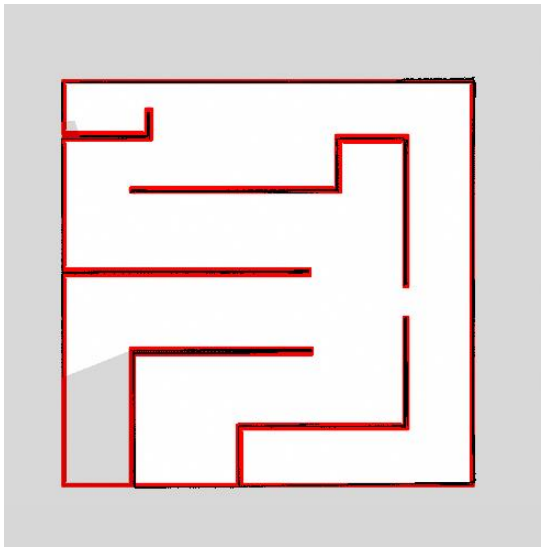
Office – Two Robots – 0.5 m/s



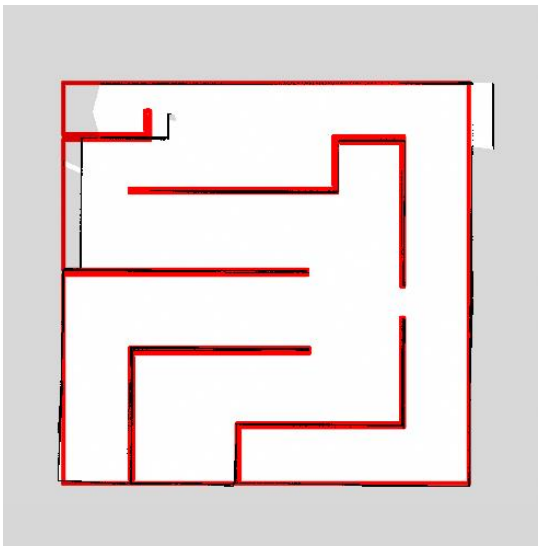
Office – Two Robots – 1.0 m/s



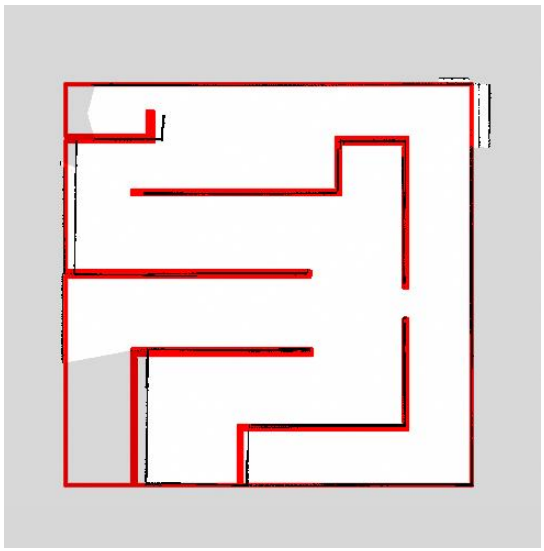
Office – Three Robots – 0.1 m/s



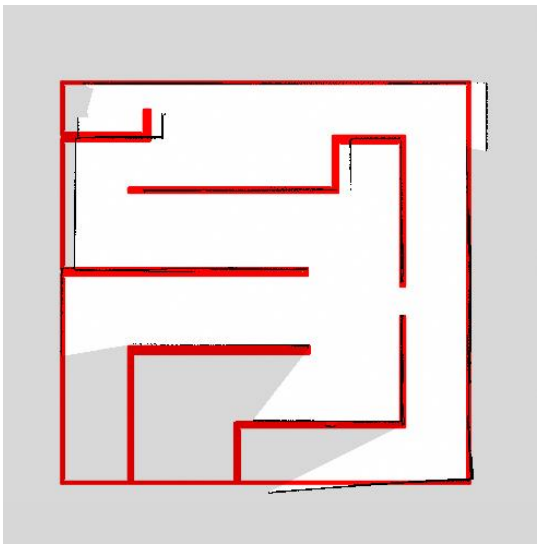
Office – Three Robots – 0.3 m/s



Office – Three Robots – 0.5 m/s

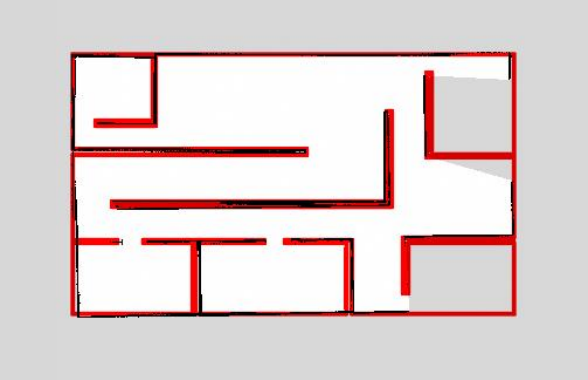


Office – Three Robots – 1.0 m/s

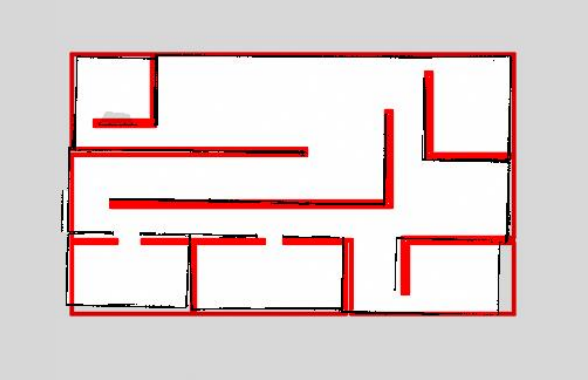


Map 2

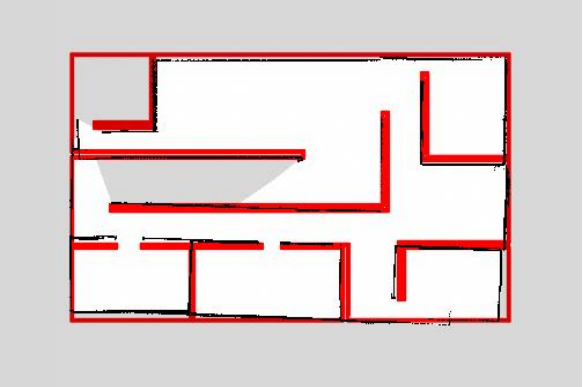
Textured – One Robot – 0.1 m/s



Textured – One Robot – 0.3 m/s



Textured – One Robot – 0.5 m/s



Textured – One Robot – 1.0 m/s

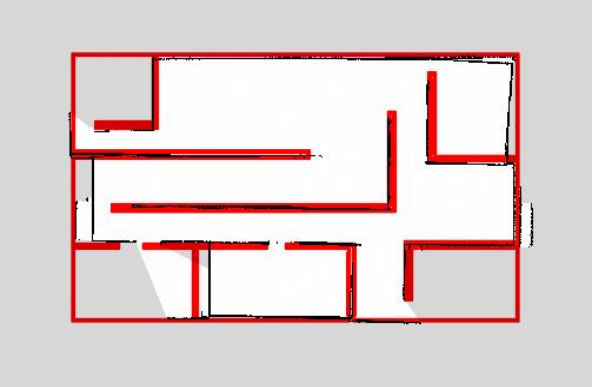
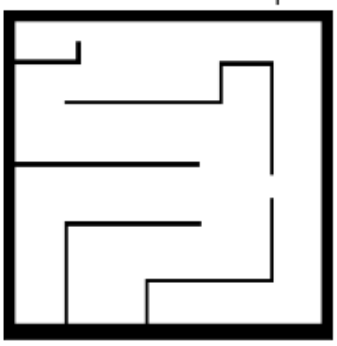
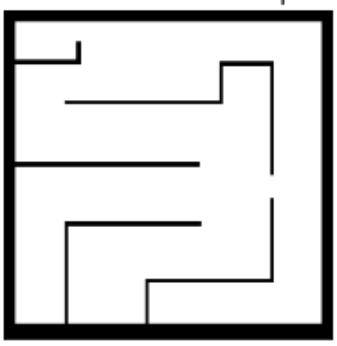
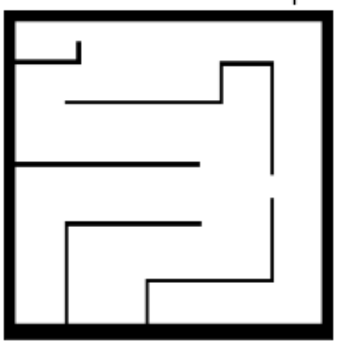
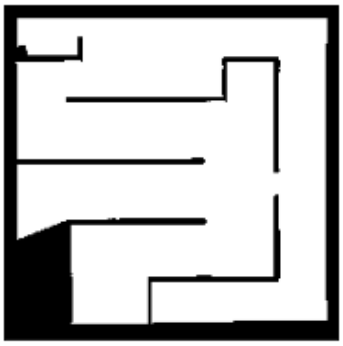
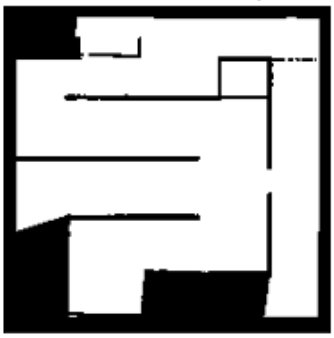
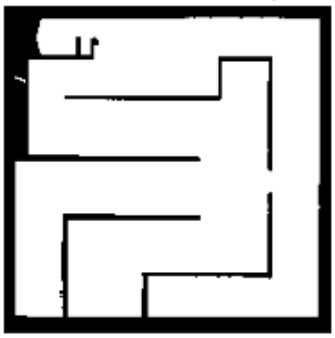
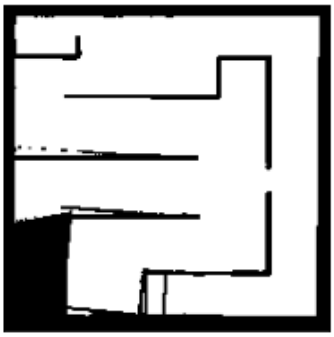
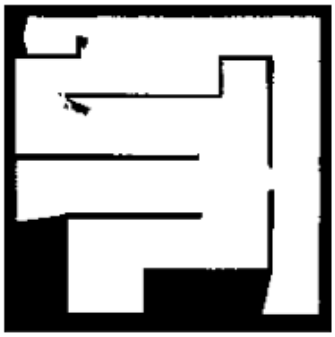
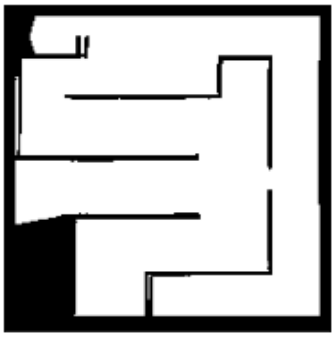
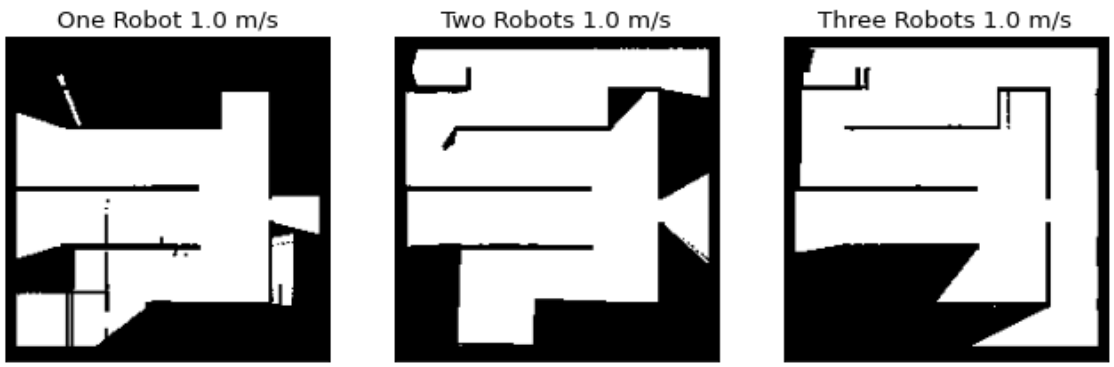


Image Processing Results

Image Processing Output Plots from Velocity Experiments		
One Robot Real Map	Two Robots Real Map	Three Robots Real Map
		
One Robot 0.1 m/s	Two Robots 0.1 m/s	Three Robots 0.1 m/s
		
One Robot 0.3 m/s	Two Robots 0.3 m/s	Three Robots 0.3 m/s
		
One Robot 0.5 m/s	Two Robots 0.5 m/s	Three Robots 0.5 m/s
		



Data Output from Map Accuracy Comparison Program

```
one robot img1% complete: 67.75367907938174
one robot img2% complete: 83.39246598586986
one robot img3% complete: 90.74136920371456
one robot img4% complete: 56.252121062536645
two robot img1% complete: 80.23015456761175
two robot img2% complete: 79.24598155061241
two robot img3% complete: 85.42251565729799
two robot img4% complete: 74.34979792058742
three robot img1% complete: 91.2103168481782
three robot img2% complete: 95.53882701385247
three robot img3% complete: 91.123931755777
three robot img4% complete: 77.49668342948817
```

Appendix H: **INDIVIDUAL ENGINEER PROJECT CONTRIBUTION SUMMARY**

Engineer	Mateo Alvarez

Summary of MEng ACES Project Activities

- Gained further insight on ROS through tutorials and investigated ultrasonic sensors simulation and functionality within *Gazebo*.
- Conducted research on different areas including 'Task allocation' and 'Path planning', 'SWARM navigation' and 'collaborative mapping' among others.
- Benefited the team with previous knowledge involving robot design, steer configuration, hardware, and kinematics.
- Learned to set up and use GitHub for data sharing within the team.
- Worked as a sub-team on the implementation of the navigation and exploration system on ACES robots using Turtlebot3 as a reference system.
- Was responsible for the investigation and implementation of the map converter node.
- Strong contribution to designing and conducting experimentation procedures.
- Ensured full communication of progress to every member of the team for successful cooperation.
- Attended team meetings with a proactive and dynamic role trying to involve all members of the team.
- Contributed to the development, formatting, and proofreading of the team presentation and thesis.

Engineer	Luke Byrne

Summary of MEng ACES Project Activities

- Responsible for general project management and development of project timelines and Gantt chart.
- Set up team GitHub page to aid in project development.
- Set up team Discord server to aid in team communication and organisation.
- Brought in previous knowledge and research on SLAM.
- Gained understanding of *Gazebo* simulator and Blender by implementing a QR code localization system using the ACES robot.
- Conducted initial search for peer reviewed material on arXiv. Retrieved 165 papers and sorted them for review by team.
- Presented team development methodology during kick-off presentation.
- Responsible for implementing collaborative SLAM system on ACES robots.
- Implemented and configured OctoMap server to produce 2D occupancy grids from 3D point clouds.
- Performed verification testing on ORBSLAM3 RGB-D, and ORBSLAM2 RGB-D and monocular SLAM systems.
- Produced high-level block diagrams of ACES 2D and 3D systems.
- Developed methods for automating ACES testing via bash scripts and launch files.
- Worked with Kamil to produce final ACES launch files to allow system to function properly as a whole.
- Parameterised launch files to allow for easy adjustment of speed and number of robots in simulations.
- Developed python script to process final results using OpenCV.
- Used Photoshop to produce final visualisation of results.
- Wrote the bulk of the documentation available on the team GitHub.
- Wrote installation script to simplify and automate installation of the ACES system and simulations, available on GitHub.

Engineer	Jamie Hardie

Summary of MEng ACES Project Activities

- Gained an understanding of ROS by exploring the test robot that was made available to us by DR. Mata.
- Completed online ROS and *Gazebo* tutorials to further enhance my understanding of the subject.
- Used the nodes within the ROS package such as the camera/image_raw to gain outputs to improve my understanding.
- Conducted research for the literature review by reading peer reviewed papers on topics such as “Multi Robot exploration”, “Multi Robot communications”, “Collaborative SLAM using Swarm systems”.
- Used GitHub to access the team page and followed a tutorial on how to utilize its functionality.
- Attended team meetings and contributed towards presenting the kick-off presentation, as well as writing the interim report.
- Researched 2D and 3D mapping systems that were suitable for ACES.
- Implemented the 3D map merging package and tested it.
- Helped conduct research into suitable testing methodology for the design of the experiments.
- Help present the group presentation for the ACES project.
- Kept an open dialog about matters regarding the project through discord and attending regularly scheduled meetings with the team and Dr. Mata.
- Helped write, edit, and proofread the final report for the ACES project

Engineer	Kamil Shabkhez

Summary of MEng ACES Project Activities

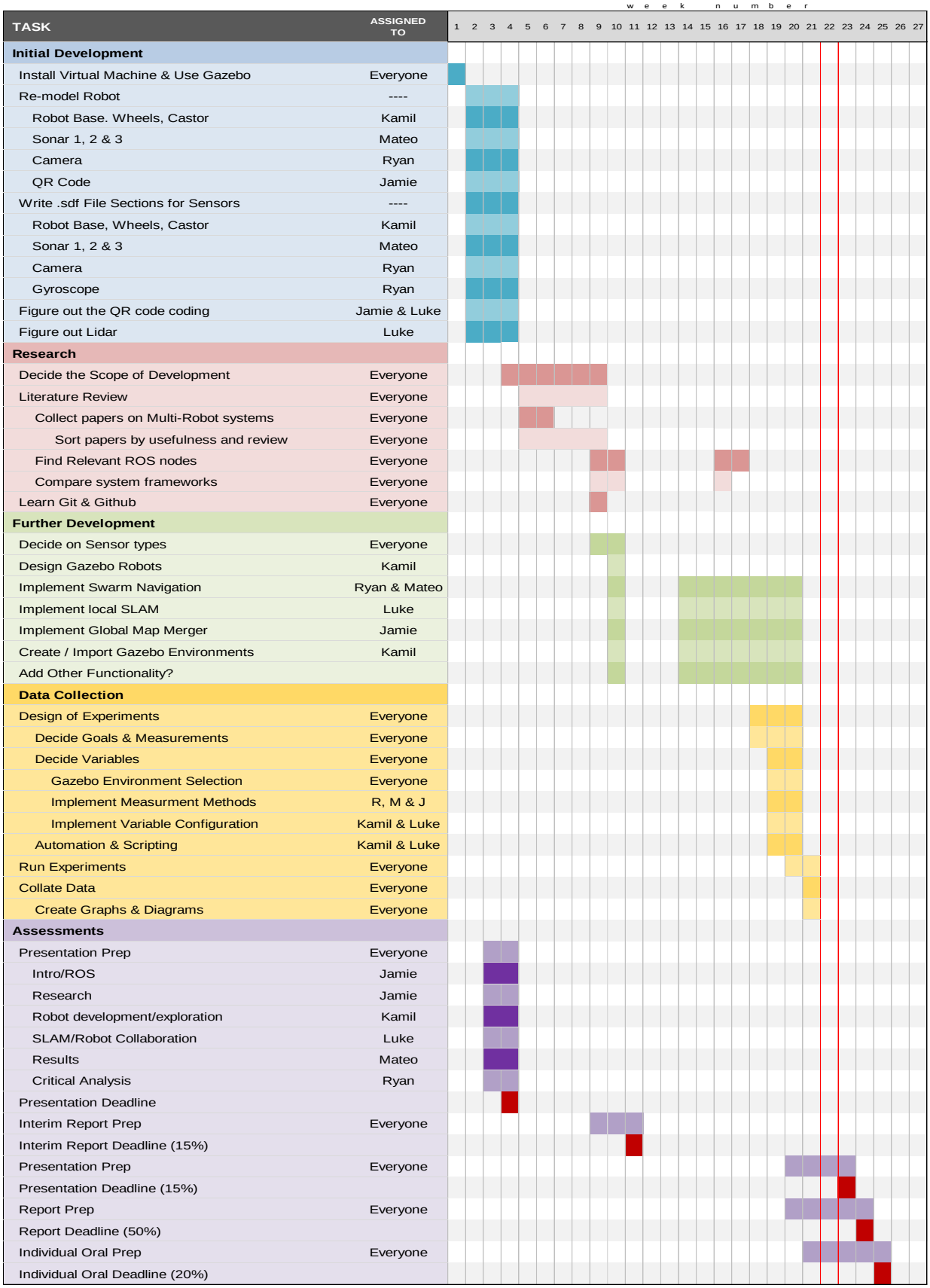
- Developed a better understanding of ROS by working on developing the ACES robot in *Gazebo*.
- Designed the ACES robot by making CAD designs using *Creo*, *Fusion 360* and *Blender*, and assembling them as URDF files.
- Added sensors and their appropriate plugins to add functionality and control the ACES robot.
- Conducted experiments to improve the control of the ACES robot.
- Integrated the **gmapping** package to enable 2D SLAM functionality.
- Integrated the **navigation stack** to be used for the ACES robot.
- Integrated the **explore-lite** package to be used with the ACES robot.
- Conducted some research of peer-reviewed papers focusing on Multi-Robot SLAM.
- Worked with Luke to facilitate multi-robot configuration for ACES system, enabling SWARM capability.
- Configured **gmapping**, **navigation**, and **explore-lite** packages to be used with multi-robot configuration.
- Integrated the 2D **multi-robot map merge** node to be used with multi-robot configuration.
- Created different *Gazebo* environments for testing, and applied textures to aid the visual SLAM systems.
- Learned to use Git and GitHub for effective collaboration between team members.
- Made a significant contribution to the project's GitHub repository, with regular updates assisting the overall project's development.
- Partook in team meetings, writing the thesis, presenting the final presentation, writing the interim report, and presenting the initial kick-off presentation.

Engineer	Ryan Walker

Summary of MEng ACES Project Activities

- Contributed to project meetings, coordinating formal online group discussions, frequently sharing ideas and resources.
- Shared organisational responsibilities, recording formal meeting minutes, and acting as point of contact with project supervisor.
- Developed understanding of ROS through experimentation with existing robot models and *Gazebo* simulation tutorials. Focused initial simulation experimentation on obtaining simulated camera data from a simulated *Gazebo* environment. Gained familiarity with Blender and OnShape CAD packages, as well as developing URDF files to import physical features into simulations.
- Research activities focused on literature relating to hardware configuration, swarm navigation, exploration and developing multi robot systems with ROS and SLAM. Supplementary research materials were also consulted, including web resources and ROS and *Gazebo* documentation.
- Focussed ROS development work on development of exploration and navigation components of the ACES system. This has involved working closely with a sub-team and effectively communicating findings and development with the wider project team to inform simultaneous development.
- Provided conceptualisation and prototyping of OpenCV based accuracy measurement application, as well as contributing to design of verification tests and experimentation.
- Contributed to GitHub readme files for final deliverable, providing description of program functionality.
- Strong written contribution to thesis and presentation, as well as layout, diagrams, and proof reading.

Appendix I: GANTT CHART



TASK	ASSIGNED TO
Initial Development	
Install Virtual Machine & Use Gazebo	Everyone
Re-model Robot	----
Robot Base. Wheels, Castor	Kamil
Sonar 1, 2 & 3	Mateo
Camera	Ryan
QR Code	Jamie
Write .sdf File Sections for Sensors	----
Robot Base, Wheels, Castor	Kamil
Sonar 1, 2 & 3	Mateo
Camera	Ryan
Gyroscope	Ryan
Figure out the QR code coding	Jamie & Luke
Figure out Lidar	Luke
Research	
Decide the Scope of Development	Everyone
Literature Review	Everyone
Collect papers on Multi-Robot systems	Everyone
Sort papers by usefulness and review	Everyone
Find Relevant ROS nodes	Everyone
Compare system frameworks	Everyone
Learn Git & Github	Everyone
Further Development	
Decide on Sensor types	Everyone
Design Gazebo Robots	Kamil
Implement Swarm Navigation	Ryan & Mateo
Implement local SLAM	Luke
Implement Global Map Merger	Jamie
Create / Import Gazebo Environments	Kamil
Add Other Functionality?	
Data Collection	
Design of Experiments	Everyone
Decide Goals & Measurements	Everyone
Decide Variables	Everyone
Gazebo Environment Selection	Everyone
Implement Measurment Methods	R, M & J
Implement Variable Configuration	Kamil & Luke
Automation & Scripting	Kamil & Luke
Run Experiments	Everyone
Collate Data	Everyone
Create Graphs & Diagrams	Everyone
Assessments	
Presentation Prep	Everyone
Intro/ROS	Jamie
Research	Jamie
Robot development/exploration	Kamil
SLAM/Robot Collaboration	Luke
Results	Mateo
Critical Analysis	Ryan
Presentation Deadline	
Interim Report Prep	Everyone
Interim Report Deadline (15%)	
Presentation Prep	Everyone
Presentation Deadline (15%)	
Report Prep	Everyone
Report Deadline (50%)	
Individual Oral Prep	Everyone
Individual Oral Deadline (20%)	

Appendix Figure I-1 – Project Gantt Chart