Thesis Final Report

Supervisor: Prof. Goldie Nejat

Student Name: Ge(Aron) Lin

Student Number: 1002342565

Abstract

The robotics applications in Urban Search and Rescue (USAR) in a disaster context raised great attention in recent years. It eliminates the human need to search the victims, which greatly reduces the potential human and property loss. However, an intelligent robot needs to be trained and upgraded from experience to execute the task safely, accurately and efficiently. While robot training in the real world is expensive and time-consuming, a 3D simulator was introduced to solve this problem. The simulation makes the robot training fast and efficient. However, there is not 3D simulator explicitly designed for USAR. Therefore, in this project, a 3D-realistic-simulator simulating multi-robots executing tasks, such as frontier exploration, is designed for future researchers to test their coordination algorithm. The 3D simulator was tested with a greedy frontier exploration algorithm. The results show that the robot fleet will explore the environment more efficiently with an increasing number of the robot, and the results met the expectation. The experiments proved that the 3D simulator design is able to mimic real-world robot behaviour accurately and has a great potential for future improvement.

Table of Contents

List of Tables

List of Figures

1. Introduction

Urban Search and Rescue (USAR) in a disaster context can be complicated and slow by humans due to complex weather and the environment [85 1]. Therefore, robotic applications raised great attention recently [1]. The main advantages of a rescue robot are the following. First, the robot will not be affected by emotion and be object-oriented. Second, robots can be easily repaired or replaced [2]. Third, robots can learn and upgrade from past experience, for example, reducing the time cost and increasing the accuracy of environment mapping. Finally, robots can be deployed in large quantities and move freely in complex terrains, such as a highly cluttered environment [1][2]. Nowadays, a fleet of autonomous robots is commonly used to run USAR because of increased efficiency [1]. The robots need to be tested and improved repeatedly before coming into service [3]. However, it is impossible to test the robot in every disaster scene. Building a simulated disaster context for testing is an alternative, but it is expensive and time-consuming. A 3D simulator was introduced to solve the problem. A 3D simulator is software simulating a realistic environment and three-dimensional models (e.g., humans and robots). A well-designed 3D simulator should follow the scientific laws close to reality so that the robot can be designed and trained accurately. [4][5][6][7][8] have achieved simulations of multi-robot for different applications, including terrestrial, aerial, and underwater robots. However, there is no simulator designed explicitly for USAR. Therefore, in this thesis project, a multi-robot 3D-realistic-simulator will be designed for future researcher to test their coordination algorithm.

Robot Operating System (ROS) is chosen as the middleware for this project to develop the robot control system because it is typically designed for robotics. It has a vibrant community with numerous support [9]. The researcher can easily share their work and implement other state-of-art in the community. In ROS, projects are built in the form of packages. ROS provided a publisher/subscriber communication system. The package can publish its result through a node and subscribe from other packages. It is easy to develop a new project because it can simply subscribe to existing packages to obtain desired data without integration [9].

2. Project Requirement

This section will describe the project requirement in detail. This project aims to build a 3D-realistic simulator to simulate multi-robot executing USAR tasks cooperatively in a 3D simulated environment. Therefore, the project can be divided into two main parts: 3D Environment Simulation and Robot Simulation.

2.1 3D Environment Simulation

For environment simulation, it is required to model all possible components in the disaster scene. However, to save the simulation cost, the simulator will focus on modelling the terrain, unstructured obstacles, natural environment, robots, and victims. The simulator should also mimic the physical properties in real-life of all the elements in the disaster, including size, shape, texture, mass, and collision.

2.2 Robot Simulation

Two main tasks of the rescue robot in USAR are unknown environment exploration and search of the victims [1]. Therefore, the robot requirement can be further divided into Unknown Environment Exploration and Victim Detection.

2.2.1 Unknown Environment Exploration

The simulated robots should be able to run frontier exploration, which is the most common exploration technique. The frontiers are defined as boundaries between unknown and explored areas [10]. The core of frontier exploration is autonomous navigation, which requires a navigation stack set up in the robot [1].

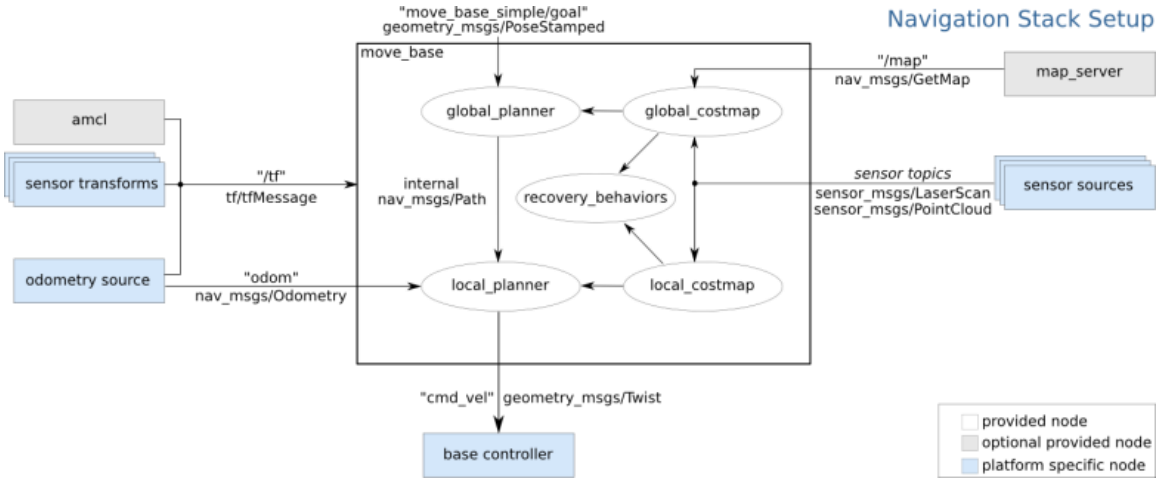


Figure 1. Navigation stack

Figure 1 shows the recommended navigation stack from ROS [11]. In general, the robot should have 1) simultaneous localization and mapping (SLAM) and then 2) path planning for autonomous navigation.

2.2.1.1 SLAM

simultaneous localization and mapping (SLAM) based algorithm is commonly used to map the unknown environment [1]. In SLAM algorithms, a map containing information about obstacles, free space, and unknown space is often generated based on the robot's sensor input [1][12][13]. By comparing the features from the sensor input in different time frames, the algorithm is then able to localize the robot in the map [1] [12] [13].

2.2.1.2 Path Planning

Path planning is defined as an agent moving from the starting point to a designated point by avoiding obstacles at minimum cost [14] [15]. The cost involves the time and energy that the robot uses and the distance it travels [14] [15]. Path planning is divided into global path planning and local path planning [14]. In global path planning, the global path planner generates a trajectory from the current position to the destination based on the map. Sometimes, there are moving obstacles across the trajectory, and the map does not update the obstacles. Therefore, the local path planner is required to adjust the trajectory to avoid obstacles based on the local sensor input.

2.2.2 Victim Identification

In order to rescue a victim, the robot should first gather information about the victim, including current condition, location, and orientation. These require the simulated robots to be capable of generating high-level semantic observations (e.g., victim detected and victim health status) based on sensor input.

3. Project Objectives

This section describes the objectives of 3D simulator design.

- Integrated with ROS
    - The proposed software or algorithms should be integrated with ROS to reduce the development cost.
- Compatible with an arbitrary model of the robot

    - The design should support any robot models so that future researchers can test their algorithms with minimum integration.
- Capable of adding an arbitrary number of robot
    - The design should be able to simulate a random number of the robot at the same time.
- Should share data with others
    - Robots are often required to communicate locally when global communication is not available due to low signal. Therefore, the design should simulate robots communicating with each other.

4. Literature Review

In this section, the state-of-art of two requirements will be reviewed and discussed in detail. A proposed software or algorithm will be selected based on the discussion for future implementation.

4.1 3D Environment Simulation

To simulate a 3D environment, the first step is to choose the 3D simulator platform to work on. This section will investigate various 3D simulators available on the internet. [16] concluded a list of the most promising 3D simulators. The paper ranked the simulators based on the number of scientific research from "Google Scholar". Table 1 shows the strings of the paper used in "Google scholar" and the rank. Santos et al. [16] analyzed the top three most promising simulators: Gazebo, Unity and V-Rep. However, Unity is a cross-platform game engine developed by Unity Technologies [17]. This paper will analyze Gazebo, V-Rep, and Webots since Webots is the next most promising simulator, and all three simulators were mainly designed for robotics applications.

Table 1. 3D simulators ranking based on number of research [16]

String	Results (n)	String	Results (n)
"Unity3D" OR "Unity 3D" OR "Unity simulator"	25.800	UsarSim	1.650
Gazebo	30.900	OpenRAVE	1.210
V-Rep	10.400	OpenHRP	877
Webots	4.730	SimSpark	425
ARgOS Simulator	2100	RoboDK	103

The literature review will be divided into six sub-categories summarized from the objectives and are essential characteristics of the simulators [5][16]. The simulators will be analyzed individually about all five sub-categories: 1) Library of Models, 2) Ability to Customize the Simulated Scene, 3) Ability to Customize the Robots, 4) Integration with ROS, 5) Realistic Simulation, and 6) Multi-robot Application.

4.1.1 Library of Models

An extensive repository of 3D models is significant to build a simulated world. With an extensive library, less effort will be needed to model the environment. The 3D models include the terrain, unstructured obstacles, natural environment, robots, and victims.

A. Gazebo

Gazebo is an open-source 3D-robotic simulator. Thanks to the broad base of contributors, Gazebo provides the user with an extensive library of 3D models, including plants, vehicles, furniture, buildings, and even collapsed buildings. Various popular robots are also equipped, including PR2, Pioneer2 DX, iRobot Create, and Turtlebot [18] [19].

B. V-Rep

V-Rep is a versatile and scalable simulation framework. It offers a large repository of models from infrastructures like walls to furniture. Many popular robotic arms, mobile robots, and even humanoid robots have already been included in the software [5].

C. Webots

Webots is a mobile robotic simulation software developed by Cyberbotics Ltd. in Swiss. It is capable of modelling and simulating any wheeled, legged, or aerial robots. Also, a complete library of sensors and actuators is provided [20].

4.1.2 Ability to Customize the Simulated Scene

The capability to customize the environment where the robots move in the simulator is essential since no disaster scene is the same. There will not always exist a matching model in the library. The ability to customize makes it possible to reproduce a specific disaster scene.

A. Gazebo

Not many modelling features are provided in Gazebo. The 3D models can be directly inserted into the scene from the local and online library. But it is not possible to edit the model inside the Gazebo. The object must be modelled through an external software, for example, Blender, and imported to Gazebo in URDF (Universal Robot Description Format) file. However, Gazebo does provide a building editor in which basic infrastructure can be built using walls and floors. The scene can also be edited through a .world file in SDF (Spatial Data File) format. The location and orientation of the models are set manually in the .world file. Other physical properties like size, inertial, mass and collision can be adjusted as well [18] [19].

B. V-REP

V-REP provides user-friendly modelling features where users can directly drag and drop the model into the V-REP scene. All the models inside the scene are available to be accessed, and their physical properties can be modified through the user interface (Figure 2). V-REP includes a URDF import tool as well [19].

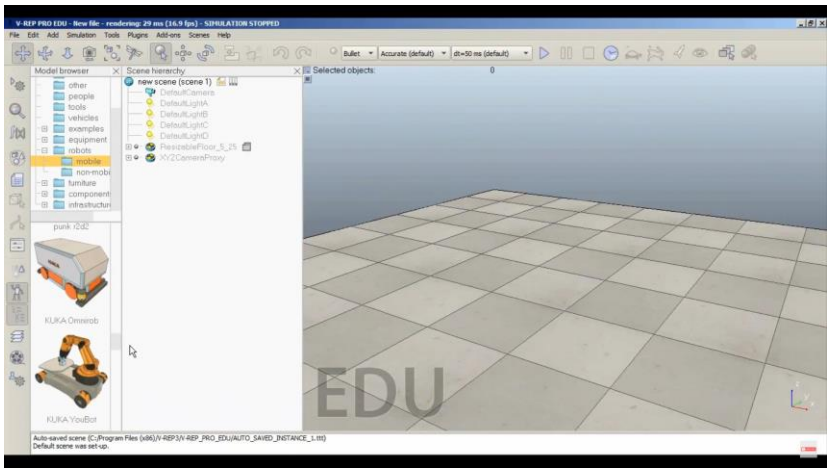


Figure 2. V-REP user interface [19]

C. Webots

Webots uses advanced hardware-accelerated OpenGL technologies to create a simulated environment. Similar to Gazebo and V-REP, the user can drag and place the 3D model in Webots. The models' physical properties are defined as nodes including geometry, physics, and textures and are ready for modification in the Webots: Scene Tree (Figure 3). 3D models can be imported through a VRML97 standard file from external 3D modelling software [20].

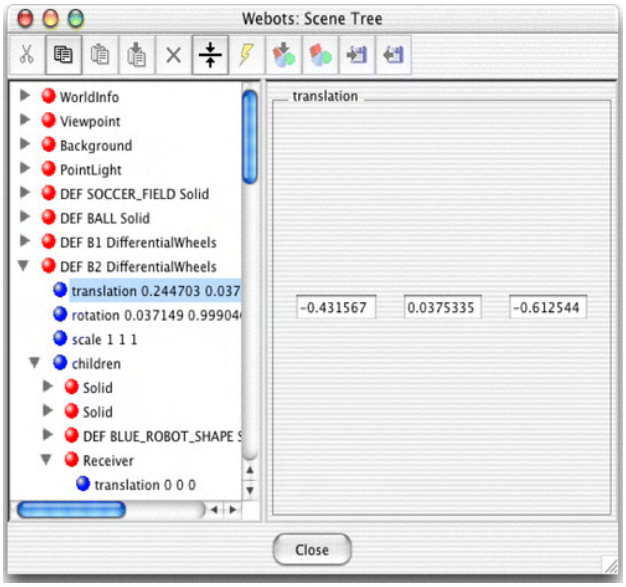


Figure 3. Webots scene tree [20]

4.1.3 Ability to Customize the Robots

The robot configuration varies in different applications. The capability of modifying the robot, for example: replacing a laser scanner with a stereo-camera, makes it possible to test different robot configurations quickly.

A. Gazebo

As discussed in the last session, a robot's geometry configuration can be modified through the URDF file outside Gazebo. The sensors and actuators are prepared as Gazebo-ROS plugins in the Gazebo database and can be added to the Robot's URDF file by following the tutorial [21]. The parameters of them can be adjusted inside the URDF file.

B. V-REP

Customizing the robots in V-REP can be done through visual editing. The sensors can be installed by simply dragging and placing the sensor on top of the Robot in V-REP. The actuators are generated as the ROS plugins through a vrep_ros_bridge package and installed in the same way Gazebo does [19].

C. Webots

Webots prepared the sensors and actuators as nodes to be readily inserted into any robots. They can be mounted to any frame by merely adding a node to the structure and selecting the desired sensors/actuators [20].

4.1.4 Integration with ROS

Because ROS is an open-source community, with a high integration with ROS, the simulator can implement many existing robot-control algorithms with minor modification.

A. Gazebo

Gazebo is readily integrated with ROS by a set of Gazebo plugins that support a large repository of robots, sensors, and actuators. Gazebo and ROS share the same communication structures, making it easy to manage and process the data from the

sensors and robots through ROS nodes. The data-communication method is well-documented on the official ROS website [22]. Besides, many existing packages can be implemented in the simulation with few modifications thanks to the ROS community.

B. V-REP

There is no native ROS node for V-REP; however, V-REP provides a ROS-Interface as part of the V-REP API framework. It duplicates the ROS API and turns V-REP into a ROS node to communicate with other nodes as Gazebo does [19].

C. Webots

Webots uses the standard ROS controller, and it generates ROS topics for each element (sensors, actuators and robots) in the scene. In this way, Webots acts as services and topics and can communicate with other ROS nodes in the same way Gazebo does [20].

4.1.5 Realistic Simulation

The closer the simulation is to reality, the more accurate the algorithm's test is. The time cost will also be reduced.

Table 2. The physics engine of V-REP, Gazebo and Webots [16]

	Physics Engine
V-REP	Bullet, ODE, Vortex, and Newton
Gazebo	Bullet, ODE, Sim-body, and DART
Webots	ODE

The table shows the native physics engines of all three robotic simulators. Both V-REP and Gazebo support four different physics engines, while Webots only has one ODE engine [16].

4.1.6 Multi-robot Application

The simulator must support the multi-agent applicant.

Table 3. Multi-robot application in V-REP, Gazebo and Webots [16]

	Multi-robot Application
V-REP	Yes
Gazebo	Yes
Webots	Yes

As can be seen from the table, all three simulators support the multi-robot application.

4.1.7 Discussion and Selection

Regarding the 3D model library, all three simulator includes a vast local library of 3D models ranging from small items like furniture to infrastructure. Besides the local library, thanks to the open-source, Gazebo provides an additional online library of 3D models supported by numerous developers [18] [19].

In terms of scene modelling and editing, Gazebo requires an in-depth knowledge of SDF and URDF file because it cannot edit the model inside Gazebo [18] [19]. For instance, to replace the laser scanner with a stereo-camera in a robot, the corresponding URDF file should be modified, and the simulation must be restarted to enable the new sensor. At the same time, it is much more convenient to work with V-REP and Webots. The scene elements can be edited directly inside V-REP and Webots [19] [20].

Besides, all three simulators are all integrated with ROS to some extent. Gazebo is currently supported by ROS and has already been integrated [22]; therefore, it is the most straightforward one to work with Gazebo. Regarding V-REP and Webots, they are integrated through an external framework and packages [19] [20]. In the meantime, all three software can simulate multi-robot simultaneously. So far, it is invidious to select; however, the Webots's poorest simulation and graphic quality make it the first to be eliminated from the list. Webots only support the ODE physics engine, while Gazebo and V-REP are equipped with various physics engines.

Overall, Gazebo will be selected as the simulation platform for future development, even though it is much easier to simulate in V-REP. The reason behind this is that even if

Gazebo requires extra effort to edit the scene, the editing can still be done in a small amount of time when the user gets familiar with URDF and SDF files. Besides, modelling through the URDF file can create a more complex and more accurate environment compared to V-REP [19]. Whatsmore, the open-source community provides extensive documentation of control algorithms, simulation scenarios, robot configuration, and sensor information, which V-REP lacks. These will help reduce the investment demanded to develop the future project.

4.2 Robot Simulation

In this section, the literature of the two main components (SLAM and path planning) in autonomous navigation will be reviewed in detail.

4.2.1 Robot SLAM Algorithm

Unlike traditional 2D-indoor localization and mapping, USAR robots are required to map and localize in a full-3D highly cluttered environment [1]. Conventional 2D laser SLAM is not applicable in this situation. The two most robust 3D SLAM algorithms were found during the research: 1) visual-based SLAM and 2) lidar-based SLAM [23] [24].

4.2.1.1 Visual-based SLAM

Visual-based SLAM used a set of cameras or stereo-camera to obtain images of the environment. It estimates the robot's motion and position by extracting numbers of features (mostly point clouds) from the photos while simultaneously using this data to generate a 2D or 3D map [23]. It stores the images and corrects the map when the robot revisits a past location. The review is limited to the methods that can estimate a real-scale environment [13]. In this section, papers pertaining to visual-based SLAM will be discussed in detail. These methods are: 1) RTAB-Map [13], 2) ORB-SLAM2 [25], 3) S-PTAM [26], 4) DVO-SLAM [27], and 5) RGBiD-SLAM [28].

In [13], a Real-Time Appearance-Based Mapping (RTAB-Map) was proposed for long-term and large-scale environment mapping. It is open-source and has been integrated into ROS as a rtab-map-ros package. When the odometry is no available on the Robot, RTAB-Map provides visual odometry using two common odometry approaches: Frame-To-map (F2M) and Frame-To-Frame (F2F) [29]. F2M registers the new frame against a

feature map created from the last keyframe, while F2F registers against the last keyframe [29]. It can use either an RGB-D or stereo camera as the input. RTAB-Map detects features called GoodFeaturesToTrack (GFTT) [30] when receiving the image of the environment. The iterative Lucas-Kanade [31] is used to derive disparity between the left and right images for stereo-image. In comparison, the RGB-D frame's depth image acts as a mask for GFTT to avoid invalid features. The detected features are then matched either by nearest neighbour search [32] with nearest neighbour distance ratio test [33], which registers BRIEF descriptors [34] against the extracted features in the feature map or directly against the last keyframe. RTAB-Map can then estimate the robot's position by computing the transformation of the current frame to the previous keyframe (F2F) or the feature map (F2M) using the Perspective-n-Point RANSAC implementation [13] [35]. A block matching algorithm was [36] then introduced to convert the stereo image into a point cloud, in which an occupancy grid is generated. Also, a loop closure detection described in [36] is used to optimize the localization and mapping accuracy. The performance was evaluated on four different datasets: KITTI [37], TUM RGB-D [38], EuRoC [39] and PR2 MIT Stata Center [40]. The matrix used to evaluate the performance is the absolute trajectory root-mean-square error (ATE) from [38].

ORB-SLAM2 [25] and S-PTAM [26] are both real-time keyframe-based (F2F) SLAM method. ORB-SLAM2 works with monocular, stereo and RGB-D cameras, while S-PTAM only uses the stereo camera. They include map reuse, loop closing and relocalization. ORB-SLAM2 prep-processes the image to only extract ORB features [41] at salient key points, and S-PTAM chose to extract GFTT features with BRISK extractor. The camera' pose is then initialized and estimated by matching the extracted features with the last keyframe [25] [26]. For ORB-SLAM2, A locally visible map is built with the covisibility graph of extracted keyframes once the estimation and feature matching is done. For S-PTAM, it generates a local map based on the method presented in [42] called Parallel Tracking and Mapping. Both ways can detect the loop closure using a bag of words place recognition module described on DBoW2 [43]. The pose and map are refined by motion-only BA(bundle adjustment) when a loop closure is detected. ORB-SLAM2 was tested on KITTI [37], EuRoC [39], and TUM RGB-D [38] dataset. The evaluation matrix is the ATE from [38] as well. At the same time, S-PTAM was only

tested on the KITTI database [37] [26]. It calculates the root-mean-square-error between the output frame and the ground truth.

Similarly, DVO-SLAM [27] and RGBiD-SLAM [28] are also keyframe-based (F2F) visual SLAM. However, they proposed a different odometry algorithm called dense visual odometry [27] [28] [44]. Instead of extracting features from the camera frame to estimate camera motion, they used joint photometric and geometric errors in RGB-D images. A single RGBD camera can achieve this. They can construct a 3D map by concatenating each keyframe's point clouds to the map point cloud [28]. It should be pointed out that DVO-SLAM requires an external loop closure detection approach to refine the pose estimation and map [27]. DVO-SLAM evaluate the ATE based on the TUM RGB-D dataset [27] [38], and RGBiD-SLAM compared their results with the state-of-the-art and their database in terms of MSR of trajectory estimation.

4.2.1.2 Lidar-based SLAM

Laser-based SLAM used a laser scanner to obtain point-cloud data of the environment. It estimates the robot's motion and position by comparing multiple laser-scan measurements and builds a map using the received data [24]. In this section, papers pertaining to lidar-based SLAM will be discussed in detail. These methods are: 1) RTAB-Map [13], 2) Google Cartographer [45], and 3) SegMatch [46].

RTAB-Map also introduced the lidar-based SLAM [13]. It collected point cloud data using a 3D-lidar. There are also two standard methods to generate odometry: Scan-To-Scan (S2S) and Scan-To-Map (S2M). They are similar to visual odometry but change the frame to point clouds and substitute the local map with a point clouds map. The point cloud map is generated from all the past keyframes using either Point-to-Point (P2P) or Point-to-Plane (P2N) method [13]. When a new point cloud is received, it will be added to either the point cloud map (S2M) or the last keyframe (S2S) after iterative-closest-point (ICP) [28] is done using libpoint-matcher [44]. Unlike visual SLAM in RTAB-Map, the pose estimation has to be first initialized from the last registration or external odometry. The pose is then estimated by matching the current point cloud data with the keyframe or the point cloud map. The pose and map optimization are completed in the same way as the visual-based SLAM. The 2D occupancy grid can also be generated by

filtering and ground segmentation described in [13]. The evaluation criteria are the same as the one mentioned in the last section.

Google proposed a lidar graph-based SLAM approach called Google Cartographer in 2016 [45]. The SLAM process is divided into two parts: local SLAM and global SLAM. The local SLAM estimates the pose and orientation based on a small and low-resolution map (submap) using a Ceres scan matcher. The submap is built in a probability grid form. Every hit and reflection on the LiDAR scan are identified as an occupied pixel and is added to the grid. The area between the obstacle and the robot and the area where the laser is not reflected is identified as free space in the grid map. The pose is estimated by comparing the current position with the submap. The global SLAM uses the sparse pose adjustment method [47] to minimize the shifting error accumulated in local SLAM. The scan matching in global SLAM is similar to the one in local SLAM but with a broader range to cover the submap. When a loop closure is detected, the submap will be refined and reconstructed based on the scan matching. Algorithms like Karto SLAM [48], Lago SLAM [49], GMapping [50] and Hector SLAM [51] have similar functions, but they only support 2D lidars. The algorithm was evaluated based on the KITTI benchmark [37] in terms of trajectory estimation and was also tested in the real world with various conditions [45].

In [46], a 3D-lidar based SLAM is proposed called SegMatch. It first segments the point clouds collected into different elements, for example, parts of vehicles and buildings, using the "Cluster-All method" from [52]. Features can then be extracted from the segments based on two descriptors: *eigenvalue based* and *ensemble of shape histograms*. A random forest classification algorithm is deployed to classify the features further, and the corresponding features are matched based on the classification. Matched elements are then fed to a random sample consensus (RANSAC) [53] to test the accuracy. It detects a loop-closure when there are features matched successfully. It is noticeable that SegMatch does not provide pose estimation. SegMatch's performance was evaluated by calculating the successful matching rate of the segments generated from the KITTI odometry dataset [37][46].

4.2.1.3 Summary of Limitation

In visual-based SLAM, the localization and mapping are achieved by matching the features extracted from the images with either the keyframe or the local map [25] [41] [26] [27] [28]. However, it is assumed that the camera is not obstructed, and there are always enough features in the image. In real life, the image quality suffers from changes in illumination and visibility [46]. The visual SLAM becomes unreliable when the image quality is low [46]. In contrast, the performance of lidar-based SLAM is not affected by illumination.

In lidar-based SLAM, the localization and mapping are achieved by matching the point cloud data from the laser scanner with either the keyframe or the point cloud map [13] [46]. It only extracts geometry information of the environment. Therefore, there is no semantic information gathered. The odometry may drift a lot when the environment has a regular shape, for example, a corridor. When a robot with a short-range scanner moves along a corridor, the point cloud data received will be two parallel lines. The map will not be correctly updated since it can not detect any change in the keyframes or the point cloud map.

4.2.1.4 Discussion and Selection

Overall, RTAB-Map is selected for future simulation because it is the only algorithm supporting both visual-based and lidar-based SLAM [13]. Both methods in RTAB-Map can generate a 2D occupancy grid required by most of the popular path planner [54]. Besides, it is readily integrated with ROS as *the rtabmap_ros* package.

4.2.2 Robot Path Planning

Once the robot is localized on the map, the next step is to navigate the robot to explore the unknown environment. In this section, papers pertaining to path planning will be discussed in detail. These papers can be further categorized by 1) global path planner and 2) local path planner [14] [15].

4.2.2.1 Global Path Planner

The global path planning algorithm can be further divided into three categories: 1) grid-based algorithms [55] [56], 2) evolutionary algorithms [57] [58] [59] [60], and 3) sampling-based planning (SBP) [54] [61] [62] [63].

4.2.2.1.1 Grid-based Algorithms

In grid-based path planning, a 2D-occupancy grid is assumed to be provided. The optimal path is calculated based on the state of each cell in the grid. In this section, papers pertaining to grid-based algorithms will be discussed in detail. These methods are: 1) Astar (A*) [55] and 2) D* [56].

Peter E Hart [55] proposed a grid-based pathfinding algorithm called Astar (A*), which is one of the most widely used algorithms. It generates the optimal path by minimizing an evaluation function $f(n) = g(n) + h(n)$. $g(n)$ is the actual cost of moving from starting cell to current cell n. $h(n)$ is the cost of the estimated path from the current location n to the destination. The cost of moving from the current cell to each adjacent cell is computed, and the path with the minimum cost is chosen. The cost is evaluated by either the Euclidean distance formula or the Manhattan distance formula [55]. The experiment results showed that A* can always find the optimal path with the least cost in a 2D environment.

Similarly, D* is a modified version of A* algorithm [56]. It does not require a completed map, and every cell in the grid is assumed to be accessible initially. When a new obstacle is detected, D* will re-compute the cost function and update the grid. Instead of calculating $g(n)$ from the current position toward the goal, it computes the cost backwards from the destination cell to each of the adjacent cells.

4.2.2.1.2 Evolutionary Algorithms

The evolutionary algorithms optimize the path planning by mimicking the biological behaviour, for example, evolution, mutation, recombination, and natural selection [91]. In this section, papers pertaining to evolutionary algorithms will be discussed in detail. These methods are: 1) Particle Swarm Optimization (PSO) [57] [58] and 2) Genetic Algorithm (GA) [59] [60].

In [57] [58], Particle Swarm Optimization (PSO) method was introduced to generate an obstacle-free path for robot navigation. It was inspired by the social foraging behaviour of birds and fishes. The obstacles are defined as a set of particles. Each particle is assigned a coordinate, for example (x_0,y_0) and has a circular region (obstacle zone) where the robot should not enter. PSO then determines a new particle for each particle. The new particle has a minimum distance to the goal and a maximum distance to the original particle. The maximum distance is set manually based on the radius of the circular region. PSO then finds the best-fitted lines among all the new particles from the starting point toward the goals. The path which does not violate the obstacle zone and has the lowest distance is selected. PSO re-computes the paths when a new obstacle is detected or when the existing obstacles move. The experiment was conducted in a simulated environment with static and moving obstacles. Results showed that PSO could generate a smooth and obstacle-free path in an environment with stationary and moving obstacles [57] [58].

A heuristic approach called Genetic Algorithm (GA) was proposed in [59] [60]. It is inspired by the evolution process. An occupancy grid of the environment is required. The robot is assumed to move only in four directions (up, down, left, and right) in the grid. GA generates "populations" of the possible paths ("individual") iteratively. From each "population", the best path determined by a user-defined fitness function are collected to form the next "population". The fitness function may include total path length, number of turns, and number of collisions. The solution is prevented from converging to the local minima using the crossover strategy [59] [60]. GA was tested to find the optimal path in a 10x10 and 100x100 grid world [59]. Results illustrated that GA could effectively handle path planning in an environment with static obstacles.

4.2.2.1.3 Sampling-based Algorithms

In sampling-based algorithms, sample points are generated in the map, and they are intended to be connected to create the path from starting point to the goal. In this section, papers pertaining to sample-based algorithms will be discussed in detail. These methods are: 1) Rapidly-Exploring Random Trees (RRT) [61] [63] and 2) Probabilistic Road Map (PRM) [62].

In [61] [63], Rapidly-Exploring Random Trees (RRT) is implemented to find the optimal path. It was designed to explore a high-dimensional space randomly by generating a space-filled tree [59]. The search space is randomly sampled into points q, which does not collide with the obstacles. The tree grows from the starting point $q_{initial}$ which is connected with a random number of points q_{new}. Each q_{new} continues to connect with adjacent points following the same pattern until q_{new} hits the goal q_{goal}. The algorithm is inherently biased to explore toward the goal and the largest unexplored area to accelerate the exploration. The shortest connected branch is then selected as the optimal path. First, it was simulated to find the optimal path in a 2D maze, and the experiment was extended to compute the path for a 3D piano moving from one room to another. Results showed that RRT performed well over a wide range of applications, including 2D and 3D environments [61][63], and it does not need any parameter tuning or pre-processing.

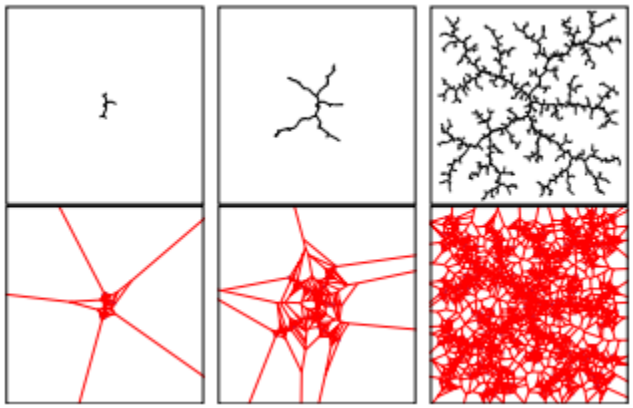


Figure 4. RRT growing process [61]

[62] used the Probabilistic Road Map (PRM) to find the path. Similar to RRT, it also extracts random sample q which is collision-free from the search space. While each q is set to connect with k nearest neighbours using a local planner. If the line between two

points is collision-free, an edge will be added between this two-point. A roadmap is then constructed (Figure 5). By adding the starting point and the goal to the map, one or more paths can be obtained from the roadmap. However, these paths might not be connected, or the path might not be optimal when the sample is not enough. The evaluation criteria for PRM are the cost of finding the best path in a 2D configuration-space. Results showed that PRM could find the path from the starting point to the goal in both 2D and 3D environments [62]. However, PRM requires a pre-processing of the search space to create a roadmap before planning the path.

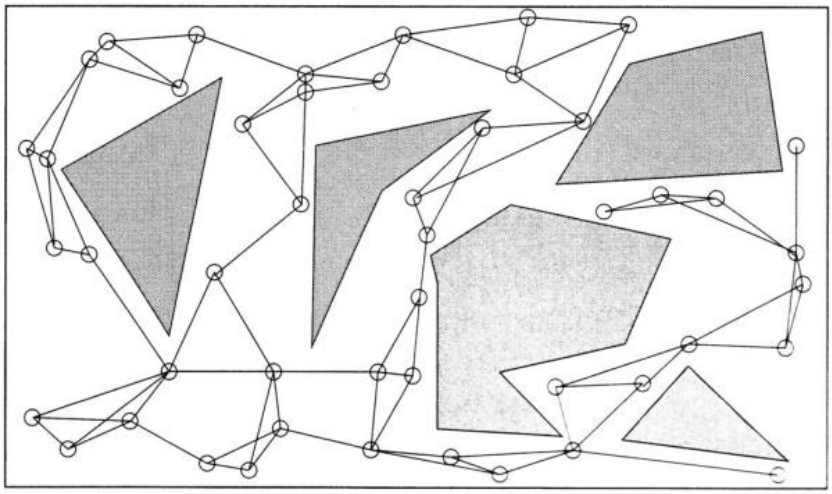


Figure 5. Pre-processed road map [62]

4.2.2.1.4 Summary of Limitations

For grid-based algorithms, because of the grid map, the computation cost increases dramatically for large scale and high-dimensional environment [55] [56] [54].

Regarding evolutionary algorithms, they face the same problem as grid-based algorithms. In addition, it is difficult to tune the parameters appropriately, and they can be easily trapped in the local minimum [54] [57] [58].

The major disadvantage of the sampling-based algorithms is that the solution is not always optimal [61] [63].

4.2.2.1.5 Discussion and Selection

Because of the difficulty of parameters tunning and computational cost, the evolutionary algorithms are listed as the third option. The grid-based algorithms are chosen regardless

of the computational cost because RTAB-map can provide 2D-occupancy map from both visual-based and laser-based SLAM. Also, A* is already integrated into ROS as one of the main global planners [64]. Therefore, A* is selected for the future development of the 3D simulator.

4.2.2.2 Local Path Planner

In this section, three local planners that are currently available in ROS will be reviewed: 1) Dynamic Window Approach (DWA) local planner [65], 2) EBAND local planner [66], and 3) Time Elastic Band (TEB) local planner [67].

4.2.2.2.1 DWA local planner

In [65], a Dynamic-Window Approach (DWA) was introduced to navigate the robot using a 2D occupancy map. DWA planner extracts samples of linear velocity (dx, dy) and angular speed (dw) from the robot's control space. The extracted samples are then used to predict the possible moving trajectories in a short time, as shown in Figure 6. The infeasible trajectories (collided with obstacles) are discarded. After that, all possible trajectories are evaluated by an objective function with characteristics including proximity to obstacles, proximity to the goal, proximity to the global path, and speed. The best trajectory is then fed to the base controller, which converts the data into the motor command to control the mobile base. The process is repeated for each discrete time and is terminated when it reaches the goal. The experiments are conducted in a 2D corridor with static obstacles. Results showed that the DWA planner yields a very robust and safe collision avoidance. The robot also reacts very fast, even with a high moving speed.

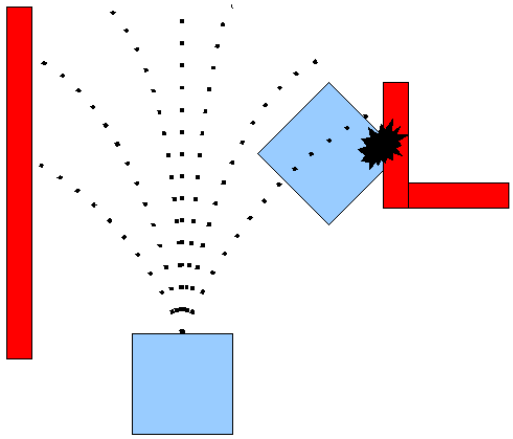


Figure 6. Estimated moving trajectories in the DWA planner [65]

4.2.2.2.2 TEB local planner

[66] proposed a local path planning algorithm called Time Elastic Band (TEB) local planner. It gathers safety distance from the obstacles, velocity and acceleration constraints, and geometric and kinematic constraints of the mobile robot to generate a sequence of vehicle posse (x and y position, and orientation) for each discrete of time [66] [68]. Each parameter is assigned with a weight (w). TEB planner then modifies the global trajectory based on the poses generated. Similar to the DWA planner, the required linear and angular velocity will be fed to the base controller moving the robot to follow the trajectory. The experiments illustrated that the TEB local planner yields a smooth moving trajectory for the mobile base [66]. This method is highly adaptive to different robot configurations and applications.

4.2.2.2.3 EBAND local planner

A similar approach [67] called Elastic Band (EBAND) local planner is proposed to drive the robot and avoid moving obstacles. EBAND planner modifies the global trajectory by simulating two forces: an internal contraction force and an external repulsive force. The global trajectory is simulated to receive a tension force toward the centroid of itself, as shown in Figure 7 a) and b). The trajectory, at the same time, is repelled from the obstacles to avoid the collision. The clearance around the obstacles is depended on the required safety distance from the obstacles. Circles called bubbles are then generated along the path, and the size of the bubble is based on the room around the robot. The planner then sends the velocity command to move the robot along the trajectory in the same way as the TED planner. Results showed that the EBAND planner has a similar outcome as the TEB planner. Both planners generate smooth trajectories and avoid the dynamic obstacle. However, instead of modifying part of the global trajectory, EBAND makes the incremental adjustment to the entire path.

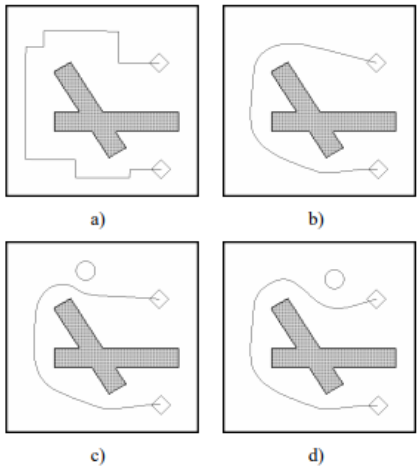


Figure 7. a) a path generated from the global planner. b) Applying both tension and contraction force. c.d) The modified path due to a new moving obstacle [67]

4.2.2.2.4 Summary of Limitation

For DWA_local_planner, the biggest drawback is that it does not consider the moving base's kinematic limitation. In another world, the robot will not keep a safe distance from obstacles when moving along the trajectory, which might cause collision [69].

Regarding TEB_local_planner and EBAND_local_planner, the direction of the moving obstacles might worsen the path planning. As shown in Figure 8, if an obstacle moves across the local trajectory, it will continuously deform the path like an elastic band. The path, therefore, might not be optimal. It may even fail the planning when the moving obstacle "pull" the path and moves too close to another obstacle, which will eventually cut the trajectory.

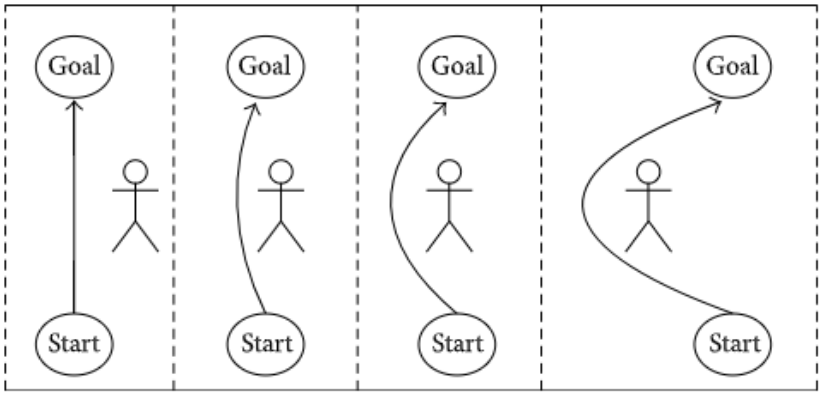


Figure 8. Pedestrian crossing the path generated by TEB planner [67]

4.2.2.2.5 Discussion and Selection

Because the disaster environment can be extremely complex and dangerous, it is critical to keep the mobile robot from colliding with the obstacles [1]. Once the robot is stuck, it is almost impossible for the human to retrieve the robot. Therefore, TEB_local_planner and EBAND_local _planner are preferred because they provide clearance room from the obstacles and smooth the moving trajectory.

4.2.3 Victim Identification

After the 3D simulator and robot navigation are chosen, the last objective is for the robot to search and identify the victim. There are five conventional equipment types used for this purpose: electric visual detecting device, fibre optic detector, thermal imaging detector, canine, and sound detector [70]. However, Gazebo only provides thermal camera and video camera plugins. Therefore, this section will review the literatures developed based on these two sensors: 1) thermal-based detection [71] [72] [73] and 2) visual-based detection [74] [75] [76] [77] [78].

4.2.3.1 Thermal-based Detection

In this section, papers pertaining to thermal-based algorithms will be discussed in detail. These methods are: 1) Heat Map Localization [71], 2) Histograms of Oriented Gradients (HoG) [72], and 3) Census Transform Histogram (CENTRIST) [73].

In [71], a heat map is generated based on readings from the thermal sensor. The heat map is shown as a 2D grid of heat pixels, which are coloured based on the colour gradient and the thermal readings (Table 4). The victim detection on the heat map is divided into two parts: offline detection and online detection. Offline detection requires a completed heat map. The pixel whose temperature is higher than a threshold is marked as a warm pixel. The warmest pixel in a continuous cluster of warm pixels is then detected as potential victims and can be confirmed as real victims based on other sensor input. Regarding online detection, the heat map is continuously updated based on the sensor input. Three algorithms are proposed to run online detection: greedy victim detection, forward victim detection and backward victim detection. The backward detection yields the highest accuracy. When a warm pixel is detected, it will iteratively mark the pixel with the highest temperature in the sensor range as the victim. If the same pixel is selected at least

three times, this pixel will be confirmed as the victim and added to the victim list. Two experiment scenarios from the RoboCup GermanOpen 2011 were created to test the algorithms. The results showed that the algorithm was able to detect most of the victims in the scene.

Table 4. Colour gradient in the heat map [71]

Temperature Range	Color Gradient
0° C - 20° C	Dark Blue - Dark Green
20° C - 30° C	Dark Green - Yellow
30° C - 38° C	Yellow - Orange
38° C - 45° C	Orange - Red

[72] proposed a different method called Histograms of Oriented Gradients (HoG) to detect victims through thermal imaging. The captured thermal image is first pre-processed to a grayscale image and equalized. The weight of each pixel and the edge orientation of the object can then be calculated. Pixels are categorized into several bins, and a histogram is generated based on each bin's weight. After that, the HoG features are classified through trained Support Vector Machines [79] to detect human presence. The method was examined based on the author's dataset, which contains subjects with different poses, appearance, illumination, and background. The result showed that HoG could successfully detect over 95% of the human presence in the images.

Similarly, the author in [73] classifies the thermal images through a trained Support Vector using features extracted. However, instead of using HoG features, they implemented a new descriptor called Census Transform Histogram (CENTRIST) [80]. The performance of the proposed method is evaluated based on the Detection Error Tradeoff [73]. The results showed that classification using CENTRIST features is more accurate than using the HoG features.

4.2.3.2 Visual-based Detection

In visual-based detection, the robot can detect the human body or parts of the human through the RGB or RGB-D image. It often involves using machine learning to classify the image [81]. The author in [81], for the first time, proposed and tested five machining learning techniques to identify victim in a cluttered USAR environment: FPN with Faster

R-CNN [74], SSD [75], YOLOv2 [76], YOLOv3 [77], and RetinaNet [78]. In this section, the methods mentioned above will be discussed in detail.

Feature Pyramid Network with Faster Region-Based Convolutional Neural Networks (FPN with Faster R-CNN) was first introduced in [74] to identify different objects. FPN implements ResNets [82] to generate multiple feature maps using a single RGB image as input. The extracted feature maps are then passed to Fast R-CNN [83], a region-based object detector, to extract features. The model is tested on the 80 category COCO detection dataset, and the detection is evaluated in both COCO-style Average Precision (AP) and PASCAL-style AP. The results show that the model can achieve 36.2 COCO-style AP in object detection.

Single Shot Detector (SSD) [75] is the first deep-network-based object detector that can generate bounding boxes with no features required. It uses the pre-trained VGG-16 network to generate feature maps whose size is decreased progressively through a set of different convolutional networks. The feature map is presented as a grid, and each layer outputs several bounded regions (bounding boxes) for the feature maps and predicts the object classification within those regions. The model was trained on the PASCAL VOC dataset consisting of 4952 images and the COCO dataset. The result shows that SSD can achieve over 70% mean average precision from PASCAL VOC2007 testing [75]. The performance of the SSD model is further evaluated using the detection analysis tool from [83].

You Only Look Once version 2 (YOLOv2) [76] is a real-time object detector that can detect over 9000 different categories of objects. The RGB image is pre-processed into an NxN grid and passed to a single pre-trained CNN called Darknet-19. The CNN then predicts five bounding boxes for each cell in the grid. Bounding boxes with no object detected are then discarded. Eventually, several bounding boxes with the corresponding classification will be generated. The author used the standard ImageNet 1000 database for classification training and PASCAL VOC for object detection training. The experiments illustrate that the mean average precision is 73.4%.

YOLOv3 [77] is an improved version of the YOLOv2 detector. It implements a different CNN for feature extraction called Darknet-53, consisting of 53 convolutional layers.

Besides, instead of predicting the bounding box in a single stage, YOLOv3's prediction is made at three different scales. It upsamples the feature maps to get more semantic information so that it can detect more small objects.

The proposed detector in [78] is called RetinaNet Detector, consisting of one backbone network and two subnetworks. The implemented backbone network is the FPN [82], and it outputs multi-scale feature maps from a single RGB-D image. The extracted feature maps are then passed to the subnetworks. The first subnetwork classifies the feature maps through 4 3x3x256 convolutional layers, and the second subnetwork generates bounding boxes for each feature map through four convolution layers of the same size. The model was tested based on the COCO dataset, and the detection performance is evaluated as COCO-style AP, which is 39.1 from the experiment result.

The results in [81] show that all five models can perform victim identification in a dark and cluttered environment. RetinaNet outperforms the rest when using the RGB-D and RGB database. The overall performance increases when using the RGB-D dataset because the depth image is not affected by illumination changes.

4.2.3.3 Summary of Limitation

In thermal-based detection, it is possible to detect and identify victims in the USAR environment. However, thermal images lack semantic information to determine the condition of the victim. Also, when the victim's temperature is close to the surrounding, the victim's accurate location might not be obtained [71].

Regarding visual-based detection, the current state-of-art can detect over a wide range of objects through RGB or RGB-D images. However, the detection accuracy suffers from poor lighting [81]. Besides, the range of the camera view is very limited, and the view can be easily blocked by obstacles and dust. Therefore, it is almost impossible to localize a victim with only a visual camera [70].

4.2.3.4 Discussion and Selection

It is found that thermal cameras and visual cameras are complementary when searching and identifying the victims [71] [70]. In USAR, the thermal camera is first deployed to determine the victim's position, and the victim's condition will then be decided through

the visual camera [70]. The online victim detection through heat map [71] is preferred in this project to localize the victims. Because a thermal camera plugin "hector_gazebo_thermal_camera" is available in the ROS. It works as a combination of thermal and visual cameras. For victim identification, instead of using deep learning, a different approach will be built to send victim information because the model requires a large dataset and other works to accurately detect the victim, which is not feasible in the timeline. The victim information will be sent to the robot through a ROS topic when the robot detects the victim. The proposed approach is sufficient for the researcher to test the multi-robot coordination algorithms since they only demand victim information as input.

5. Final Simulator Design

This section will describe the final simulator design in detail, which is divided into five components: 1) 3D Environment Simulation, 2) Frontier Exploration, 3) Map Merging, 4) Victim Identification, and 5) Multi-Robot Frontier Exploration

5.1 3D Environment Simulation

In this section, a simulated world was built consisting of all the components from the project requirements, which divides this section into four categories: 1) Terrain, 2) Unstructured Obstacles, 3) Natural Environment and 4) Victim.

5.1.1 Terrain

In Gazebo, terrain surface can be simulated through a digital elevation model (DEM) or a height map [84]. A DEM is a 3D model of a terrain surface created using sensors, for example, LIDAR, radar, and cameras. In comparison, a height map is a raster image of a terrain surface in grayscale. Each pixel stores the elevation data of the terrain surface. The DEM file of a specific region of interest can be downloaded from Global Land Cover Facility, which provides a high-resolution topographic database. The heightmap can also be downloaded from the web. Also, both DEM and height maps can be manually created in Blender, a free-open-source 3D creation software.

In order to insert a terrain surface into a Gazebo world, the following lines (Figure 9) should be added to the corresponding .world file.

```
<model name="heightmap">
  <static>true</static>
  <link name="link">
    <collision name="collision">
      <geometry>
        <heightmap>
          <uri>file://media/dem/mtsthelens_129.dem</uri>
          <size>150 150 50</size>
          <pos>0 0 -685</pos>
        </heightmap>
      </geometry>
    </collision>

    <visual name="visual_abcedf">
      <geometry>
        <heightmap>
          <texture>
            <diffuse>file://media/materials/textures/dirt_diffusespecular.png</diffuse>
            <normal>file://media/materials/textures/flat_normal.png</normal>
            <size>1</size>
          </texture>
          <texture>
            <diffuse>file://media/materials/textures/grass_diffusespecular.png</diffuse>
            <normal>file://media/materials/textures/flat_normal.png</normal>
            <size>1</size>
          </texture>
          <texture>
            <diffuse>file://media/materials/textures/fungus_diffusespecular.png</diffuse>
            <normal>file://media/materials/textures/flat_normal.png</normal>
            <size>1</size>
          </texture>
          <blend>
            <min_height>2</min_height>
            <fade_dist>5</fade_dist>
          </blend>
          <blend>
            <min_height>4</min_height>
            <fade_dist>5</fade_dist>
          </blend>
          <uri>file://media/dem/mtsthelens_129.dem</uri>
          <size>150 150 50</size>
          <pos>0 0 -685</pos>
        </heightmap>
      </geometry>
    </visual>

  </link>
</model>
```

Figure 9. SDF file of a terrain model [84]

The <uri> session specifies the location of the DEM file or the heightmap image. Gazebo will automatically detect the DEM file or the plain image and transform it into a terrain model. The terrain's size and position can be changed in <size> and <pos>, respectively.

A random heightmap (Figure 10. left) was chosen to demo the terrain modelling. As can be seen in Figure 10, Gazebo automatically transforms the grayscale heightmap into a 3D terrain model.

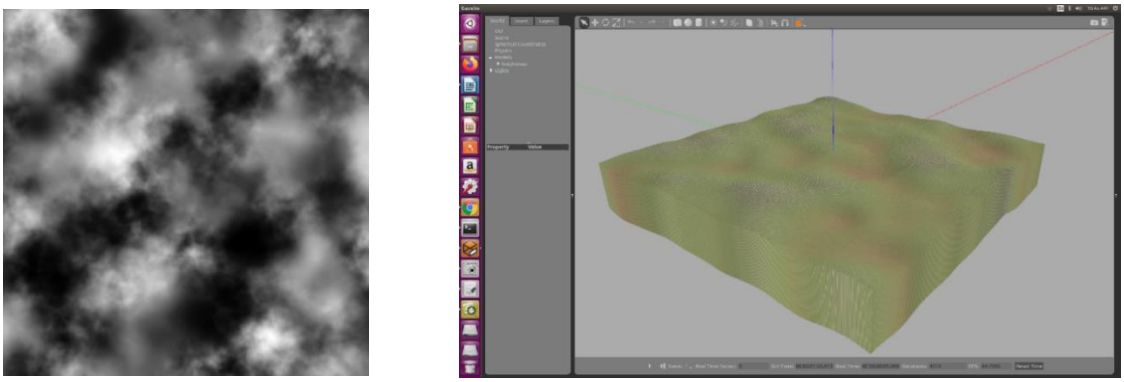


Figure 10. Height map (left) and terrain model (right)

5.1.2 Unstructured Obstacles

Gazebo provides various unstructured obstacles models in the online library, such as collapsed fire stations, collapsed houses, and barriers (Figure 11). Any model that is not available in the online library can also be created in Blender and imported to Gazebo [85]

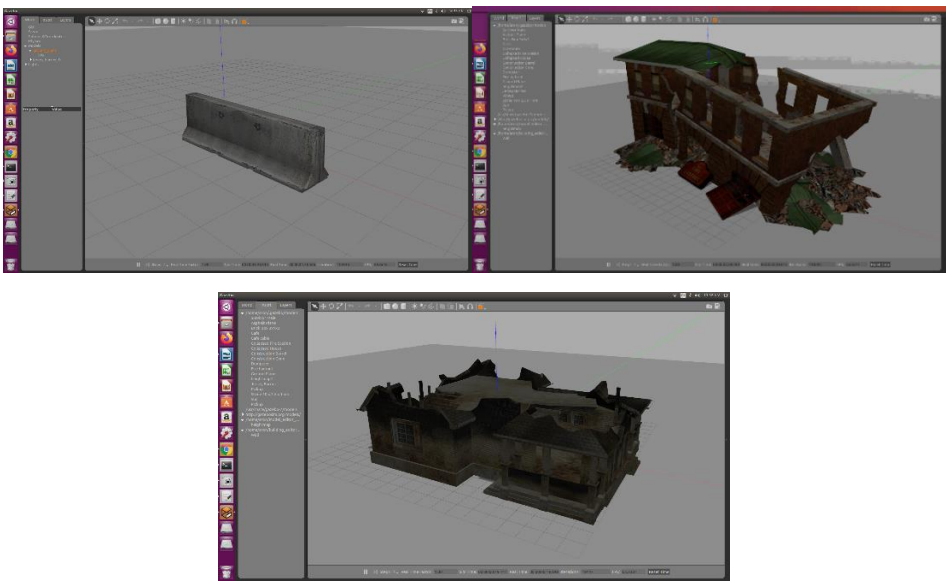


Figure 11. 3D models of unstructured obstacles

5.1.3 Natural Environment

Like unstructured obstacles, Gazebo also provides various models that we can find in the natural environment, such as oak trees and pine trees (Figure 12). External 3D models can also be imported to Gazebo.

Figure 12. 3D models of trees

5.1.4 Victim

Unfortunately, Gazebo does not provide a human model. However, the human model can be created using an external software called MakeHuman [86]. It can create human

models of different gender, ages, height, colour, and posture. The model can then be directly exported as a .dae file and import to Gazebo.

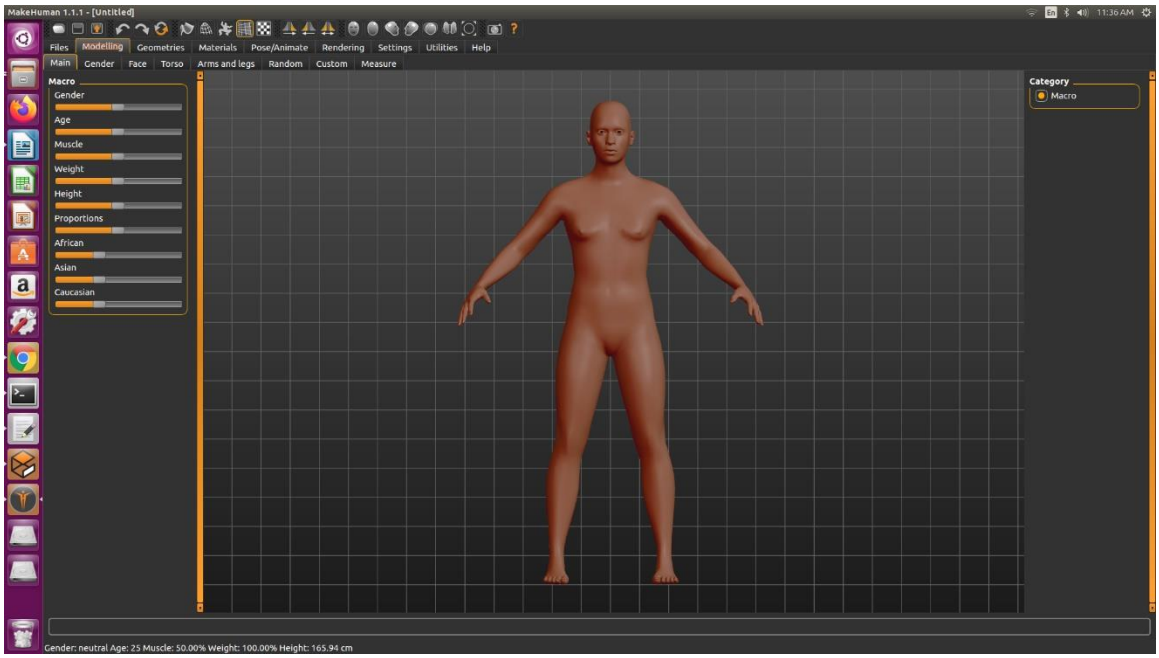


Figure 13. MakeHuman user interface

5.1.5 Demo

A world containing all the required features with uneven terrain was created to demo the capability of the 3D simulator. Figure 14 shows the scene has collapsed buildings, vehicles, barriers, trees, and victims. The red walls are built intentionally to avoid the robot exploring unboundedly.

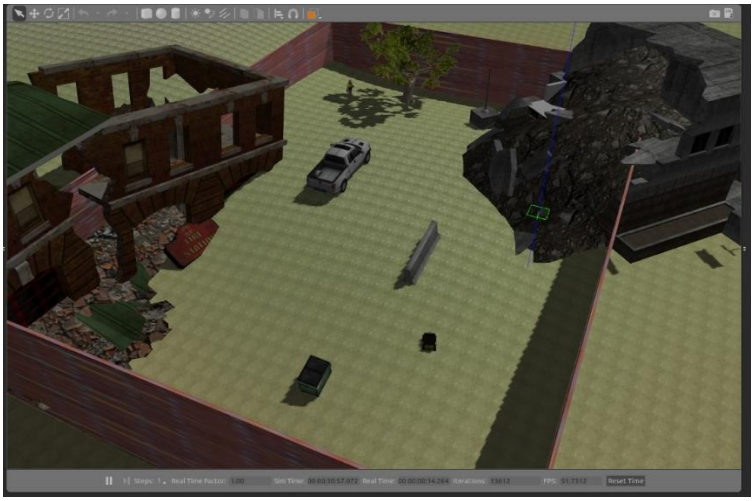


Figure 14. The simulated scene in Gazebo

5.2 Frontier Exploration

This section discusses the simulation for robot frontier exploration. The frontier exploration requires four critical components: 1) Robot, 2) SLAM, 3) Navigation, & 4) Exploration Algorithm.

5.2.1 Robot

The robot selected for autonomous navigation is HUSKY [87]. HUSKY is a non-holonomic four-wheel robot. Its rugged structure and high-torque drivetrain are suitable for robotics research in all terrains, including flat terrain, uneven terrain, and cluttered terrain. Also, HUSKY is readily integrated with ROS as a *ros-kinetics-husky-simulator* package [87].

A HUSKY robot can be spawned in any world in Gazebo through the *spawn_model* node in *spawn_husky.launch* file in the husky_gazebo package. By default, it also initiates the robot_staate_publisher node to calculate the robot's forward kinematics and publish the transformation result through *tf*.

The robot's configuration will be discussed amply in the Experiment section.

5.2.2 SLAM

RTAB-Map [13] was selected as the SLAM method for autonomous navigation. Visual-SLAM mode was first tested because it yields better performance than the lidar-SLAM [13]. RTAB-Map requires stereo-camera input. However, HUSKY is not equipped with a stereo-camera in default. A Gazebo stereo-camera plugin called *libgazebo_ros_multicamera.so* was installed and added to the HUSKY Robot URDF file to simulate the stereo-camera [21]. It will create two camera topics (*stereo_camera_left* and *stereo_camera_right*) and output the raw image (Figure 15).

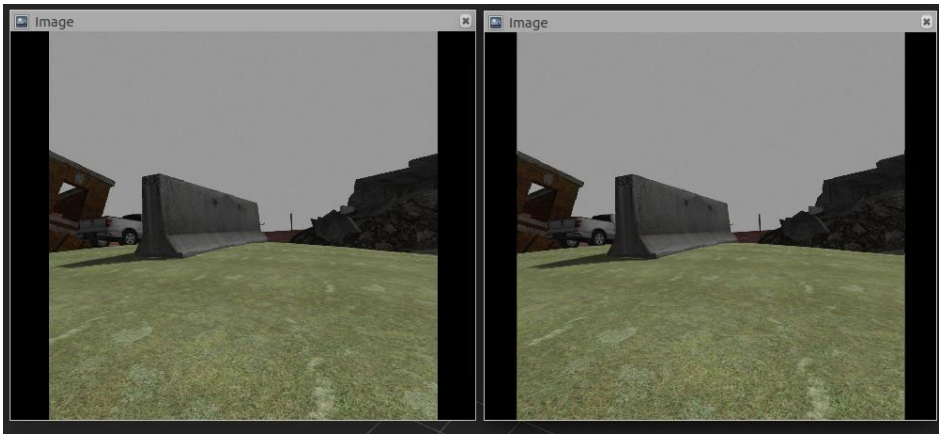
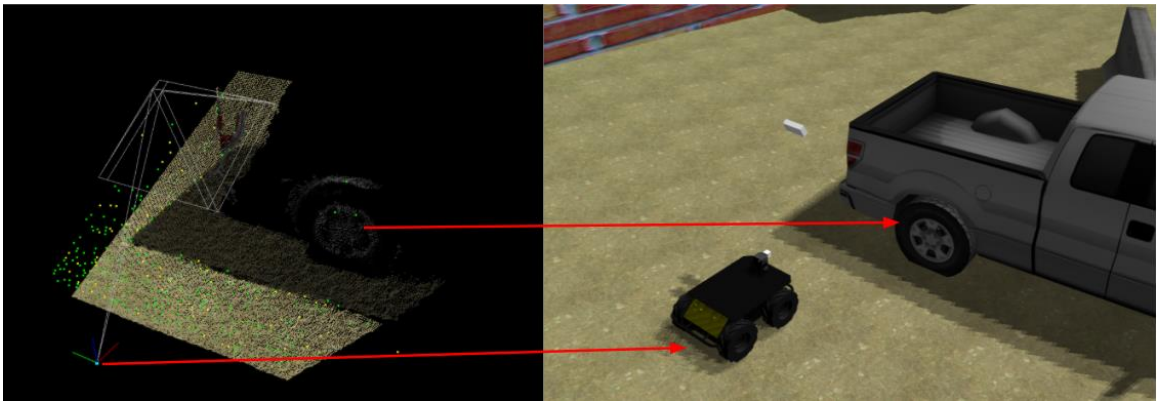


Figure 15. The Raw Image Output from Stereo Cameras

The raw images are then passed to the stereo_image_proc node to perform rectification and de-mosaicing [13]. It projects two raw images into one common plane so that RTAB-Map can better match the features.

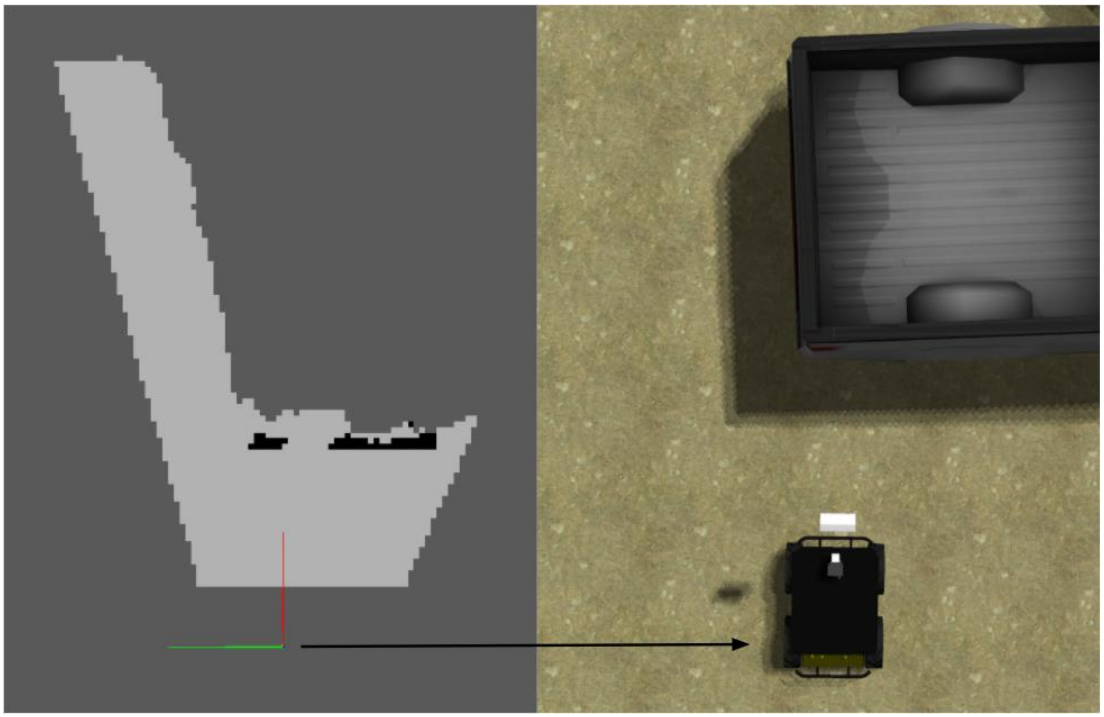
As shown in the navigation stack, the robot's odometry is required for the SLAM. RTAB-Map can estimate the robot's odometry based on the pre-processed stereo image, robot transformation *tf*, and the base frame information using the *stereo_odometry* node. The estimated odometry information is then passed to the *rtab_map* node, which is the core of the *rtabmap_ros* package to start the SLAM. A 3D map consisting of point cloud data extracted from the image is generated (Figure 16). The black area represents the unexplored environment, and the green and yellow pixels are the features extracted from the stereo images.



a) *3D Point cloud map* *b) Simulated scene*

Figure 16. 3D point cloud map of the environment

Meanwhile, a 2D occupancy grid is created, as shown in Figure 17. The coordination represents the current position of the robot. The white and black pixels are the free space and the space occupied by the obstacles, respectively, while the grey pixels are unexplored space. The obstacle is defined as any object higher than a threshold that the user can change.



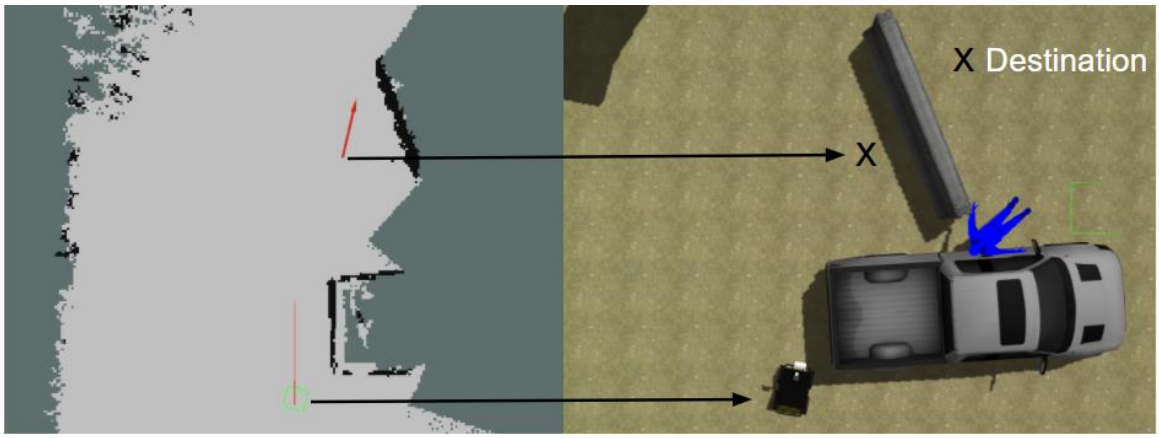
a) *2D occupancy map* *b) Bird view of the simulated scene*

Figure 17. 2D occupancy grid of the environment

5.2.3 Navigation

After the robot generates the 2D occupancy map and localizes itself in the map, the next step is to navigate the map's robot. *move_base* package was implemented to achieve the global navigation in the simulated environment [88]. *move_base* is the most commonly used navigation package in ROS and is also the HUSKY's officially recommended package [88]. *move_base* package contains several global (e.g. A* algorithm) and local planners (e.g. *DWA and teb_local_planner*). Based on the 2D occupancy grid, *move_base* can generate a global moving trajectory and local trajectory for robot navigation when a destination is set, as shown in Figure 18. The red arrow indicates the goal and the robot's final orientation, while the red line and green line are the global trajectory and the local

trajectory, respectively. The *move_base* then sends the velocity command to the robot to follow the path to reach the destination. The green square represents the moving base.



a) *2D occupancy map* *b) Bird view of the simulated scene*

Figure 18. Global and local trajectories from move_base

5.2.4 Exploration Algorithm

Now, once the navigation stack is set up, the final step is for the robot to decide on its own to explore the unknown environment. A package called *m-explore* was implemented to run the frontier exploration greedily [89]. It subscribes to the 2D occupancy grid from RTAB-Map and calculates the map's frontiers as a set of points. It then ranks the frontiers according to the cost (distance to the robot). Frontiers whose cost is higher than a threshold will be discarded. The algorithm finds the centroid of the remaining frontiers and publishes it to the *move_base*. As shown in Figure 19, the blue line represents the frontier in the 2D grid.

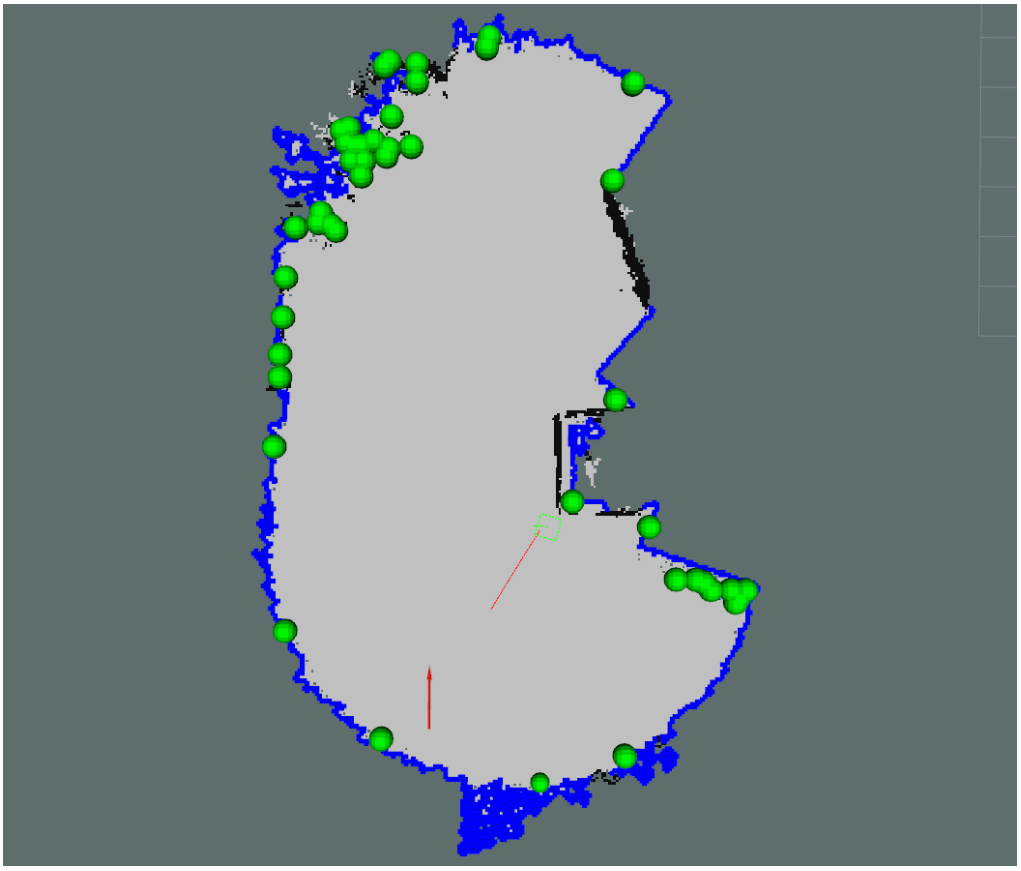


Figure 19. Frontiers calculated in the 2D occupancy grid

If the robot can not reach that frontier in a specified period, that point will be thrown into the blacklist. Until there is no more frontier found, the algorithm will terminate the exploration. In this way, the robot can eventually explore the entire space. Figure 20 shows the flow diagram of the overall simulation.

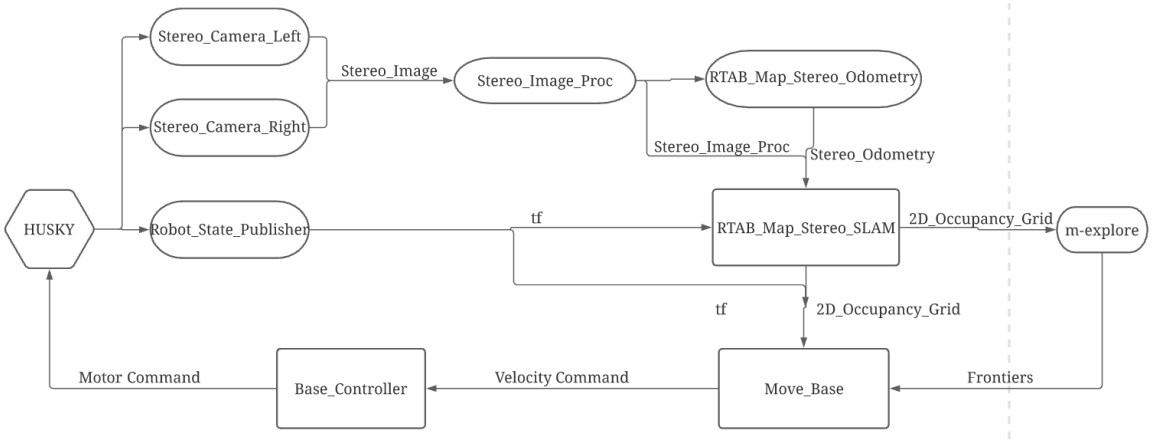
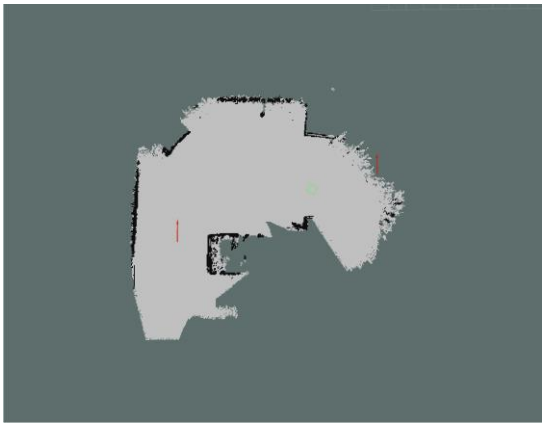


Figure 20. Flow chart of the overall simulation

5.3 Map Merging

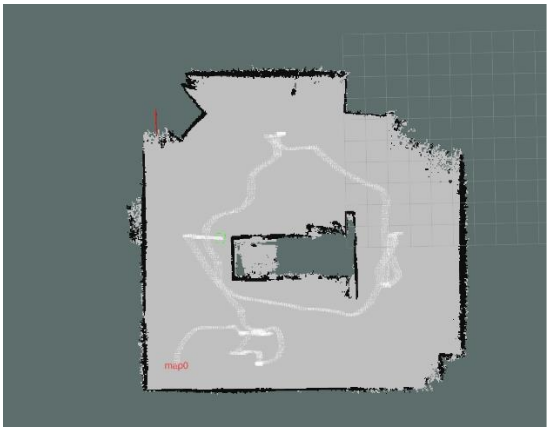
The map merging algorithm designed for this project is to greedily combine the 2D occupancy maps based on the robots' initial positions. The robot which calls the map merging node will be selected as the main robot, and its map will be chosen as the main map, while the maps of the rest robots will be identified as secondary maps. The secondary maps are shifted based on their relative positions to the main robot and added to the main map. The merged map is published on a new topic. Because the merged map is built upon the main map, the main robot can directly subscribe to the merged map and use it for navigation. The algorithm works for an arbitrary number of robots. Figure 21 shows a demo of the map merging using two maps.

a) 2D occupancy map – robot_1



b) 2D occupancy map – robot_2

c) Merged map



d) Ground Truth

Figure 21. Map merging demo

However, the algorithm has some limitations, and they will be discussed in the Experiment section.

5.4 Victim Identification

As mentioned in the victim identification section's literature review, the machine learning model requires a large dataset to be appropriately trained to identify the victim, which is not feasible for the project timeline. Therefore, the victim identification is simulated in a more straightforward approach. The first step is to detect the presence of a victim in the camera image. A thermal camera plugin called *hector_gazebo_thermal_camera* is implemented [90]. It colours the objects with heat signature as white (value equal to 255 in greyscale) in the thermal image, and the rest of the image will be replaced with colour with much-reduced intensity (Figure 22). Any object with its emissive and ambient material properties set to maximum red will be detected as heat signature and marked as white pixels in the thermal image. As shown in Figure 22, the red circles' emissive and ambient material properties are set to maximum red, and they are represented as white circles in the thermal image. A victim model is created through MakeHuman [86], and its cloth's emissive light is set to maximum red through the SDF file. In this way, whenever the robot detects a heat signature in the thermal image, the robot knows the victim's presence, as shown in Figure 24. After that, a ROS topic containing victim information, such as victim position, orientation, and health condition, is created. Each time the robot detects a victim, it can subscribe from that topic to obtain corresponding victim information.

However, only one victim can be simulated in this way because the value of the heat signature is set. If the robot can not consider the object's shape, any object with the heat signature will be identified as the same victim. To simulate different victims, a modification was made to the *hector_gazebo_thermal_camera* plugin. When an object with its emissive and ambient properties set to maximum green is detected, the heat signature value will be adjusted to 200. When the ambient and emissive properties are set to maximum blue, the heat signature's corresponding value is 150. Therefore, the robot can identify three different victims based on their heat signature, as shown in Figures 23

and 24. More victims can be simulated if more types of heat signatures are added to the simulation.

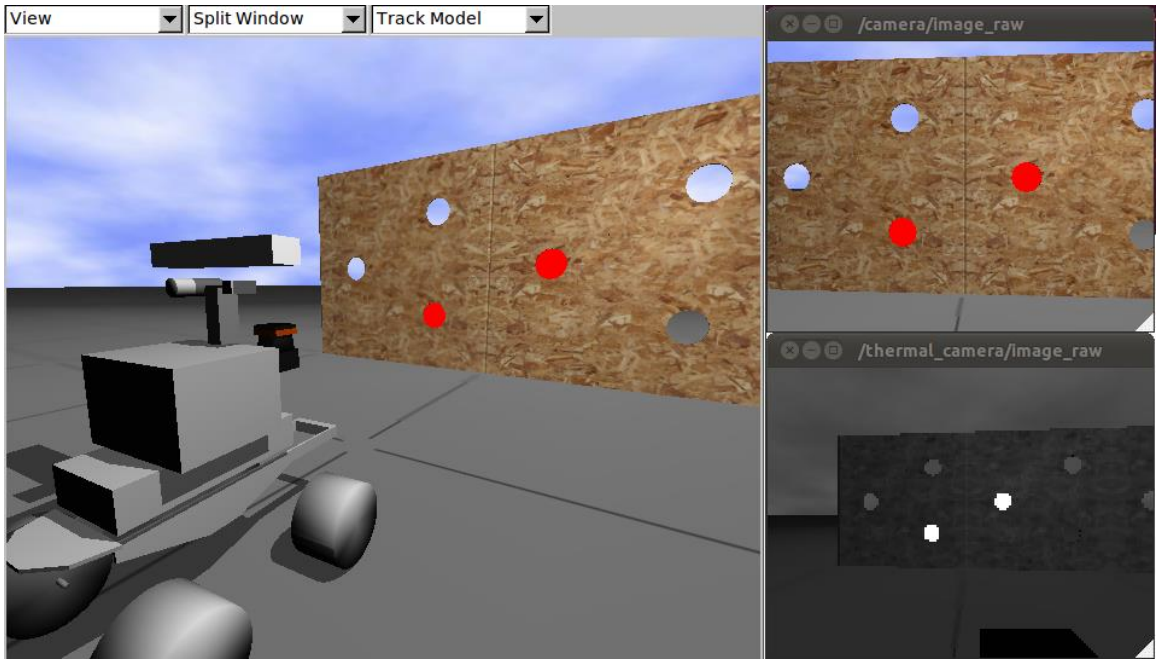


Figure 22. Heat signature and processed thermal image

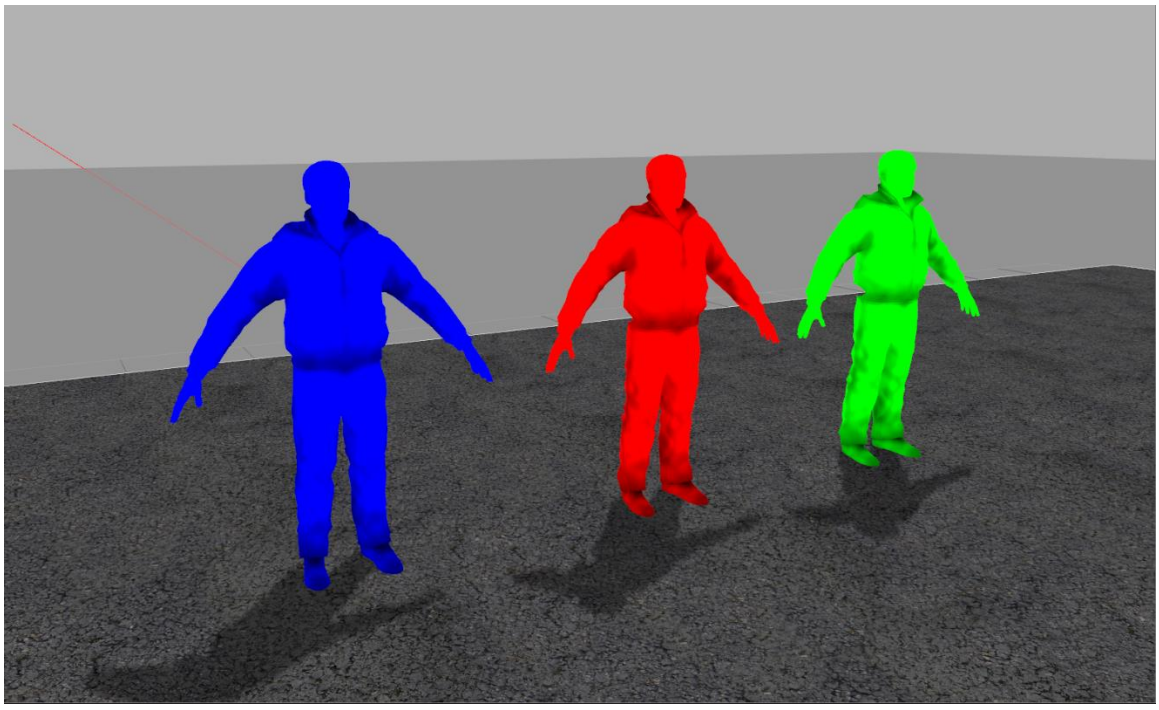


Figure 23. Victim models in the visual camera view

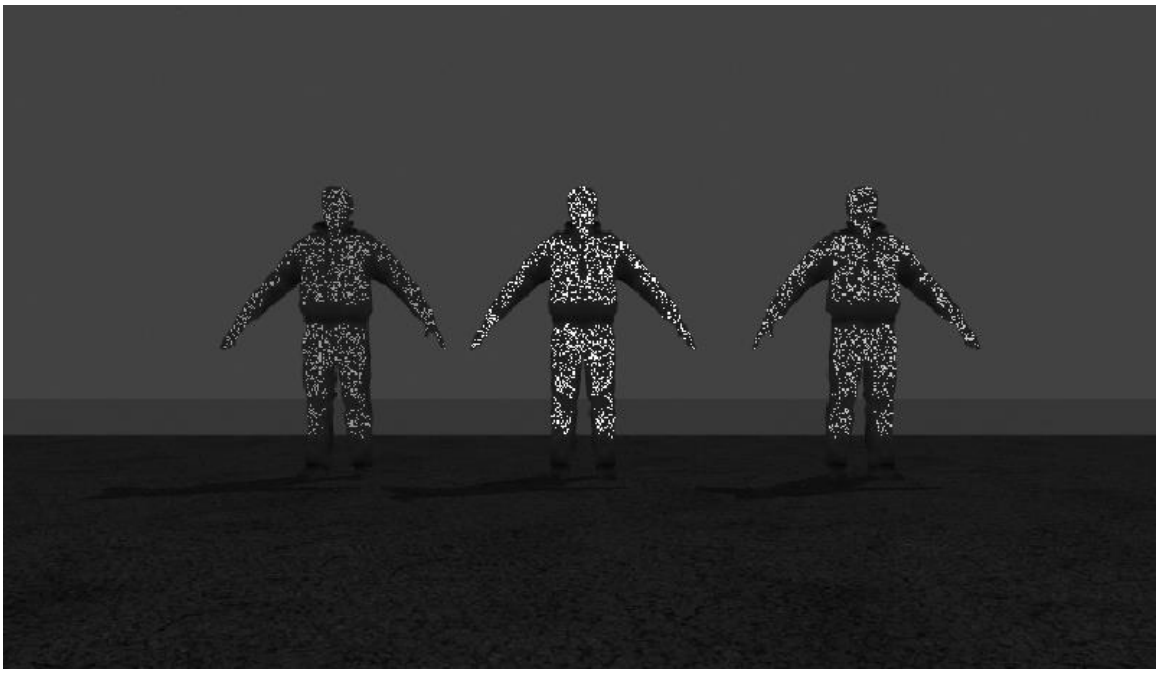


Figure 24. Victim models in the thermal camera view (left to right: blue, red, and green)

5.4 Multi-Robot Frontier Exploration & Victim Identification

Since single robot frontier exploration was successfully simulated, more robots can be added to simulate multi-robot frontier exploration & victim identification.

The <group> tag in the ROS launch file makes it possible to run multi-robot independently in a single world. The simulation described in the previous sections can be assigned with a namespace [92], for example, h_1. The simulation will run in the specified namespace. Every topic published in this simulation will have a prefix /h_1. A second simulation can then start with a namespace h_2. In this way, two robots can both run frontier exploration independently in the same world. Similarly, more robots with different namespaces can be added to explore the world.

6. Experiments & Results

This section will describe 1) the experiments performed to validate the 3D simulator design and discuss 2) the results concluded from the experiments.

6.1 Experiments

In this section, 1) Computer Specification, 2) Assumption & Limitation, 3) Robot Setup, 4) Environment Setup, and 5) Experiment Procedure will be described for future researchers to make the comparison.

6.1.1 Computer Specification

Before conducting the experiments, it is necessary to know about the hardware where the simulation is run. The computer specification is shown in the table below.

Table 5. Computer specification

Components	Model
CPU	Core i7-8700 @ 3.20GHz
RAM	16 GB
GPU	Nvidia GeForce GTX 1660 6 GB

6.1.2 Assumptions

Several assumptions were made to simplify the simulation.

1. The simulation will be run on the premise of adequate lighting conditions.

    The visual SLAM from RTAB-map suffers from poor lighting conditions. Since the 3D simulator is designed to test the coordination algorithm, a correct mapping and location should be guaranteed.

2. The initial yaw orientations of the spawned robots should be the same.

    The map merging algorithm greedily adds the maps together. It does not account for the rotational matrix between the maps. Therefore, the initial yaw orientation should be set to the same to avoid an incorrect map.

3. The 2D occupancy map size should not change during the exploration.

    The 2D occupancy map generated by RTAB-map will expand when the map goes beyond the boundary. The map's origin will move, and the robots' initial positions in the maps will consequently change, which results in an incorrect merged map.

4. The resolution of the 2D occupancy maps should be consistent.

Using different map resolutions will fail the map merging algorithm.

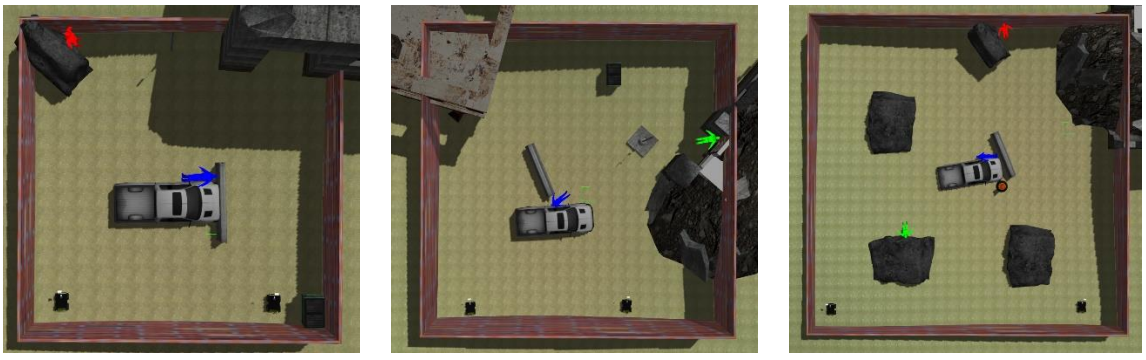
6.1.3 Robot Setup

The husky robot is modified to be equipped with one stereo camera using the Gazebo *stereo_camera_plugin* and one thermal camera using the Gazebo *hector_gazebo_thermal_camera*. The stereo-camera is located at (x = 0.8 m, y = 0 m, z = 1.5 m) with respect to the robot's origin. The stereo camera is pointing downward at a 60-degree angle with respect to the horizontal plane to achieve optimal mapping accuracy [13]. The thermal camera is located at (x = 0.4 m, y = 0 m, z = 0.4 m) with respect to the origin. The thermal camera is looking forward along the positive x-direction. The field of view of both cameras is 80 degrees.

Regarding the software setup, the robot uses RTAB-map for SLAM, and the A* algorithm and DWA planner are used for global and local path planning, respectively.

6.1.4 USAR Environment Setup

Three environments with different sizes (15 m x 15 m, 20 m x 20 m, and 25 m x 25 m) were built to test the simulator's ability, as shown in Figure 25. Each environment contains several components discussed in the 3D Environment Simulation section and three victims (red, blue, and green). The obstacles were placed randomly, and the spacing between the obstacles should be larger than the robot size. The environment setup is user-defined and can be changed based on the user's requirement.



a) 15x15 m^2 *b) 20x20 m^2* *c) 25x25 m^2*

Figure 25. USAR simulated environments

6.1.5 Experiment Procedure

Once the robot and the environments are set up, the next step is to run the frontier exploration. The experiments were divided into two tests: single robot frontier exploration and two robots frontier exploration. For single robot frontier exploration, one husky robot will be spawned at the corner of the environment and execute the frontier exploration. Similarly, in two robot frontier explorations, one robot will be spawned at the bottom left corner and the other at the bottom right corner. Both robots will start the frontier exploration at the same time. Each test was run four times for all three USAR environments.

It is important to note that the robots were separated because when two robots started from the exact location, they would easily collide due to the limited control and map updating frequency. The SLAM can not identify the robot as an obstacle rapidly, and the robot could not react in time to avoid the collision, as shown in Figure 26.

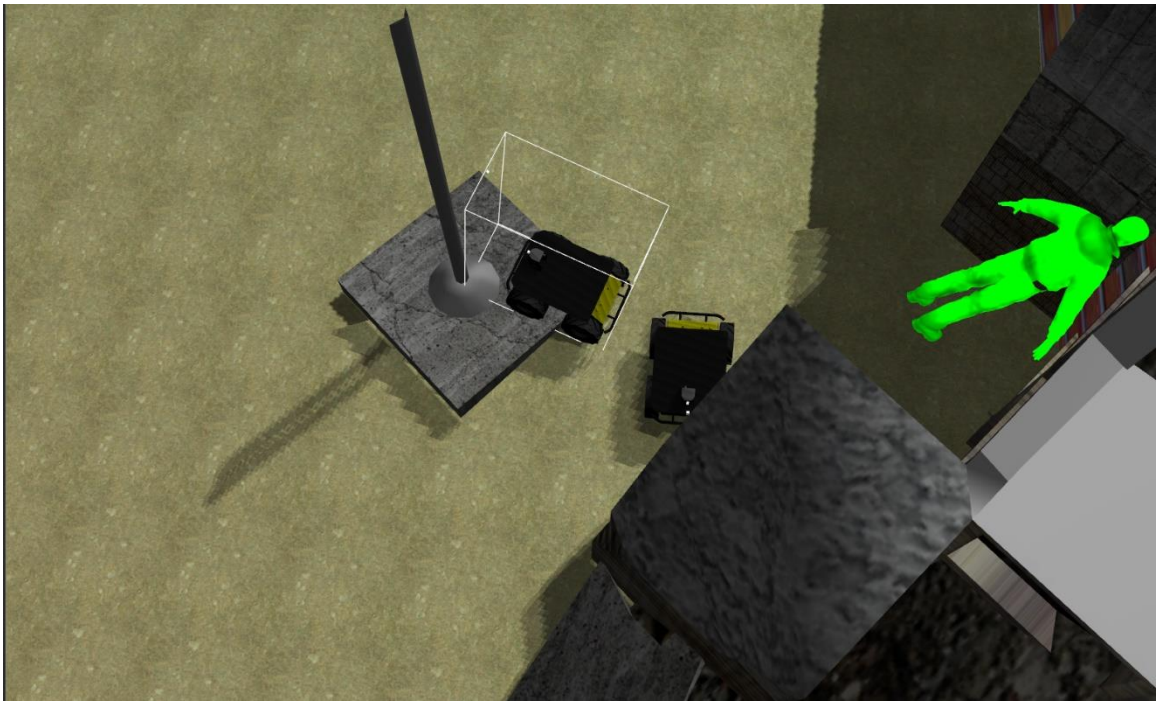
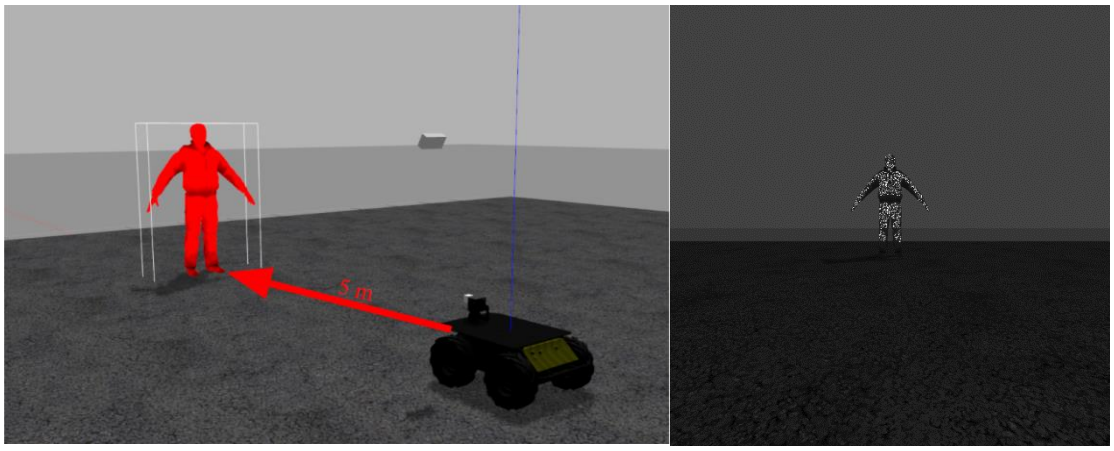


Figure 26. Robots collision during exploration

Regarding victim identification, the victim is assumed to be identified when the thermal image's heat signature takes up more than 0.3% of the pixels' total number. The 0.3%

threshold is user-defined and is obtained when the robot is 5 m in front of the victim model, as shown in Figure 27.



a) Simulated Scene *b) Thermal Camera View*

Figure 27. Victim identification distance (5 m)

For each experiment, the area explored, the distance travelled, and the victims found were recorded for each robot. The area explored is calculated based on the 2D occupancy map generated by the RTAB-map, and the distance travelled is obtained based on the *stereo_odometry* readings. For two robots frontier exploration, the area explored is calculated based on the merged map from the map merging node.

The test was terminated either when the total area explored is greater than 85% of the environment or when no more frontier was found. The 85% was user-defined and can be changed based on the user's requirement.

Due to limited computational power, the computer cannot run frontier exploration with more than two robots. When running three robots frontier exploration, the velocity control's actual frequency was so low that it resulted in jerks in the robot's movement. The obstacle avoidance failed quickly.

6.2 Results & Discussion

This section shows the results from the experiments mentioned in the previous section and the associated discussions.

Figures 27-30 indicate the average area explored with two fleet sizes in different USAR environments. The shaded area represents the range of area explored from the four trials. It is clear from the figures that the fleet with two robots explored the environment faster than the single robot in all environments. Also, the exploration rate drops over time when the area explored increases because of the greedy frontier exploration. The robot may be required to travel to the other side of the map, which is far from the current position, to continue mapping the unknown environment. Thus, the time will be wasted in travelling, which consequently reduces the exploration efficiency.

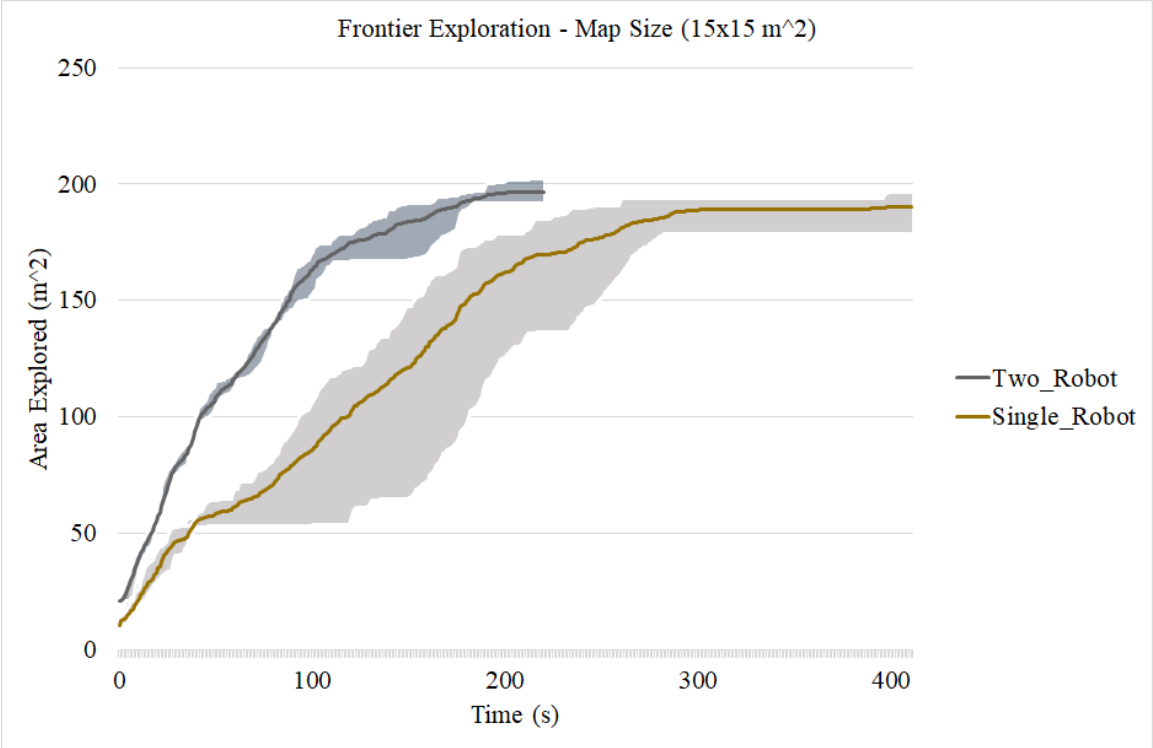


Figure 28. The average area explored in 15x15 m^2 environment

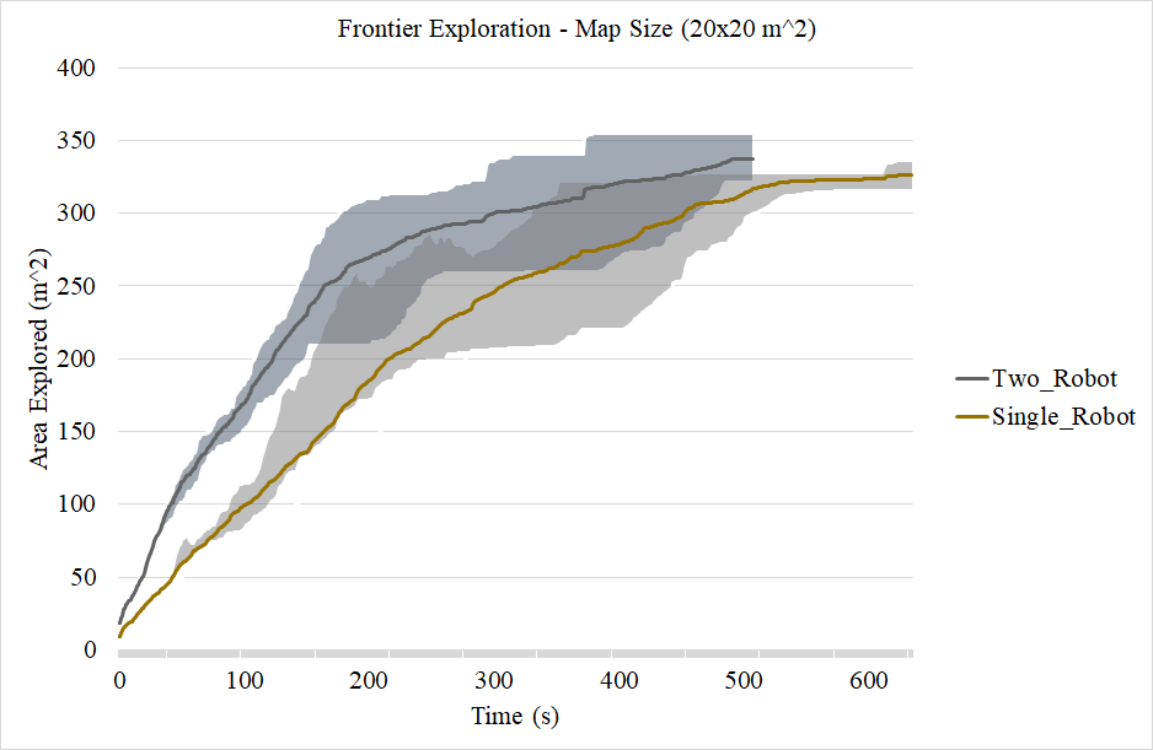


Figure 29. The average area explored in 20x20 m^2 environment

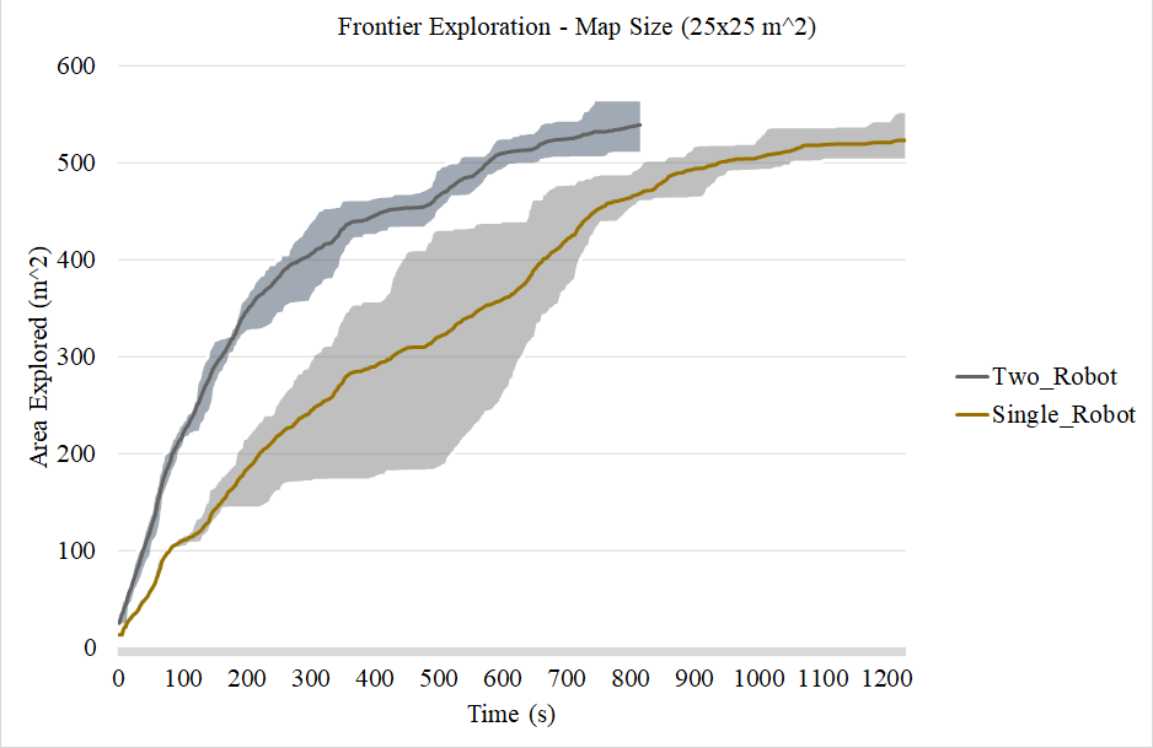


Figure 30. The average area explored in 25x25 m^2 environment

Also, as shown in Figure 31, the average distance travelled with two robots is always lower than the one with a single robot. *Two_Robot_#1* and *Two_Robot_#2* were the two robots deployed in the two-robot frontier exploration.

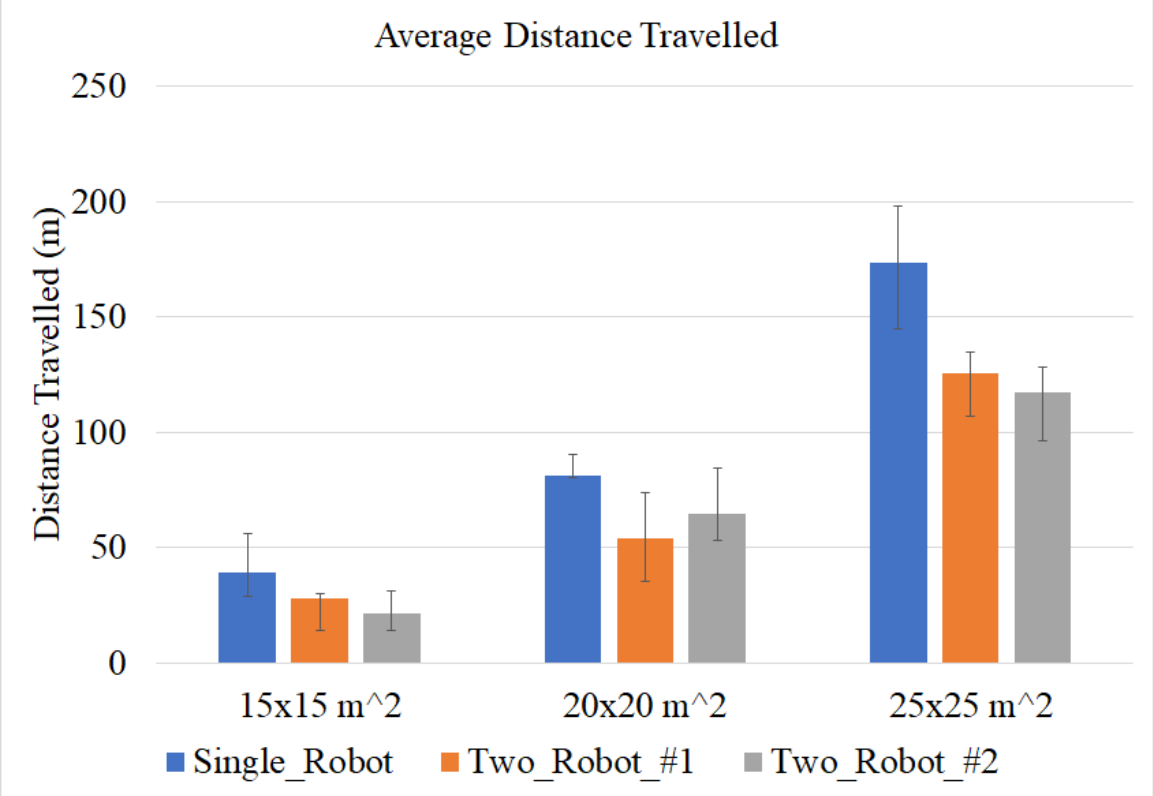


Figure 31. Distance travelled in different environments

These results are consistent with the statement that the exploration efficiency should improve with increased robot fleet size [1].

However, it should be pointed out that the average distance travelled rise exponentially with an increase in the environment size. This is because there were more frontiers found in the 25x25 m^2 environment and the robot started to travel back and forth between two areas which were far from each other, as shown in Figure 32. The greedy frontier exploration algorithm caused this inefficient travelling. For example, in the case where the robot started to travel from point 5 to point 8. When the robot reached point 8, the number of frontiers found around point 8 was less than the number of frontiers found around point 5 because it reached an obstacle. Therefore, the algorithm published a goal around point 5 to let the robot return to the starting point to continue mapping. This

repeating exploration happens more commonly in a large environment where an enormous amount of frontiers can be found. Consequently, the robot spends more timing exploring a larger environment. Compared to the path in the 25x25 m^2 environment, the paths in 15x15 m^2 and 20x20 m^2environments were cleaner and more straightforward, as shown in Figures 33 and 34.

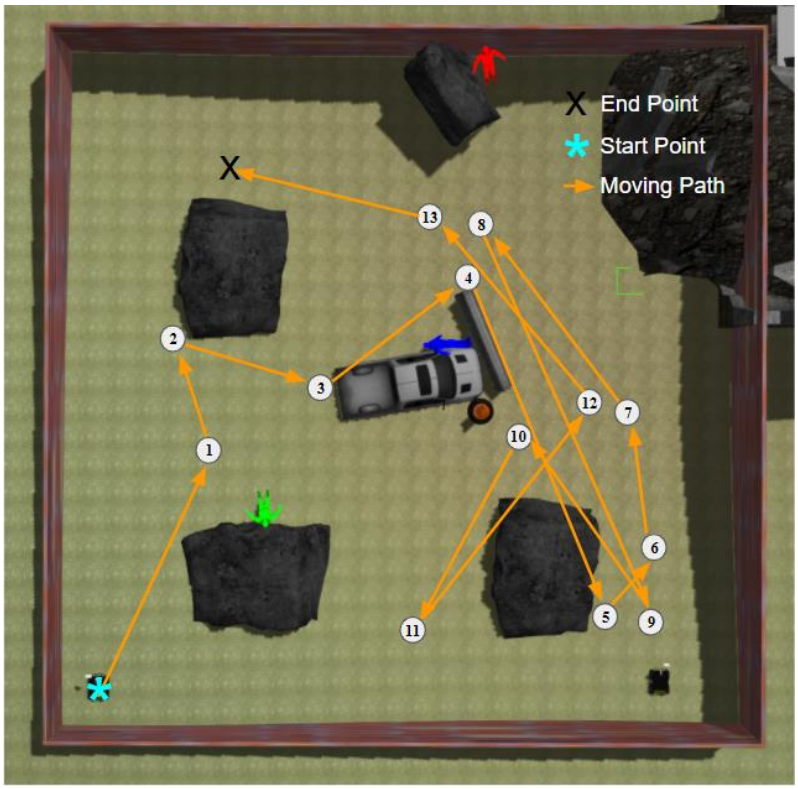


Figure 32. Exploration path in 25x25 m^2 environment

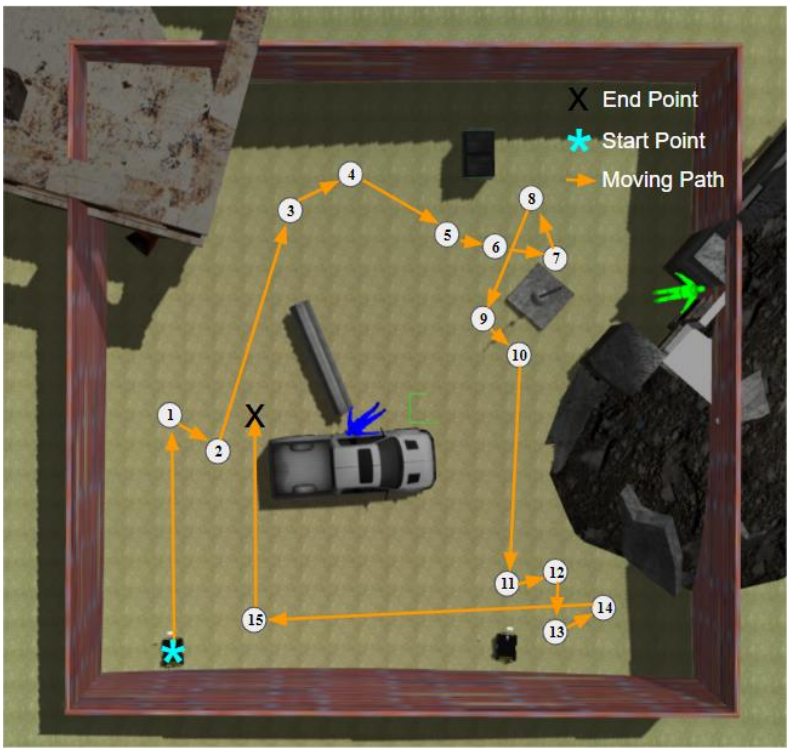


Figure 33. Exploration path in 20x20 m^2 environment

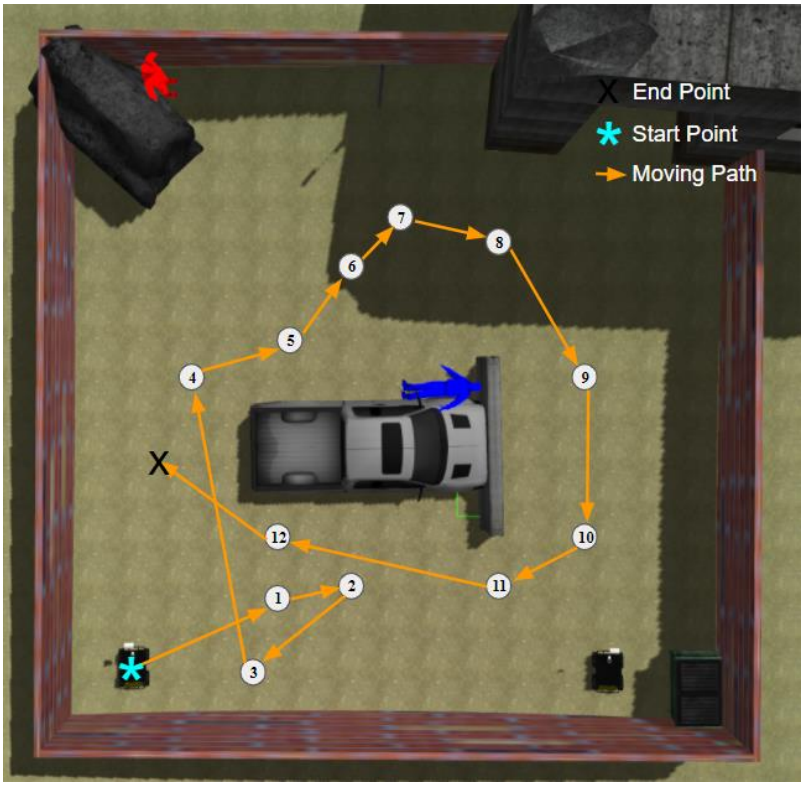


Figure 34. Exploration path in 15x15 m^2 environment

One unexpected result was found from the experiments. The two-robots fleet was supposed to find more victims than a single robot. However, as shown in Figure 35, the two-robot fleet had better performance in finding the victim only in the 20x20 m^2 environment. The possible reason is that the victims were placed with random orientation and posture. The robot needs to move closer than 5 m to identify the victim while the mapping range is further than the identification distance, which caused failure in identifying the victims. A conclusion can also be drawn from Figure 35 that the combination of greedy frontier exploration and victim identification through thermal camera did not guarantee to find all victims in the environment.

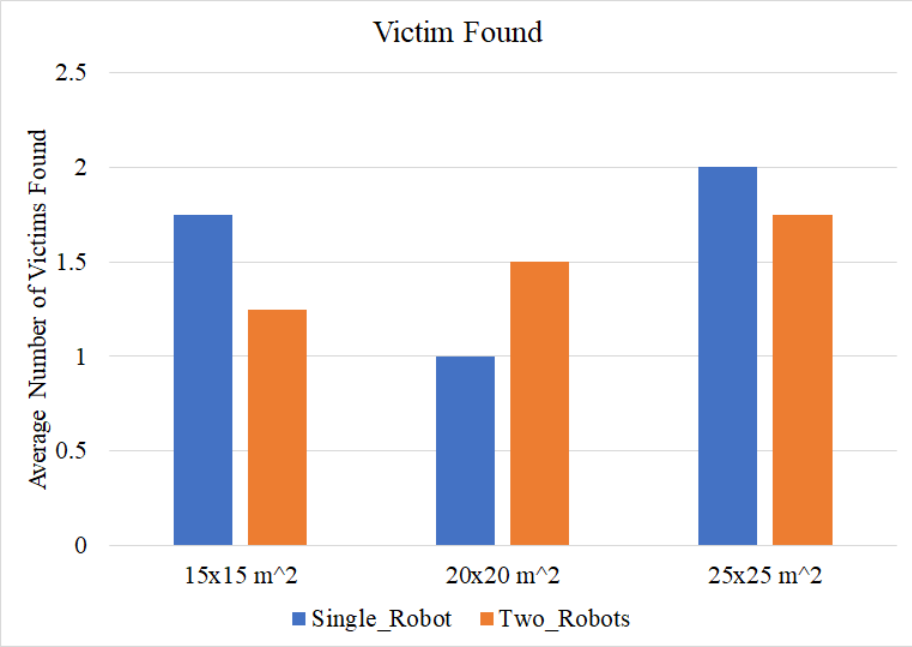


Figure 35. Average victims found in different environments

7. Conclusion and Future Work

In conclusion, the 3D simulator design can simulate frontier exploration and victim identification with multi-robots in the real world. A realistic 3D environment can be easily built based on the user's requirements as customized models can be imported from external software to the Gazebo simulator. RTAB-Map grants the robot the ability to map flat, cluttered, and uneven terrains and localize the robot accurately on the map. At the same time, the *hector_gazebo_thermal_camera* plugin enables the user to create distinguishable victim models with heat signatures. Regarding the map merging, a correctly merged map can be produced based on the initial robot positions.

The design was tested with the greedy frontier exploration from the *explore_lite* package. The simulated result met the expectation that the frontier exploration efficiency improves with a larger robot team size [1].

However, there still exist some limitations to the design. The computer used for experiments can only support two-robot simulations. If a more powerful computer is available, more tests can be run for frontier exploration with a larger fleet size to further validate the design. Also, a more robust map merging algorithm can be designed to consider the transformation between two maps so that the robots can start exploration with arbitrary orientation. Last, machine learning can be introduced to simulate victim identification close to reality.

Overall, the design meets the requirements for future researcher to test their coordination algorithm and has great potential for future improvement.

Reference

[1] Liu, Y., Nejat, G. *Robotic Urban Search and Rescue: A Survey from the Control Perspective*. J Intell Robot Syst 72, 147–165 (2013).

[2] Casper, Jennifer & Murphy, Robin. (2003). *Human-robot interactions during the robot-assisted urban search and rescue response at the World Trade Center*. IEEE transactions on systems, man, and cybernetics. Part B, Cybernetics : a publication of the IEEE Systems, Man, and Cybernetics Society. 33. 367-85. 10.1109/TSMCB.2003.811794.

[3] Laval, Jannik & Fabresse, Luc & Bouraqadi, Noury. (2013). *A methodology for testing mobile autonomous robots.* Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems. IEEE/RSJ International Conference on Intelligent Robots and Systems. 1842-1847. 10.1109/IROS.2013.6696599.

[4] Topiwala, A., Inani, P., & Kathpal, A. (2018). *Frontier Based Exploration for Autonomous Robot.* ArXiv, abs/1806.03581.

[5] B. Yamauchi, *A frontier-based approach for autonomous exploration*, Proceedings 1997 IEEE International Symposium on Computational Intelligence in Robotics and Automation CIRA'97. 'Towards New Computational Principles for Robotics and Automation', Monterey, CA, USA, 1997, pp. 146-151, doi: 10.1109/CIRA.1997.613851.

[6] Yan, Z.; Fabresse, L.; Laval, J.; Bouraqadi, N. *Building a ROS-Based Testbed for Realistic Multi-Robot Simulation: Taking the Exploration as an Example*. Robotics 2017, 6, 21.

[7] R. Mendonça, P. Santana, F. Marques, A. Lourenço, J. Silva and J. Barata, Kelpie: *A ROS-Based Multi-robot Simulator for Water Surface and Aerial Vehicles*, 2013 IEEE International Conference on Systems, Man, and Cybernetics, Manchester, 2013, pp. 3645-3650, doi: 10.1109/SMC.2013.621.

[8] M. M. M. Manhães, S. A. Scherer, M. Voss, L. R. Douat and T. Rauschenbach, *UUV Simulator: A Gazebo-based package for underwater intervention and multi-robot simulation*, OCEANS 2016 MTS/IEEE Monterey, Monterey, CA, 2016, pp. 1-8, doi: 10.1109/OCEANS.2016.7761080.

[9] Ros.org. 2021. ROS.org | *Is ROS For Me?*. [online] Available at: <https://www.ros.org/is-ros-for-me> [Accessed 29 January 2021].

[10] F. Niroui, B. Sprenger, and G. Nejat, *Robot exploration in unknown*

cluttered environments when dealing with uncertainty, in Proc. IEEE

Int. Symp. Robot. Intell. Sensors, Ottawa, Canada, 2017

[11] Wiki.ros.org. 2021. *navigation/Tutorials/RobotSetup - ROS Wiki*. [online] Available at: <http://wiki.ros.org/navigation/Tutorials/RobotSetup#Robot_Setup> [Accessed 29 January 2021].

[12] Zhu, Z., Yang, S., Dai, H., & Li, F. (2018). *Loop Detection and Correction of 3D Laser-Based SLAM with Visual Information*. CASA 2018.

[13] M. Labbé and F. Michaud, *RTAB-Map as an Open-Source Lidar and Visual SLAM Library for Large-Scale and Long-Term Online Operation*, in Journal of Field Robotics, vol. 36, no. 2, pp. 416–446, 2019.

[14] M. Korkmaz and A. Durdu, *Comparison of optimal path planning algorithms*, 2018 14th International Conference on Advanced Trends in Radioelecrtronics, Telecommunications and Computer Engineering (TCSET), Slavske, 2018, pp. 255-258, doi: 10.1109/TCSET.2018.8336197.

[15] Gerkey, Brian P.. *Planning and Control in Unstructured Terrain*. (2008).

[16] M. Santos Pessoa de Melo, J. Gomes da Silva Neto, P. Jorge Lima da Silva, J. M. X. Natario Teixeira and V. Teichrieb, *Analysis and Comparison of Robotics 3D Simulators, 2019 21st Symposium on Virtual and Augmented Reality (SVR)*, Rio de Janeiro, Brazil, 2019, pp. 242-251, doi: 10.1109/SVR.2019.00049.

[17] E. Kučera, O. Haffner and R. Leskovský, *Multimedia Application for Object-oriented Programming Education Developed by Unity Engine*, 2020 Cybernetics & Informatics (K&I), Velke Karlovice, Czech Republic, 2020, pp. 1-8, doi: 10.1109/KI48306.2020.9039853.

[18] Gazebosim.org. 2021. *Gazebo*. [online] Available at: <http://gazebosim.org/> [Accessed 29 January 2021].

[19] A. Ayala, F. Cruz, D. Campos, R. Rubio, B. Fernandes and R. Dazeley, *A Comparison of Humanoid Robot Simulators: A Quantitative Approach*, 2020 Joint IEEE 10th International Conference on Development and Learning and Epigenetic Robotics (ICDL-EpiRob), Valparaiso, 2020, pp. 1-6, doi: 10.1109/ICDL-EpiRob48136.2020.9278116.

[20] O. Michel, Cyberbotics Ltd. *Webots: Professional mobile robot simulation*, International Journal of Advanced Robotic Systems, vol. 1, no. 1, pp. 39-42, 2004.

[21] Gazebosim.org. 2021. *Gazebo : Tutorial : Gazebo plugins in ROS*. [online] Available at: <http://gazebosim.org/tutorials?tut=ros_gzplugins> [Accessed 29 January 2021].

[22] Wiki.ros.org. 2021. *ROS/Patterns/Communication - ROS Wiki*. [online] Available at: <http://wiki.ros.org/ROS/Patterns/Communication> [Accessed 29 January 2021].

[23] Yousif, K., Bab-Hadiashar, A. & Hoseinnezhad, R. *An Overview to Visual Odometry and Visual SLAM: Applications to Mobile Robotics*. Intell Ind Syst 1, 289–311 (2015). https://doi.org/10.1007/s40903-015-0032-7

[24] Zhu, Z., Yang, S., Dai, H., & Li, F. (2018). *Loop Detection and Correction of 3D Laser-Based SLAM with Visual Information*. CASA 2018.

[25] Raúl Mur-Artal and Juan D. Tardós. *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.

[26] Taihú Pire, Thomas Fischer, Gastón Castro, Pablo De Cristóforis, Javier Civera, Julio Jacobo Berlles, *S-PTAM: Stereo Parallel Tracking and Mapping, Robotics and Autonomous Systems*, Volume 93, 2017, Pages 27-42, ISSN 0921-8890,

[27] Kerl, C., Sturm, J., and Cremers, D. (2013). *Dense visual SLAM for RGB-D cameras*. In Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems, pages 2100–2106.

[28] Gutiérrez-Gómez, D., & Guerrero, J.J. (2018*). RGBiD-SLAM for Accurate Real-time Localisation and 3D Mapping*. ArXiv, abs/1807.08271.

[29] Scaramuzza, D. and Fraundorfer, F. (2011). *Visual odometry [tutorial]*. IEEE Robotics & Automation Magazine, 18(4):80–92.

[30] Shi, J. et al. (1994). *Good features to track.* In Proceedings IEEE Computer Society Conference on Computer Vision and Pattern Recognition, pages 593–600.

[31] Lucas, B. D. and Kanade, T. (1981). *An iterative image registration technique with an application to stereo vision*. In Proceedings International Joint Conference on Artificial Intelligence, pages 674–679. Vancouver, BC, Canada.

[32] Muja, M. and Lowe, D. G. (2009*). Fast approximate nearest neighbors with automatic algorithm configuration.* In Proceedings International Conference on Computer Vision Theory and Application, pages 331–340.

[33] Lowe, D. G. (2004). *Distinctive image features from scale-invariant keypoints. International Journal of Computer Vision*, 60(2):91–110.

[34] Calonder, M., Lepetit, V., Strecha, C., and Fua, P. (2010). *Brief: Binary robust independent elementary features*. In European Conference on Computer Vision, pages 778–792. Springer.

[35] Bradski, G. and Kaehler, A. (2008). *Learning OpenCV: Computer vision with the OpenCV library*. O'Reilly Media, Inc.

[36] Labb'e, M. and Michaud, F. (2013). *Appearance-based loop closure detection for online large-scale and long-term operation.* IEEE Transactions on Robotics, 29(3):734–745.

[37] Geiger, A., Lenz, P., and Urtasun, R. (2012). *Are we ready for autonomous driving? The KITTI vision benchmark suite*. In Proceedings IEEE Conference on Computer Vision and Pattern Recognition, pages 3354–3361.

[38] Sturm, J., Engelhard, N., Endres, F., Burgard, W., and Cremers, D. (2012). *A benchmark for the evaluation of RGB-D SLAM systems*. In Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems, pages 573–580.

[39] Burri, M., Nikolic, J., Gohl, P., Schneider, T., Rehder, J., Omari, S., Achtelik, M. W., and Siegwart, R. (2016). *The EuRoC micro aerial vehicle datasets.* The International Journal of Robotics Research, 35(10):1157–1163.

[40] Fallon, M., Johannsson, H., Kaess, M., and Leonard, J. J. (2013). *The MIT stata center dataset.* The International Journal of Robotics Research, 32(14):1695–1699.

[41] Klein G., Murray D.*Parallel tracking and mapping for small AR workspaces*

Proceedings of the IEEE International Symposium on Mixed and Augmented Reality, ISMAR, 978-1-4244-1749-0, IEEE Computer Society, Washington, DC, USA (2007), pp. 1-10, 10.1109/ISMAR.2007.4538852

[42] Klein G., Murray D.*Parallel tracking and mapping for small AR workspaces*

Proceedings of the IEEE International Symposium on Mixed and Augmented Reality, ISMAR, 978-1-4244-1749-0, IEEE Computer Society, Washington, DC, USA (2007), pp. 1-10, 10.1109/ISMAR.2007.4538852

[43] G'alvez-L'opez, D. and Tard'os, J. D. (2012). *Bags of binary words for fast place recognition in image sequences.* IEEE Transactions on Robotics, 28(5):1188–1197.

[44] Daniel Guti'errez-G'omez, Walterio MayolCuevas, and J. J. Guerrero. *Dense rgb-d visual odometry using inverse depth.* Robotics and Autonomous Systems (RAS), 75(Part B):571 – 583, 2016. Special Section on 3D Perception with PCL.

[45] Dwijotomo, A., Rahman, M.A., Ariff, M.H., Zamzuri, H., & Azree, W.M. (2020). *Cartographer SLAM Method for Optimization with an Adaptive Multi-Distance Scan Scheduler*. Applied Sciences, 10, 347.

[46] R. Dubé, D. Dugas, E. Stumm, J. Nieto, R. Siegwart and C. Cadena, *SegMatch: Segment based place recognition in 3D point clouds*, 2017 IEEE International Conference on Robotics and Automation (ICRA), Singapore, 2017, pp. 5266-5272, doi: 10.1109/ICRA.2017.7989618.

[47] Konolige, K.; Grisetti, G.; Kümmerle, R.; Burgard, W.; Limketkai, B.; Vincent, R. *Sparse Pose Adjustment for 2D mapping.* In Proceedings of the IROS, Taipei, Taiwan, 18–22 October 2010

[48] Vincent, R., Limketkai, B., and Eriksen, M. (2010). *Comparison of indoor robot localization techniques in the absence of GPS*. In Detection and Sensing of Mines, Explosive Objects, and Obscured Targets XV, volume 7664, page 1Z. International Society for Optics and Photonics.

[49] Carlone, L., Aragues, R., Castellanos, J. A., and Bona, B. (2012). *A linear approximation for graph-based simultaneous localization and mapping.* Robotics: Science and Systems VII, pages 41–48.

[50] Grisetti, G., Stachniss, C., and Burgard, W. (2007*). Improved techniques for grid mapping with RaoBlackwellized particle filters*. IEEE Transactions on Robotics, 23(1):34–46.

[51] Kohlbrecher, S., Meyer, J., von Stryk, O., and Klingauf, U. (2011). *A flexible and scalable SLAM system with full 3D motion estimation*. In Proceedings IEEE International Symposium on Safety, Security and Rescue Robotics.

[52] Douillard, J. Underwood, N. Kuntz, V. Vlaskine, A. Quadros, P. Morton, and A. Frenkel, *On the segmentation of 3d lidar point clouds*, in IEEE Int. Conf. on Robotics and Automation, 2011.

[53] M. A. Fischler and R. C. Bolles, *Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography*, Communications of the ACM, vol. 24, no. 6, pp. 381–395, 1981.

[54] Noreen, I., Khan, A., & Habib, Z. (2016). *Optimal Path Planning using RRT* based Approaches: A Survey and Future Directions*. International Journal of Advanced Computer Science and Applications, 7.

[55] J. Yao, C. Lin, X. Xie, A. J. Wang and C. Hung, *Path Planning for Virtual Human Motion Using Improved A* Star Algorithm*, 2010 Seventh International Conference on Information Technology: New Generations, Las Vegas, NV, 2010, pp. 1154-1158, doi: 10.1109/ITNG.2010.53.

[56] C. Saranya, K. Koteswara Rao, Manju Unnikrishnan, Dr. V. Brinda, V.R. Lalithambika, M.V. Dhekane, *Real Time Evaluation of Grid Based Path Planning Algorithms: A comparative study*, IFAC Proceedings Volumes, Volume 47, Issue 1, 2014, Pages 766-772, ISSN 1474-6670, ISBN 9783902823601.

[57] Ahmadzadeh, S., & Ghanavati, M. (2012). *Navigation of Mobile Robot Using the PSO Particle Swarm Optimization.* Journal of Academic and Applied Studies, 2.

[58] A. Z. Nasrollahy and H. H. S. Javadi, *Using Particle Swarm Optimization for Robot Path Planning in Dynamic Environments with Moving Obstacles and Target*, 2009 Third UKSim European Symposium on Computer Modeling and Simulation, Athens, 2009, pp. 60-65, doi: 10.1109/EMS.2009.67.

[59] smail A, Sheta A, Al-Weshah M (2008) *A mobile robot path planning using genetic algorithm in static environment*. J Comput Sci 4(4):341–344

[60] K. H. Sedighi, K. Ashenayi, T. W. Manikas, R. L. Wainwright and Heng-Ming Tai, *Autonomous local path planning for a mobile robot using a genetic algorithm*, Proceedings of the 2004 Congress on Evolutionary Computation (IEEE Cat. No.04TH8753), Portland, OR, USA, 2004, pp. 1338-1345 Vol.2, doi: 10.1109/CEC.2004.1331052.

[61] J. J. Kuffner and S. M. LaValle, *RRT-connect: An efficient approach to single-query path planning*, Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on

Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065), San Francisco, CA, USA, 2000, pp. 995-1001 vol.2, doi: 10.1109/ROBOT.2000.844730.

[62] Karaman, S., and E. Frazzoli. *Sampling-based algorithms for optimal motion planning*. The International Journal of Robotics Research 30, no. 7 (June 22, 2011): 846-894.

[63] LaValle SM, Kuffner JJ. *Rapidly-exploring random trees: progress and prospects.* Algorithmic and Computational Robotics:New Directions 2001; 5:293–308.

[64] Wiki.ros.org. 2021. *global_planner - ROS Wiki.* [online] Available at: <http://wiki.ros.org/global_planner> [Accessed 29 January 2021].

[65] D. Fox, W. Burgard and S. Thrun, *The dynamic window approach to collision avoidance*, in IEEE Robotics & Automation Magazine, vol. 4, no. 1, pp. 23-33, March 1997, doi: 10.1109/100.580977.

[66] C. Roesmann, W. Feiten, T. Woesch, F. Hoffmann and T. Bertram, *Trajectory modification considering dynamic constraints of autonomous robots,* ROBOTIK 2012; 7th German Conference on Robotics, Munich, Germany, 2012, pp. 1-6.

[67] S. Quinlan and O. Khatib, *Elastic bands: connecting path planning and control,* [1993] Proceedings IEEE International Conference on Robotics and Automation, Atlanta, GA, USA, 1993, pp. 802-807 vol.2, doi: 10.1109/ROBOT.1993.291936.

[68] Marín-Plaza, P., Hussein, A., Martín, D., & Escalera, A.D. (2018). *Global and Local Path Planning Study in a ROS-Based Research Platform for Autonomous Vehicles*. Journal of Advanced Transportation, 2018, 1-10.

.[69] Johnson, J. J., Li, L., Liu, F., Qureshi, A. H., and Yip, M. C., *Dynamically Constrained Motion Planning Networks for Non-Holonomic Robots*, arXiv e-prints, 2020.

[70] Bäckström, C., & Christoffersson, N. (2006). *Urban Search and Rescue - An evaluation of technical search equipment and methods*.

[71] R. Hahn, D. Lang, M. Häselich and D. Paulus, *Heat mapping for improved victim detection*, 2011 IEEE International Symposium on Safety, Security, and Rescue Robotics, Kyoto, 2011, pp. 116-121, doi: 10.1109/SSRR.2011.6106769.

[72] Navneet Dalal and Bill Triggs, *Histograms of Oriented Gradients for Human Detection*, International Conference on Computer Vision & Pattern Recognition (CVPR '05) 1 (2005) 886—893.

[73] Riaz, I.; Piao, J.; Shin, H. *Human detection by using centrist features for thermal images*. In Proceedings of the IADIS International Conference Computer Graphics, Visualization, Computer Vision and Image Processing, Prague, Czech Republic, 22–24 July 2013.

[74] • Lin T-Y, Dollar P, Girshick R, et al. *Feature pyramid networks for object detection.* In: 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). Honolulu: IEEE; 2017. p. 936–44.

[75] Liu W, Anguelov D, Erhan D, Szegedy C, Reed S, Fu CY, et al. *SSD: single shot MultiBox detector*. In: Leibe B, Matas J, Sebe N, Welling M, editors. Computer Vision – ECCV 2016. Cham: Springer International Publishing; 2016. p. 21–37.

[76] Redmon J, Farhadi A (2016) *YOLO9000: better, faster, stronger*. arXiv:161208242 [cs].

[77] Redmon J, Farhadi A (2018) *YOLOv3: an incremental improvement*. CoRR abs/1804.02767:

[78] Lin T-Y, Goyal P, Girshick RB, et al. *Focal loss for dense object detection*. In: 2017 IEEE International Conference on Computer Vision (ICCV); 2017. p. 2999–3007.

[79] Navneet Dalal and Bill Triggs, *Histograms of Oriented Gradients for Human Detection*, International Conference on Computer Vision & Pattern Recognition (CVPR '05) 1 (2005) 886—893.

[80] Wu, J. et al, 2011. *Real-Time Human Detection Using Contour Cues*. In Proc. ICRA, Shanghai, China, pp. 860-867.

[81] Fung, A., Wang, L.Y., Zhang, K. et al. *Using Deep Learning to Find Victims in Unknown Cluttered Urban Search and Rescue Environments*. Curr Robot Rep 1, 105–115 (2020).

[82] K. He, X. Zhang, S. Ren, and J. Sun. *Deep residual learning*

for image recognition. In CVPR, 2016.

[83]Hoiem, D., Chodpathumwan, Y., Dai, Q.: *Diagnosing error in object detectors*. In: ECCV 2012. (2012)

[84] Gazebosim.org. 2021. *Gazebo : Tutorial : Digital Elevation Models*. [online] Available at: <http://gazebosim.org/tutorials/?tut=dem> [Accessed 29 January 2021].

[85] Gazebosim.org. 2021. *Gazebo : Tutorial : Import Meshes*. [online] Available at: <http://gazebosim.org/tutorials/?tut=import_mesh> [Accessed 29 January 2021].

[86] Makehumancommunity.org. 2021. [online] Available at: <http://www.makehumancommunity.org/> [Accessed 29 January 2021].

[87] Wiki.ros.org. 2021. *Robots/Husky - ROS Wiki*. [online] Available at: <http://wiki.ros.org/Robots/Husky> [Accessed 29 January 2021].

[88] Wiki.ros.org. 2021. *move_base - ROS Wiki*. [online] Available at: <http://wiki.ros.org/move_base> [Accessed 29 January 2021].

[89] Wiki.ros.org. 2021. *explore_lite - ROS Wiki*. [online] Available at: <http://wiki.ros.org/explore_lite> [Accessed 29 January 2021].

[90] Wiki.ros.org. 2021. *hector_gazebo_thermal_camera - ROS Wiki*. [online] Available at: <http://wiki.ros.org/hector_gazebo_thermal_camera> [Accessed 29 January 2021].

[91] C. Bagavathi, O. Saraniya, Chapter 13 - *Evolutionary Mapping Techniques for Systolic Computing System*, Editor(s): Arun Kumar Sangaiah, Deep Learning and Parallel Computing Environment for Bioengineering Systems, Academic Press, 2019, Pages 207-223, ISBN 9780128167182.

[92] Wiki.ros.org. 2021. *Names - ROS Wiki*. [online] Available at: <http://wiki.ros.org/Names> [Accessed 13 April 2021].