IMPLEMENTATION OF MULTI ROBOT SIMULTANEOUS LOCALIZATION AND MAPPING

by

Jaydeep Kshirsagar

A thesis submitted to the faculty of
The University of North Carolina at Charlotte
in partial fulfillment of the requirements
for the degree of Master of Science in
Electrical Engineering

Charlotte

2018

Approved by:

____

Dr. James Conrad

____

Dr. Andrew Willis

____

Dr. Ronald Sass

ABSTRACT

JAYDEEP KSHIRSAGAR. Implementation of Multi Robot Simultaneous Localization and Mapping. (Under the direction of DR. JAMES CONRAD)

For any robot to effectively traverse its environment, it requires a map and its pose (location along with the orientation) within that map. Often, a map will not be available if the robot is presented with an unfamiliar environment. This scenario requires a robot to construct the map and localize itself in it to navigate. The problem of concurrently building the map and localizing a robot in that environment is defined as Simultaneous Localization and Mapping (SLAM). Many SLAM applications already exist for a single robot such as navigating the unmanned mines and exploring the sites with natural calamities. With the advent of swarming robots that must interact with each other, extensive research is being conducted for the extension of the SLAM problem to multiple robots known as Multi-Robot SLAM (MR-SLAM). In the MR-SLAM environment, the efficiency is improved, and time constraints are reduced, but its implementation is restricted due to constraints such as communication bandwidth, memory requirements and problems faced during map merging and coordinate transformation.

This research introduces a way to simulate a multi-robot environment in the Gazebo simulator by creating a launch file with attributes and specifications for many robots. A generalized MATLAB script is written in order to populate a Gazebo world with user defined number of TurtleBots along with their communication topics. This system is further tested with an implementation of *Sparse Extended Information Filter* (SEIF) with known correspondences for such multi-robot environment.

ACKNOWLEDGEMENTS

I wish to present my sincere thanks to my advisor, Dr. James M. Conrad for his constant encouragement during this journey. This work would not have been possible without his support and belief in me. I am grateful to Dr. Andrew Willis and Dr. Ronald Sass for accepting my request to be on my thesis committee. Each of the members of my thesis committee has provided me with an extensive personal and professional guidance. A special thank you to Dr. Willis for the technical guidance and making available all the resources.

I would like to thank all those whose assistance proved to be a milestone in the accomplishment of my end goal. I am especially indebted to Dr. Sam Shue, whose advice for my research proved to be a landmark effort towards the success of my thesis. I am grateful to my friends Aishwarya Panchpor and Mukul Gosavi. It has been a pleasure working with these researchers.

Finally, I would like to appreciate the efforts and support provided by my parents and sisters. Their love, encouragement, and inspiration always aspire me to go an extra mile.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

EKF	Extended Kalman Filter
GUI	Graphical User Interface
JCBB	Joint Compatibility Branch and Bound
LIDAR	Light Detection and Ranging
MR-SLAM	Multi-Robot Simultaneous Localization and Mapping
NN	Nearest Neighbor
RBPF	Rao-Blackwellized Particle Filter
ROS	Robot Operating System
SEIF	Sparse Extended Information Filter
SLAM	Simultaneous Localization and Mapping
SR-SLAM	Single Robot Simultaneous Localization and Mapping
URDF	Universal Robotic Description Format

CHAPTER 1: INTRODUCTION

Many applications in industries like health care, military, and consumer electronics are now using robots. The tremendous growth in automation can be witnessed by products such as vacuum cleaners which will clean the floor by itself without any human participation. Such applications of robotics can be extended to situations like:

- natural or civil disasters
- industrial accidents
- crime scenes where there is a need to respond quickly and effectively

To justify the fact that human tasks are more efficiently and accurately performed by robots, robots have to function more like humans. Similarly, for a robot to traverse in any environment, it should be well acquainted with the environment's features and its own position with respect to those features. The process of collecting information about environment features and building a virtual environment for local reference is commonly referred to as 'mapping'. Similarly, localization can be defined as a process through which a robot determines its position in a given environment. To build an error-free map with essential information about the environment, it is important for a robot to have knowledge about its precise location in the environment and at the same time to determine a precise location of a robot, an accurate map is required. To address this problem, an innovative approach called *Simultaneous Localization and Mapping* (SLAM) is introduced, where a robot maps an environment concurrently with the localization.

1.1 Motivation

Extensive research is being conducted in the field of robotics [6]. Robots performing day-to-day tasks are not surprising to a common man. For a robot to perform these actions, it has to understand its surroundings and react according to it. A lot of work has been done on a robot standing in an environment. Questions like "How will it perceive the world?" and "How will it find its way to the destination?" have been explored to a great depth and engineers have succeeded to create many state-of-the-art solutions for the problem above.

To explore a large area, multiple robots will take much less time than a single robot. For example, exploration and mapping of a full house require a single robot to traverse through the entire house. The total time required to build a map of the house can be reduced by deploying multiple robots in different areas of the house and combining their respective information to map the environment.

Even though state-of-the-art-solutions are available for *Single robot Simultaneous Localization and Mapping* (SR-SLAM), its immediate extension for an implementation of *Multi robot Simultaneous Localization and Mapping* (MR-SLAM) is restricted due to factors such as dependency on wireless communication, limitations on memory requirements and sensor networks.

Many simulators are available to imitate the operation of the real-world system. Although Gazebo is the most popular amongst them, it becomes complicated to spawn multiple robots in a Gazebo environment. The default Gazebo configurations will spawn a default robot along with its communication attributes. Even for the multiple numbers of robot, name and communication attributes for those robots will be common which makes it complicated to control each robot individually. It has been a challenging problem for many researchers to develop a simulation system with multiple robots to test the application. This motivated the idea of having a tool which will help Gazebo to support multi-robot simulations.

1.2 Simultaneous Localization And Mapping

SLAM can be defined as a process of developing a map of an environment and simultaneously localizing a robot in it. SLAM consists of four major tasks:

- Perception
- Localization
- Cognition
- Motion Control

Perception can be viewed as the robot's interpretation of its environment, where a robot tries to collect and process information from the outside world through the use of sensors such as cameras and range-finders. Localization is an estimation of a robot's pose using the data collected from sensors and a previously constructed map. Data collected from the sensors is used by a robot to estimate the current position during motion as well as to generate a map of the environment. Cognition is a process of analyzing the environment and deciding the response of the robot. Motion control can be defined as a process of traversing in an already mapped or unmapped environment.

SLAM can be implemented for several different topologies such as SLAM for a system having a single robot in a static environment, a system with a robot in a dynamic environment and a system with multiple robots in an environment. Extensive research has already been done on SR-SLAM. Many techniques, including the use of recursive estimators such as *Extended Kalman Filter* (EKF), *Rio-Blackwellized particle filter*(RBPF), and *Sparse Extended Information Filter* (SEIF), have been developed to solve the problem of localization in SR-SLAM. Research regarding mapping techniques has brought into focus several types of maps such as topological maps, metric maps, and hybrid maps.

The SLAM problem can be categorized into two main parts: Online SLAM and Full SLAM. In Online SLAM, the posterior (probability of next state, i.e location and orientation of a robot) is only estimated over the current pose of the robot, whereas in Full SLAM the posterior is calculated over the entire path, i.e. all the previous readings are considered for the estimation of the posterior.

The posterior of a robot and map is described by:

$p(x_{1:t}\ , m \mid z_{1:t}\ ,\ u_{1:t-1})$

Where,

$x_{1:t}$: pose of a robot from time $T = 0$ to $T = t$

m : map (a vector containing landmark positions)

$z_{1:t}$: Observations from time $T = 1$ to $T = t$

$u_{1:t-1}$: controls from time $T = 1$ to $T = t - 1$

For Full SLAM:

$$p(x_t, m|z_{1:t}\ , u_{1:t-1}) = \int \cdots \int (x_{1:t}, m|z_{1:t}, u_{1:t-1})dx_1 dx_2 ... dx_t \tag{1.1}$$

For Online SLAM, the posterior is obtained by integrating out previous robot poses of a Full SLAM algorithm and then summing it over all past correspondences.

For Online SLAM:

$$p(x_t, m, c_t|z_{1:t}\ , u_{1:t-1}) = \int \cdots \int \sum_{c1} \sum_{c2} \cdot \cdot \sum_{c3} (x_{1:t}, m|z_{1:t}, u_{1:t-1})dx_1 dx_2 ... dx_{t-1} \tag{1.2}$$

Where c_t is defined as the correspondence for the landmark.

Though very efficient techniques are developed for SR-SLAM, they cannot be immediately extended to the MR-SLAM. In MR-SLAM, observations of a robot are dependent on every other robots' pose, which makes the posterior of one robot dependent on the trajectory or pose of the other. Major challenges faced during implementation of MR-SLAM are posterior estimation from data gathered by different robots, limitations due to unreliable wireless sensing network, coordination between robots, an individual frame of reference to shared world representation, complexity, memory requirements, and dynamic environment [7, 8]. Knowledge of the initial position of the robot also plays a key role in underlying techniques. If the initial position of a robot is known then MR-SLAM is a simple extension from SR-SLAM [9, 10] but for an unknown initial position, consistent integration over a common map is difficult. For MR-SLAM with a known initial robot position, there must be a third agent monitoring the pose of all the robots or they should be initialized to a known position in a common map. In MR-SLAM with unknown initial positions, every robot will take its own sensor measurements and fuse them into a common map including the measurements by other robots.

1.3 Completed Research Work

Many state-of-the-art solutions can be found for SR-SLAM. The advantages of MR-SLAM over SR-SLAM was a motivating factor for the study of the MR-SLAM and complications in extending the methods used to implement SR-SLAM to MR-SLAM provided an opportunity to contribute to this area of study.

Using Gazebo for a multi-robot simulation was a challenging part for the research. This research introduced a tool which can be used to spawn multiple TurtleBots in the Gazebo world. This tool enabled researchers to use as many numbers of TurtleBot as an application requires with an individual interface to each TurtleBot.

This research implemented SEIF algorithm followed by the map merging routine. Map merging has an important contribution to this area of study in exploiting the advantages of the multi-robot systems. A decentralized system with two TurtleBots and six unique feature based landmarks was developed to test the TurtleBot spawning tool during the research.

1.4 Organization of Thesis

This thesis is divided into following Chapters: Literature Survey, Implementation, Results, Conclusion and Future Work, and Bibliography.

Chapter 2 is a literature survey for MR-SLAM. It discusses various current methods and state-of-the-art solutions. Chapter 3 introduces ROS along with its structure, Gazebo, TurtleBots and their contribution to the implementation. It includes a process to build a simulation environment along with the system block diagram. Chapter 4 presents the results of the implementation in Chapter 3. It further discusses results and factors affecting them. Chapter 5 concludes the research and discusses the future work followed by the bibliography.

CHAPTER 2: LITERATURE: MULTI ROBOT SLAM

Solutions to MR-SLAM can be classified as follows according to their method of controlling and processing distribution structure (Figure 2.1):

- Centralized System
- Decentralized System
- Distributed System

In a centralized system, all the robots are connected to a central server in which each robot will gather its local information and transfer the data to a central server. This central server is responsible for collecting and fusing all the data into a single global map and transfer back the fused data (map). This method imposes an immense amount of overhead due to a continuous high exchange of large amounts of data between server and robots. Its complexity is directly related to the number of robots connected to a server because it becomes impractical to connect many robots to a single server and it requires continuous monitoring of each deployed robot. One major drawback of this system includes functionality dependency on a single unit such as a central server because the failure in server functionality will cause the entire system to break down. In Decentralized systems data processing power is divided amongst predetermined few robots. Robots will continue to gather the data in the local frame and this data is shared with the robots having the processing power. This system is more robust than the centralized system. Distributed systems consist of many robots implementing its own SLAM and generating a local map with respect to the local frame. Whenever there is a meeting of two robots, data will be exchanged between them and this data will be used for mapping individual map.

This approach implements a more flexible and scalable algorithm for MR-SLAM. This paper showcases different techniques used to solve the problem of MR-SLAM and several types of maps that can be implemented along with it.

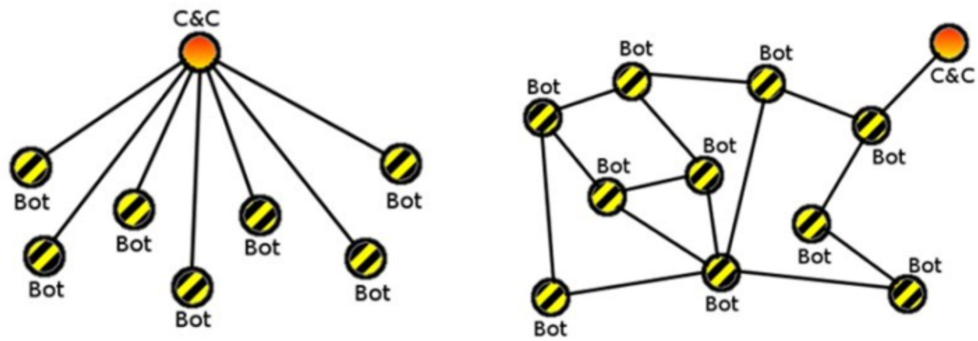


Figure 2.1: (a) Centralized System (b)Decentralized System

2.1 Problem Of Localization And SLAM

Localization in SLAM is the process by which the position of a robot in a known or unknown environment is computed. The main challenge faced in localization is the design of the transformation matrix. A robot perceives the world through its sensors, so environmental information is perceived with reference to the robot's perspective, or local frame, and must be transmitted to the maps reference frame, or the global frame. To resolve the differences between these two reference frames, and to convert a robot's local coordinate system to a global coordinate system, a transformation matrix is computed. Usually, transformation matrices consider the relative pose - delta distance (distance between two robots), theta (angle) and noise in the observations from the sensors. Sensors that are used as an interaction between environment and robot can be categorized into two types: exteroceptive sensors and proprioceptive sensors. Exteroceptive sensors like *Light Detection and Ranging* (LIDAR) and *SOund Navigation And Ranging* (SONAR) use acoustic or electromagnetic energy to sense and to interact with the environment. Whereas proprioceptive sensors do not rely on the environment for sensing and they use the system's internal sensors for measurement. The odometer can be categorized as an example of a proprioceptive sensor as it uses wheel encoders to estimate the current pose of a robot. Sensors are susceptible to noise and they induce some error in data over time. As this error affects the precision of localization, many filters are used to eliminate those errors. State-of-the-art solutions to MR-SLAM implements filters such as Kalman filter, Extended Kalman filter, and Particle filter to eliminate error and increase the precision of Localization.

Howard [11] developed an Online approach for MR-SLAM using Rao-Blackwellized particle filter with known initial positions as well as with unknown initial positions. This algorithm can be explained using two robots exploring the environment. Whenever there is an encounter between these two robots, their respective poses along with the data are fed to filter. The filter generates two virtual robots and these measurements are fed to them in reverse time order to build a common map. For MR-SLAM each particle will be a tuple of $< x^i_{t1}, x^i_{t2}, m^i_t, w^i_t >$, where x^i_{t1}, x^i_{t2} are Robot 1 and Robot 2 poses (location along with its orientation) respectively, m^i_t is map and w^i_t represent weight of that particle. At the starting point, each robot will implement individual SLAM with occupancy grid mapping and store robot pose measurements in a data structure during continue independent exploration. Now whenever Robot 1 and Robot 2 are in the line of sight of each other or able to detect each other, Robot 2 will establish a communication and transfer its data to Robot 1. Now Robot 1 will update the tuple and occupancy grid. For this algorithm to work, a relative pose of Robot 2 with respect to Robot 1 is considered to define the transformation matrix. This algorithm assumes that the pose of robots will not interfere with the measurements of each other and it is implemented in an algorithm to ignore those readings.

For MR-SLAM with unknown initial poses, the encounter will be detected before incorporating the data into the map. In this approach, only one robot with arbitrary pose will start mapping and wait for an encounter with each additional robot before taking the data. All the subsequent encounters are neglected. Figure 2.2 Shows a

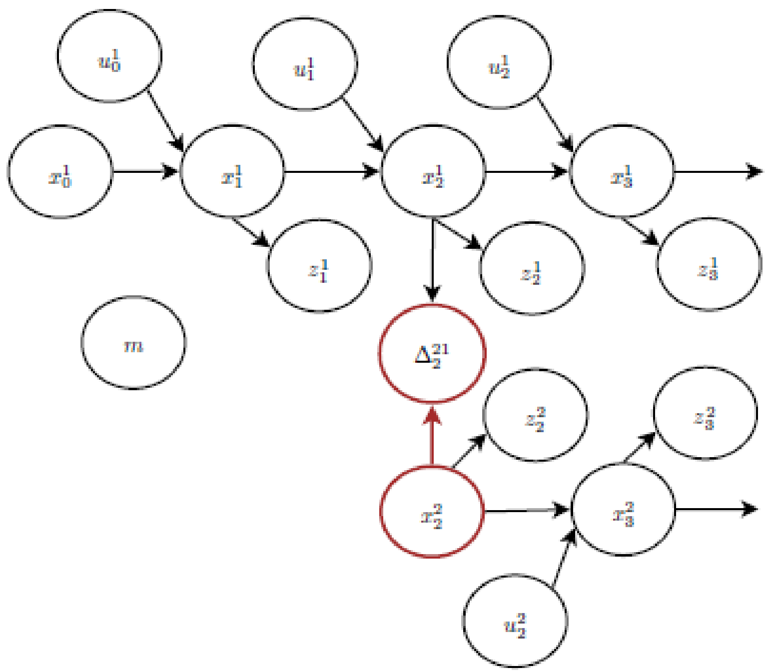


Figure 2.2: MR-SLAM for particle filter [1]

multi-robot SLAM with unknown initial poses using particle filter. At some point in time, t, Δ_{21} represents the relative pose of Robot 2 with respect to Robot 1 which is used to incorporate data into a map along with the coordinate transform. For a successful implementation of this filter, the author makes following approximations:

- Conditional dependencies between robot trajectories are ignored and treated as an independent variable.

- Uncertainty in the relative pose (Δ_{21}) is small.

All the data collected by the first robot will be considered and incorporated into the map, while data obtained only after the encounter is considered from other robots.

Another approach suggested by Thrun [1] to solve MR-SLAM is to implement *Sparse Extended Information Filter* (SEIF) with known initial correspondences. SEIF uses a Bayesian approach and instead of localizing robots in each other's map, this solution focuses on comparing local maps acquired by robots. SEIF is an intermediate algorithm between an Online SLAM and a full-SLAM algorithm. SEIF, just like online SLAM, implements the solution to determine the posterior over current pose and the map and at the same time SEIF takes advantages from full SLAM algorithm by keeping all the information. SEIF represents the SLAM posterior with the natural parameters of multivariate Gaussian in the form of an information matrix (Ω) and information vector (ϵ). An information matrix is a square matrix comprising pose of a robot and positions of the landmarks. It contains numerical links across the robot - landmark or landmark - landmark which gives the direct correlation between them. An information vector is an informative representation of the state vector.

SEIF is implemented in four steps: a motion update step, a measurement update step, a sparsification step, and a state estimation step. A measurement update step updates the elements in the information matrix and information vector corresponding to a robot and the respective landmarks by a non-zero number. These elements corresponding to robot and landmarks (non-diagonal elements) are denoted as links. This step in SEIF, links features to the current robot pose. A motion update step does not consider measurements from the environment features but it takes into account the command issued to the wheel motors (control signal). It updates the information matrix and information vector based on the control signal (u_t). A motion update algorithm takes previous information matrix and information vector as an input and incorporates motion control signal to update information vector and information matrix. Before robot motion, a link was established between features and poses. So, after a movement of the robot, these links are weakened as a robot motion introduces uncertainty in the information state. This uncertainty in the pose of a robot relative to the map results in the loss of information. This information is not entirely lost because of the fact that it lost information about pose relative to the map, but the relative information of features with respect to other features was retained in the map. This causes the filter to shift the information from pose-feature to feature-feature pair. This step updates direct links between feature pairs. Sebastian Thrun suggests that this transformation of a link is possible only if a feature has an active measurement link between itself and pose. This feature of SEIF helps to limit the computational complexity. So, by controlling the number of active features at any point of time, the computational complexity during the motion update and measurement update step can be controlled. This transformation of highly dense information matrix to sparsely populated information matrix by approximating the elements between passive features and robots state to near zero is also called as 'Sparsification'. One important advantage of this algorithm is the time required for sparsification is independent of a size of the map.

The last step in SEIF deals with the state estimation which can be estimated using equation 2.1:

$$\mu = \Omega_t^{-1} * \epsilon_t \tag{2.1}$$

Where,

Ω_t : information matrix

ϵ_t : information state

This step introduces high time complexity in the algorithm and so to avoid this SEIF exploits 'relaxation algorithm'. In relaxation algorithm, each step is updated based on the best estimates of nearby elements in the information graph. The accuracy of the results for SEIF and computational efficiency is decided by the degree of sparseness (number of active landmarks) in SEIF.

Extending this approach to MR-SLAM can be viewed as an implementation of

SR-SLAM in a local reference frame with respect to each robot and then exchanging of data between robots. Additivity and locality of update step make this algorithm amenable for MR-SLAM. In [1], Thrun explains the fusion of maps faces mainly two problems: each robot maintains its own coordinate system so transformation from one robot reference frame to another is nonlinear and along with it the data association problem must be solved while fusing the maps. Suppose we have two robots j and k each with information as $(\Omega_t^k, \epsilon_t^j)$ and $(\Omega_t^k, \epsilon_t^k)$ respectively. In order to fuse the maps, we require a transformation vector with linear displacement matrix 'd' rotational difference element 'A'. Using this transformation matrix, we can map the positions of robot j and its features to the k robot's coordinate system with a rotation of 'A' and a displacement by 'd'. Equations for map fusing can be written as:

$$\begin{aligned} \Omega^{j \to k} &= A * \Omega^j * A^T \\ \epsilon^{j \to k} &= A * (\epsilon^j - \Omega^{j \to k} d) \end{aligned} \tag{2.2}$$

Where,

$$A = \begin{bmatrix} cos\alpha & sin\alpha & 0 \\ -sin\alpha & cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$d = \begin{bmatrix} dx & dy & 0 \end{bmatrix}^T$$

Fused maps are now having a common coordinate system, which enabled Information state and information vector both to be added to find the joint information state. For a correspondence list with identical features in the joint map, common information can be computed by Equation 2.3 and Equation 2.4 as given below:

$$\begin{bmatrix} \Omega_{11} & \Omega_{12} & \Omega_{13} & \Omega_{14} \\ \omega 21 & \Omega_{22} & \Omega_{23} & \Omega_{24} \\ \Omega_{31} & \Omega_{32} & \Omega_{33} & \Omega_{34} \\ \Omega_{41} & \Omega_{42} & \Omega_{43} & \Omega_{44} \end{bmatrix} = \begin{bmatrix} \Omega_{12} & \Omega_{12} + \Omega_{14} & \Omega_{13} \\ \Omega_{21} + \Omega_{41} & \Omega_{22} + \Omega_{42} + \Omega_{24} + \Omega_{44} & \Omega_{23} + \Omega_{43} \\ \Omega_{31} & \Omega_{32} + \Omega_{34} & \Omega_{33} \end{bmatrix} \tag{2.3}$$

$$\begin{bmatrix} \epsilon_1 \\ \epsilon_2 \\ \epsilon_3 \\ \epsilon_4 \end{bmatrix} = \begin{bmatrix} \epsilon_1 \\ \epsilon_2 + \epsilon_4 \\ \epsilon_3 \end{bmatrix} \tag{2.4}$$

The results obtained using above algorithm are shown in Figure 2.3 and Figure 2.4 Thrun divided data collected from a single robot into eight local subsets (Figure 2.3) and ran map merging algorithm on them to produce a common map as shown in Figure 2.4.

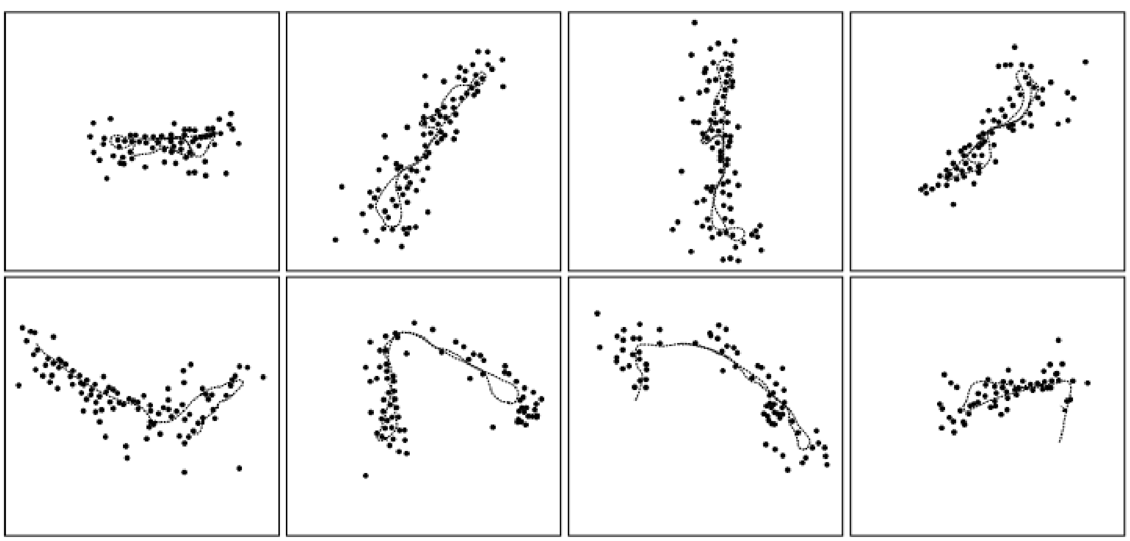


Figure 2.3: Eight local maps obtained by splitting map [1].

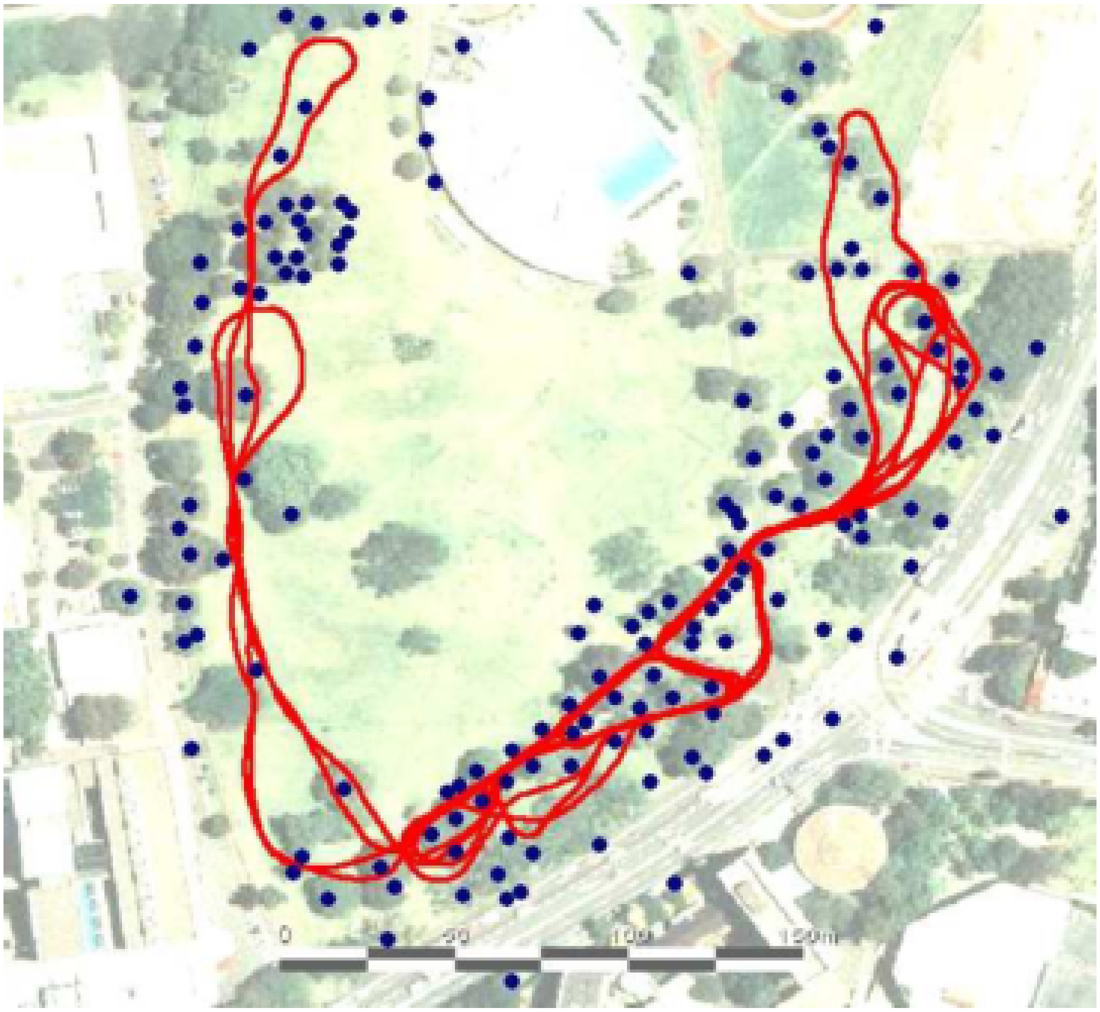


Figure 2.4: Above eight maps merged using SEIF [1].

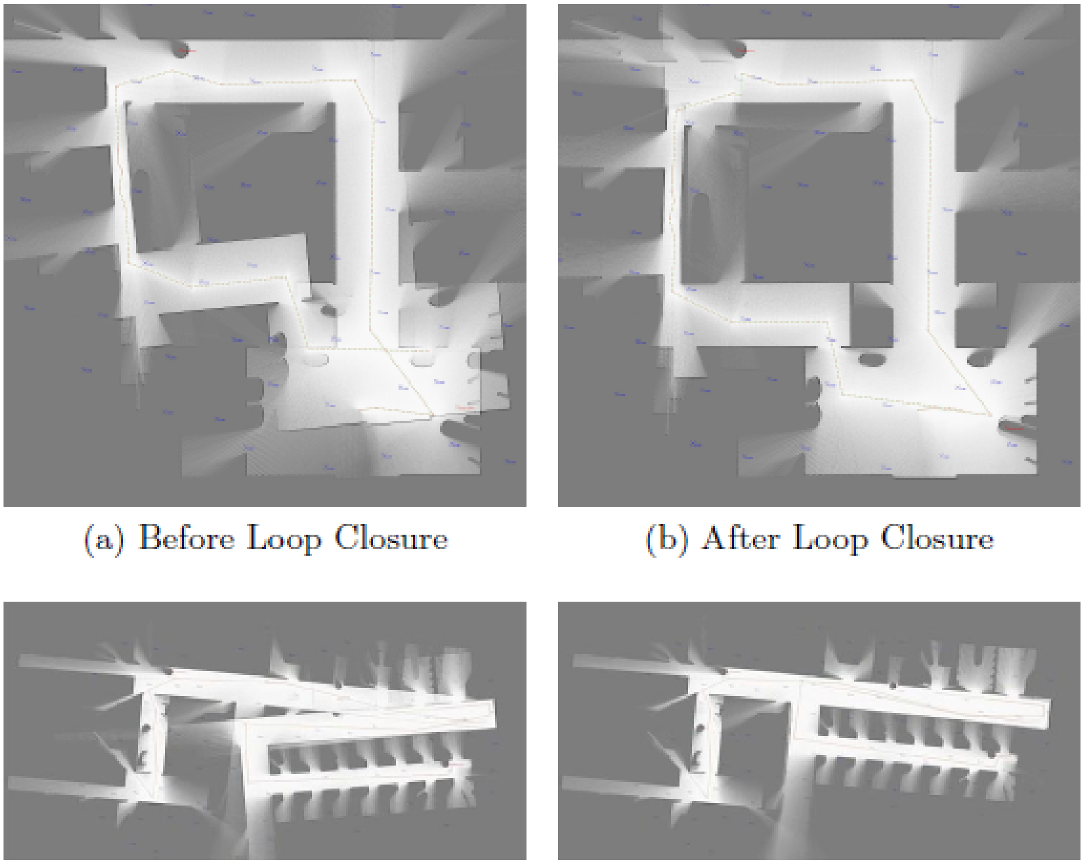
2.2 Problem Of Mapping in SLAM

SLAM is defined as a process of concurrently building a map of the environment from sensor measurement and using this map to obtain an estimate of the position of a robot. In SR-SLAM, a mapping is limited to a robot itself but in MR-SLAM the final aim of the system is to build a joint map. Thus, an extension of the techniques used for SR-SLAM to MR-SLAM is challenged by the factors: constraints on memory requirements, coordination between robots, data association problem - where robots have to decide if the current observation belongs to a previously observed landmark, or a different landmark of similar properties, the loop closure problem, and transformation of reference frames. In an environment with two robots with unknown initial locations, each robot will start exploring and use data from sensors like a laser scan, and Pan-Tilt camera [7],[8] to build a local map. Each robot maintains its own local map with a respective local reference frame. Once communication is established between two robots, each robot will transfer previous readings to fuse into a common map (global map).

Andrew Howard describes a similar approach in [11] with a Rao-Blackwellized particle filter. In any multi-robot environment, a starting point for a solution can be an implementation of single robot SLAM, unless and until there is an encounter with another robot. Initially, no robot will know the locations of any other robots, and so each robot will start individual SLAM. Once it encounters another robot it will localize other robot with relative to the self-position. Each robot will maintain a data structure, either a queue or a listing, to store the observations by laser and odometry sensors (sometimes odometry along with laser also known as 'Lodometry' is also used for precision). These reading can be divided into casual and acasual readings (before and after an encounter with another robot). This data is then fused into a common map avoiding the similar observations and incorporating the associated data that was recorded by the robot before an encounter. Each particle in the filter, used to develop a map and localize through filter, is a tuple $<$x, m, w$>$, where x is a robot pose at time t, m is a map generated at time t and w is the particle weight which eventually decides the validity of a respective particle. Particle having large weights are incorporated in the map and those with the low weights are discarded [11].

Different types of maps can be implemented while solving a problem of SLAM, amongst which topological maps (graph) are highly scalable due to their compact description and on the other hand grid-based maps have higher resolution but at the cost of a larger amount of memory [2]. Pfingsthorn et. al. put hybrid approach [2], where sensor data without any processing will be stored at nodes and edges will represent the transformation between these nodes. In MR-SLAM each robot will have an individual part of a map (either same or different), this method helps to merge different section of maps by correctly connecting the proper nodes. Two important concepts to be considered during merging maps are loop closing and overlap detection. 'Loop closing' is a process of connecting two separate regions of a map, which also helps to reduce the cumulative error. Seemingly two different observations in a map may belong to the same location in the environment. When displacement between two nodes is computed, inconsistency introduced by the this is resolved by incorporating the additional information regarding common landmark into the global map [2, 12, 13]. This additional information is in the form of correspondences, their relative distance and bearing in local maps. 'Map merging' is a process of connecting two different regions into the common or global map. It is very difficult to decide how two different maps from the two robots are connected to each other. For this reason, a common point of reference is decided, and map overlap is checked.

Every robot has its own local reference frame and therefore to merge maps in MR-SLAM, a transformation between their coordinate frames must be determined either by searching for common landmarks in two maps or by a robot to robot measurement when the robot meets [3]. Zhou et. al. propose another approach in which, when two robots are in communication range but with unknown relative locations, one robot will receive data from another robot and try to estimate its location by matching the received scan patch with respect to its own map. These robots then try to meet at a location on this assumed common map and if and only if they succeed, maps will be combined permanently, otherwise, maps will be discarded and exploration will continue. For efficient merging of two maps, a transformation between their respective frames must be determined. To determine the transformation vector, estimation of both robot poses, robot to robot distance and bearing measurements is essential. Both robots use Kalman Filters for individual mapping before rendezvous and a



(a) Before Loop Closure (b) After Loop Closure

(c) Overlaid maps (d) Aligned and refitted

Figure 2.5: Loop Closing and Loop Merging [2]

combined map will be computed from the above information. As these individual maps can contain common regions, map overlapping, and matching is also performed to reduce alignment errors and state vector matrix. There are two primary techniques used for the identification of ideal landmarks as *Nearest neighbor* (NN) [3] and *Joint Compatibility Branch and Bound* (JCBB) algorithms [14]. NN simply search for the closest two landmarks from two maps whereas JCBB find the largest number of many compatible pairing. NN is easier to implement whereas JCBB is robust but with high computational cost.

Figure 2.6, 2.7 and 2.8 shows map generated during MR-SLAM. Local map developed by Robot 1 is shown in Figure 2.6 while local map generated by Robot 2 is shown in Figure 2.7. Then these two maps are compared for common areas and merged together to form a global map shown in Figure 2.8.

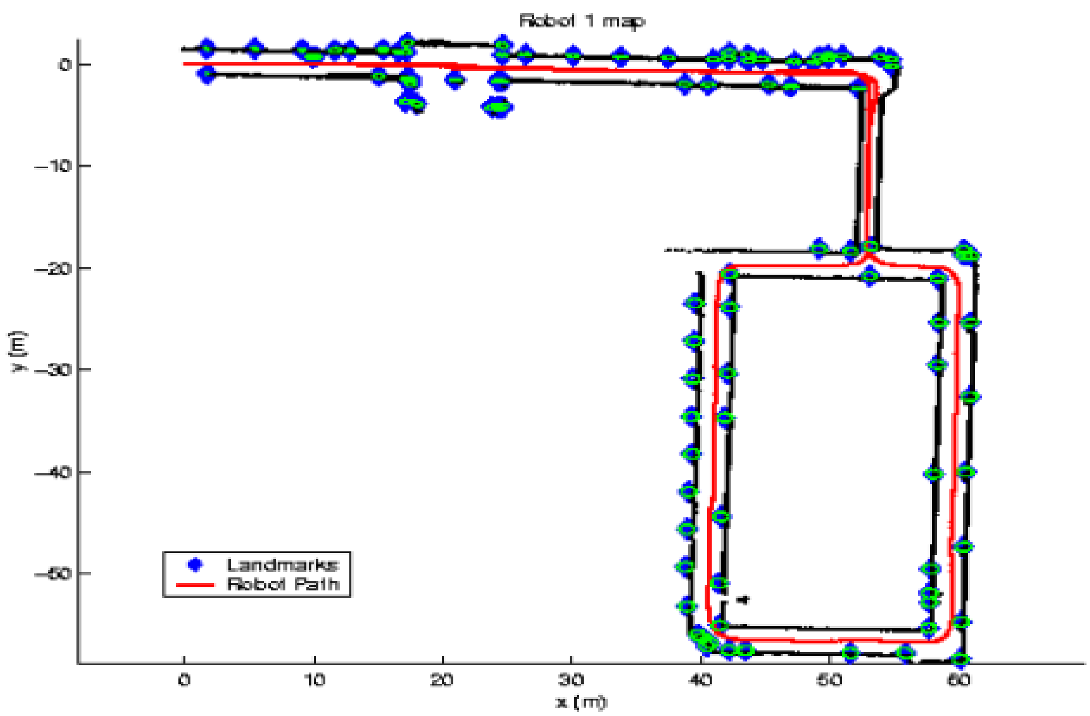


Figure 2.6: Map developed by Robot 1 [3].

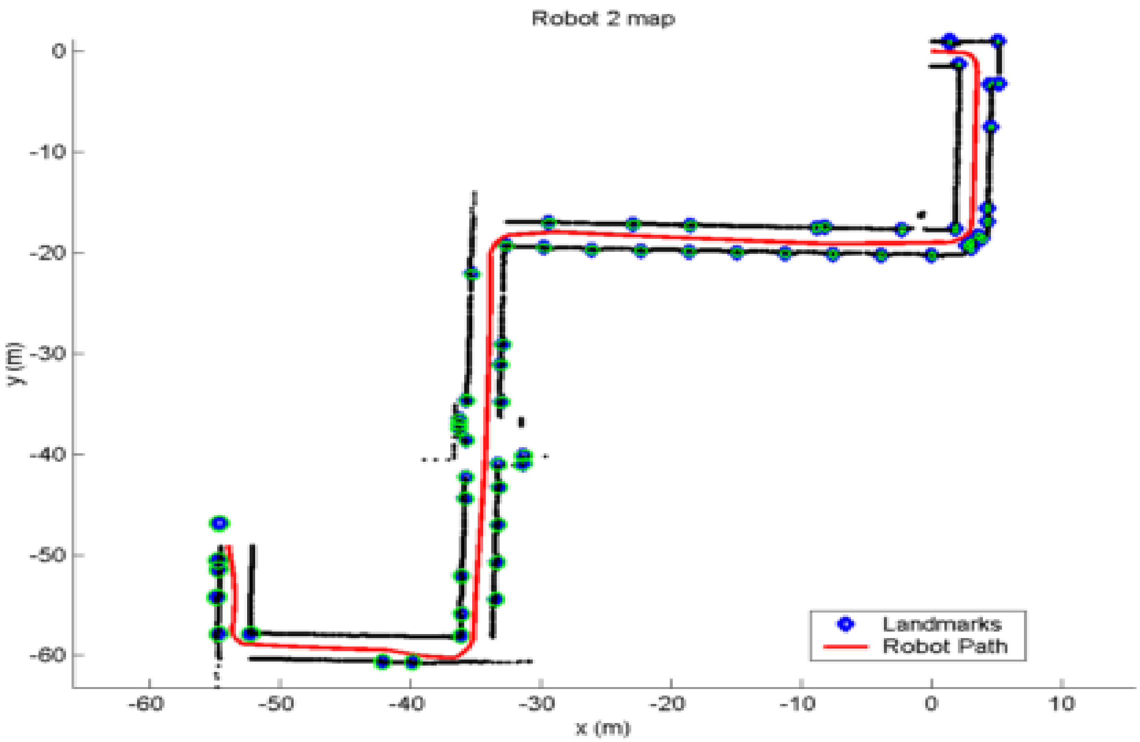


Figure 2.7: Map developed by Robot 2 [3].

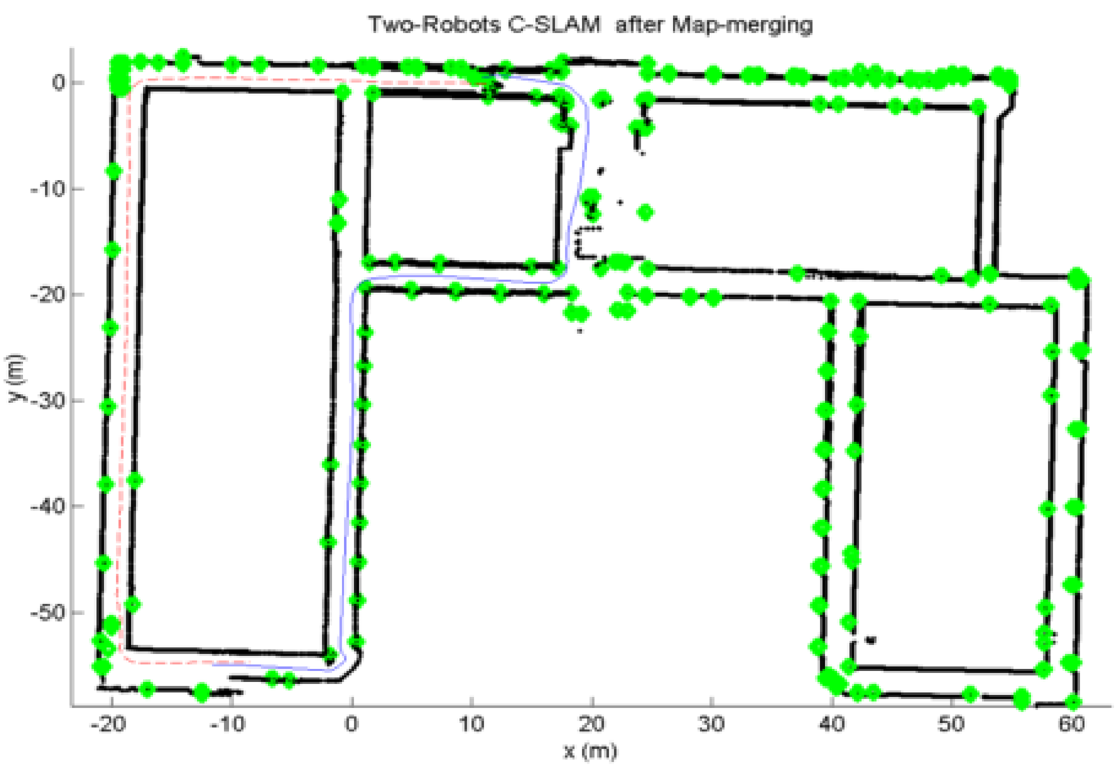


Figure 2.8: Common map after map merging [3].

CHAPTER 3: IMPLEMENTATION

3.1 System Overview

The objective of this research was to develop a simulation environment for multiple robots and to implement a SLAM algorithm. Figure 3.1 can be used to represent a multi-robot distributed environment with two robots. MR-SLAM environment

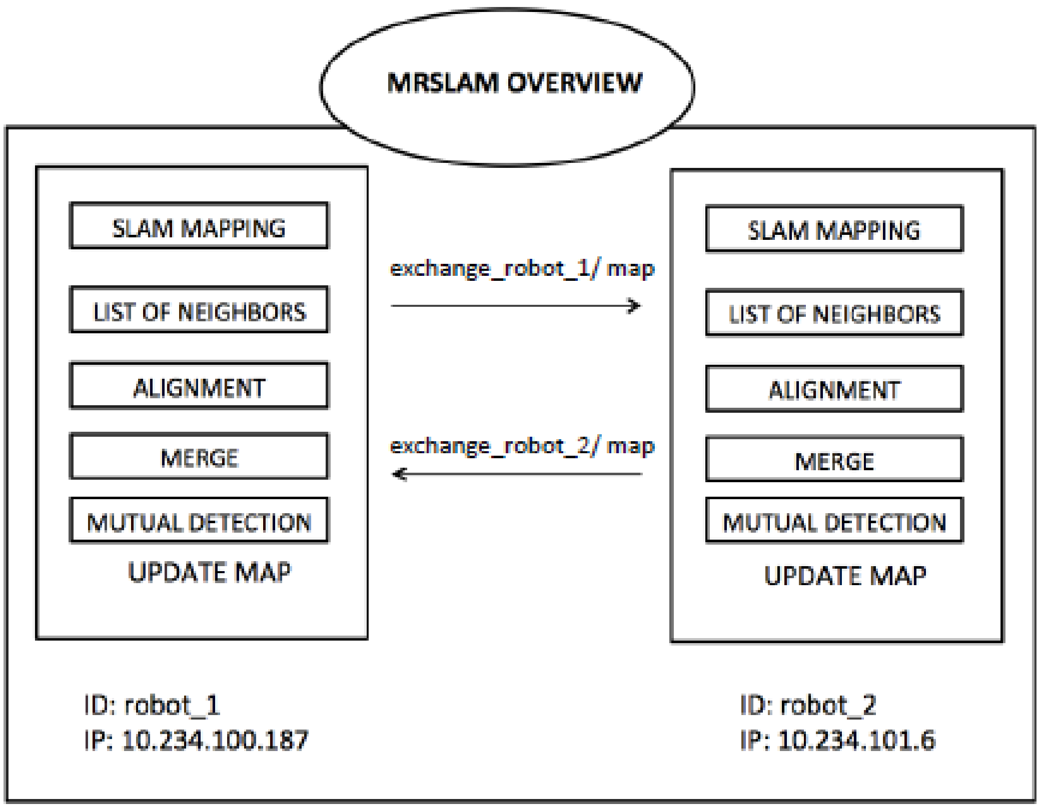


Figure 3.1: Overview of multi-robot architecture.

focuses on an idea that each robot will run an individual SLAM and develop a local map with a goal of creating a global map by merging the information collected. A multi-robot system can be considered as an integration of a real world, TurtleBot platform (Kobuki base, sensors and wheels) and *Robot Operating System* (ROS).

3.2 TurtleBot

TurtleBot is a low-cost development platform with open source software [4]. It was designed at Willow Garage by Melonee Wise and Tully Foote. This robotic platform was developed to navigate through the house and it can be qualified as a home service robot due to its core technologies such as SLAM and navigation. There are three types of TurtleBots named as 'TurtleBot', 'TurtleBot2' and 'TurtleBot3'.

Figure 3.2: TurtleBot2 [4]

TurtleBot2 has a Kobuki base, a Kinect 3D sensor, and a laptop running ROS. A user can use TurtleBots to implement real-time obstacle avoidance, autonomous navigation algorithms explore the house using a laptop or an android device, and implement SLAM algorithms. This system uses TurtleBot2 for an implementation because of its features and availability.

3.3 Gazebo

The Gazebo is a 3D rigid body robot simulator developed for the robotic environment. 'gazebo_ros_pkg' is a set of ROS packages used for an interface to simulate a robot in Gazebo environment [15]. ROS messages are used for the communication between a simulator and ROS.

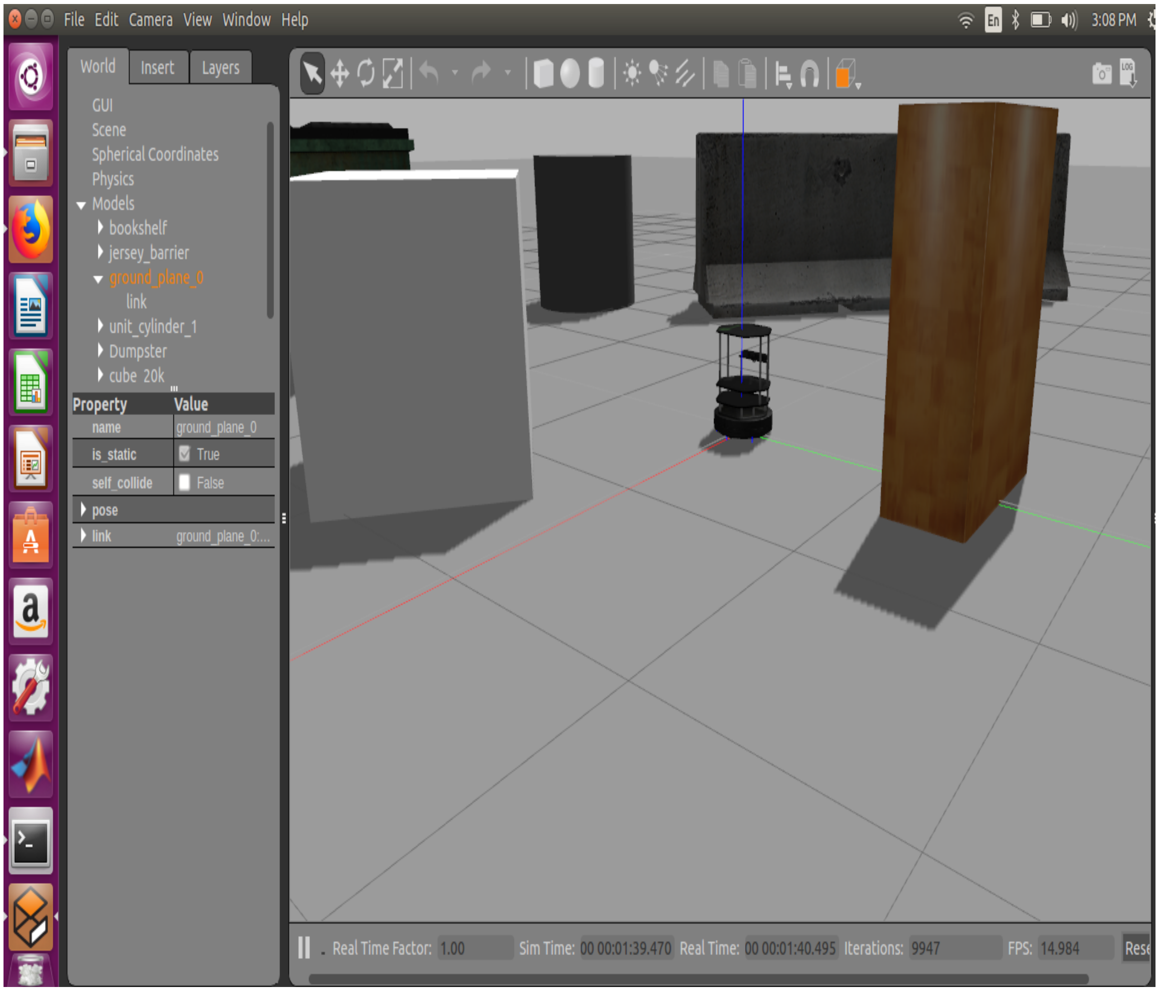


Figure 3.3: Gazebo world with TurtleBot

Figure 3.3 shows a robotic environment for TurtleBot. Simulators are essential aspects of a robotic toolkit because of its scalability, ease of implementation, and absence of real-world unwanted data. Robotic applications can be regressively tested with the realistic scenarios using a robot simulator.

Gazebo simulations are launched using a 'launch file' in a ROS package. These ROS packages need to be compiled using *catkin_make*. The simulation in 3.3 was launched using a *.launch* file 'turtlebot_world.launch' located in the package 'Turtlebot_gazebo'. Launch files are written in XML format. These launch files are comprised of a path to world file with an extension of *.world* (environment containing the list of objects used for the simulation), parameters and arguments those are passed along with it and a path to a 'urdf file' (*Unified Robot Description Format*). An urdf file is a descriptive file of a robot which contains all the information related to the robot dynamics, robot appearances and 'links and joints' of a robot. For example, the construction of a TurtleBot includes a Kinect 3D sensor mounted on a black mobile Kobuki base, battery on the same base, motors, and all the appearances of a robot are defined in an urdf file. When the '.launch' file is launched, it publishes all the topics related to the spawned robot on the network.

3.4 ROS

Robot Operating System can be defined as a framework developed for creating complex and robust robot environments across a wide variety of robotic platforms. ROS is a collection of tools and libraries [16]. ROS is an open source platform which was built for collaborative robotics software development. Discrete development in the robotics has collaborated in ROS.

ROS uses a central master known as a '*roscore*' for the low-level publisher/subscriber communication architecture. This communication enables ROS for efficient communication. Suppose task A requests data from the sensor and task B processes that data. Data published by task A is written to a file by ROS, which can be republished at a later time. Task B can request the data without caring about the source. This flexibility and modularity in the system improves the encapsulation and promotes code reuse.

ROS consist of nodes, topics, services, messages, bags, master and parameter server. A 'node' is a process which is used for the computation. It is a program which will process the data. A robot control system comprises many nodes. For example, the software driving a Kinect sensor is a node, a laser range finder is a node, there will be a node dedicated for robot localization and path planning. Nodes are written in programming languages such as C++, Python, Java or Lisp. This flexibility encourages many programmers to develop custom nodes which perform special services.

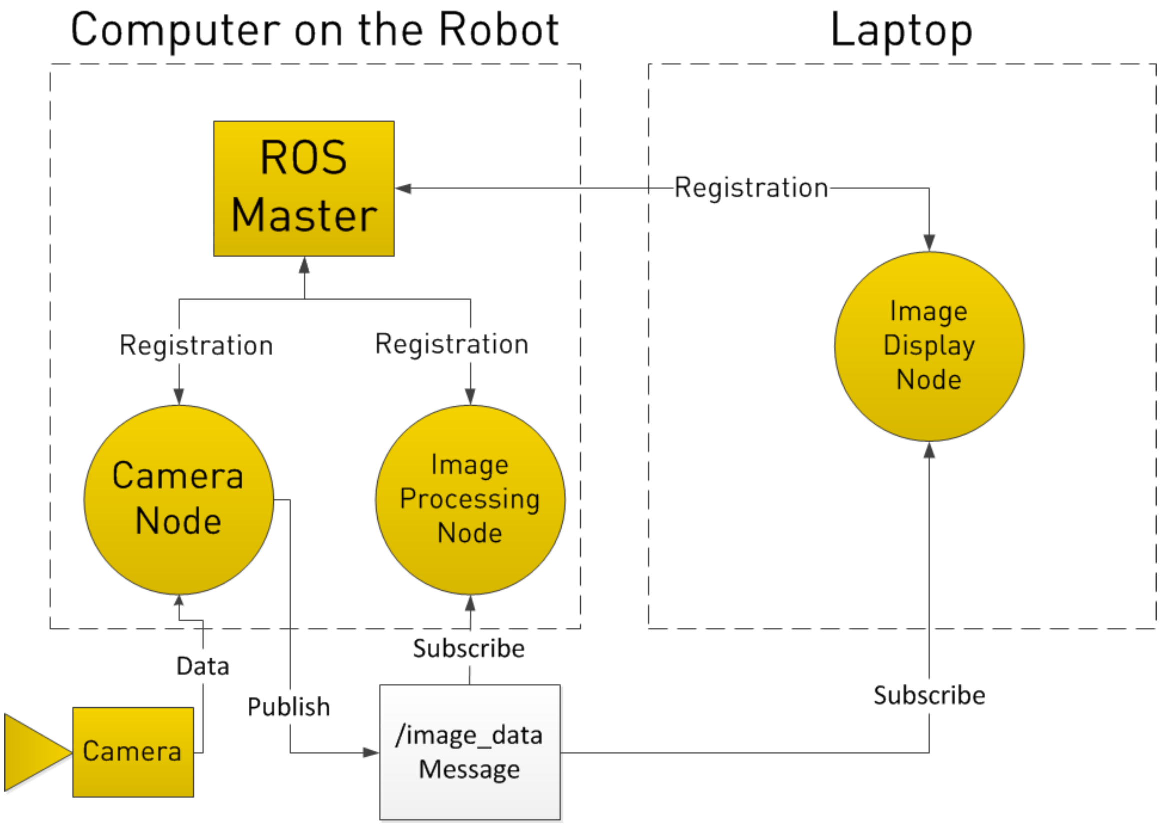


Figure 3.4: ROS Structure

For a robot to perform a specific task, there has to be communication between nodes. Path planning algorithm might ask for services from laser range finder and robot wheel motors. In this scenario, communication between these nodes is achieved over a bus called as 'Topic'. Topics can be explained as a medium over which nodes communicate with each other. Topics, due to its anonymous publish/subscribe semantic, keeps source and client separate from each other. Nodes are not aware of the other node to which they are communicating. Nodes that are interested in the data, subscribe to the topic and nodes those are generating the data, publishes to the topic. In the example provided above, a laser range finder and a motor control both publish to the respective topic. All the topics present on the roscore can be viewed using *rostopic list* command.

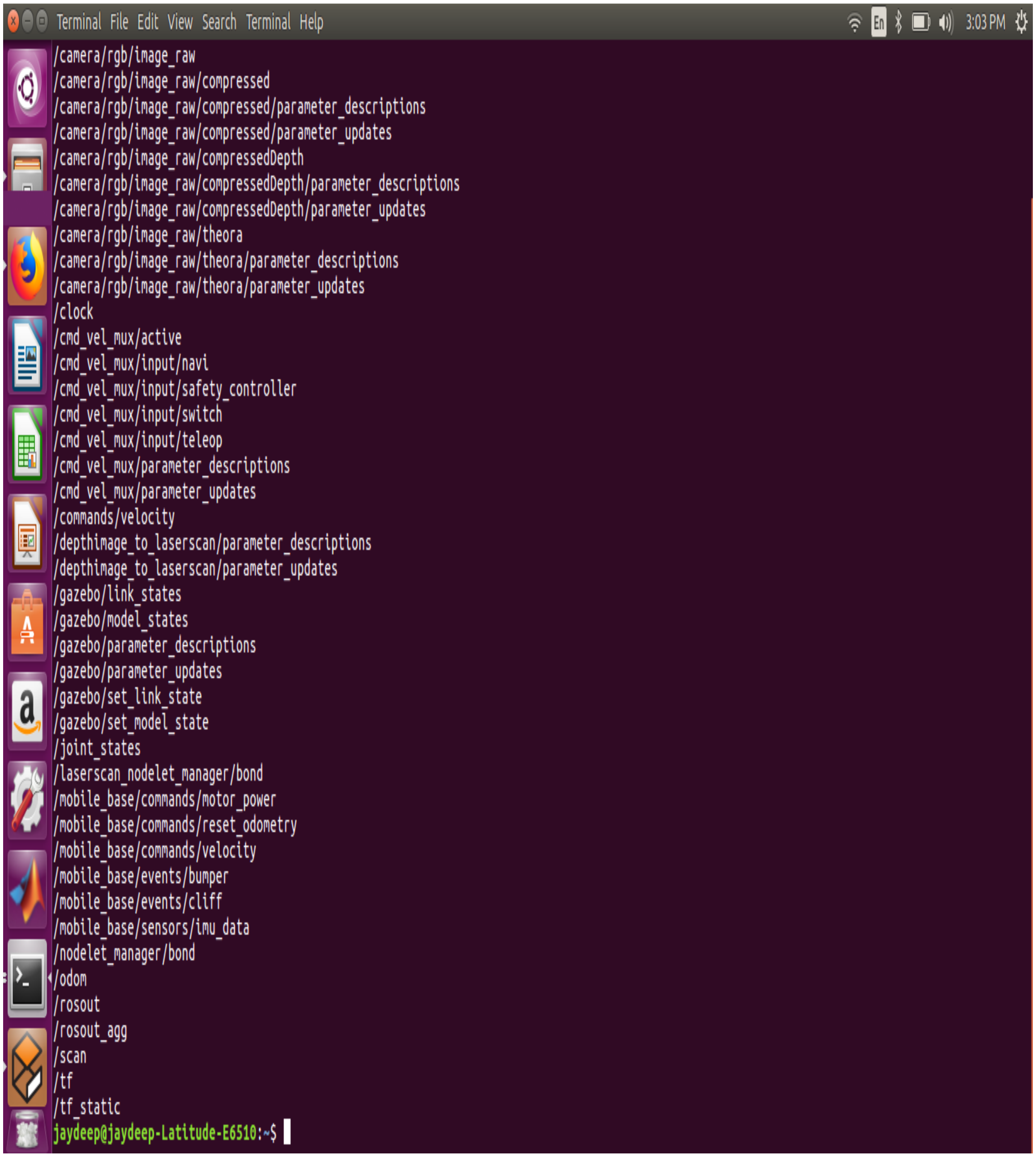


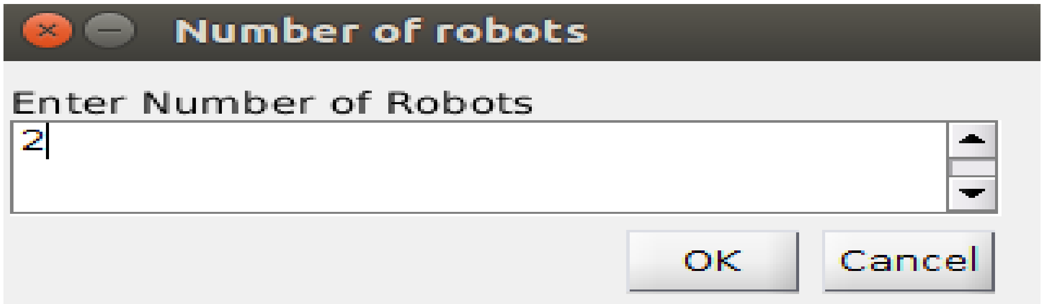
Figure 3.5: ROS Topics

Figure 3.5 displays topics presented on a network for an environment having a TurtleBot. Every topic can carry a message to the node for further computing. For example, '/scan' publishes data obtained from laser range finder and for a node to get the data related to an odometry, it needs to subscribe to the topic '/odom'.

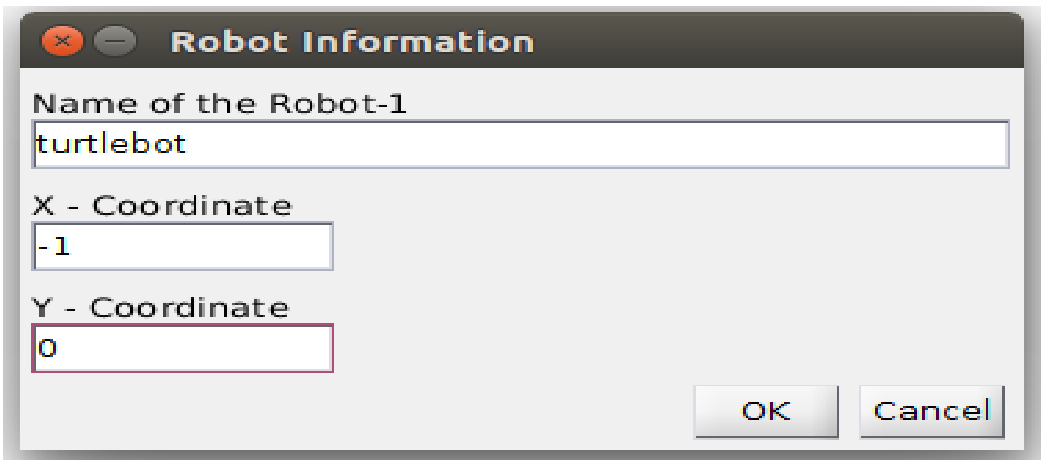
Robot movement can be controlled by publishing linear and angular velocities to the '/mobile_base/commands/velocity'.

3.5 Multi Turtlebot Spawning Tool

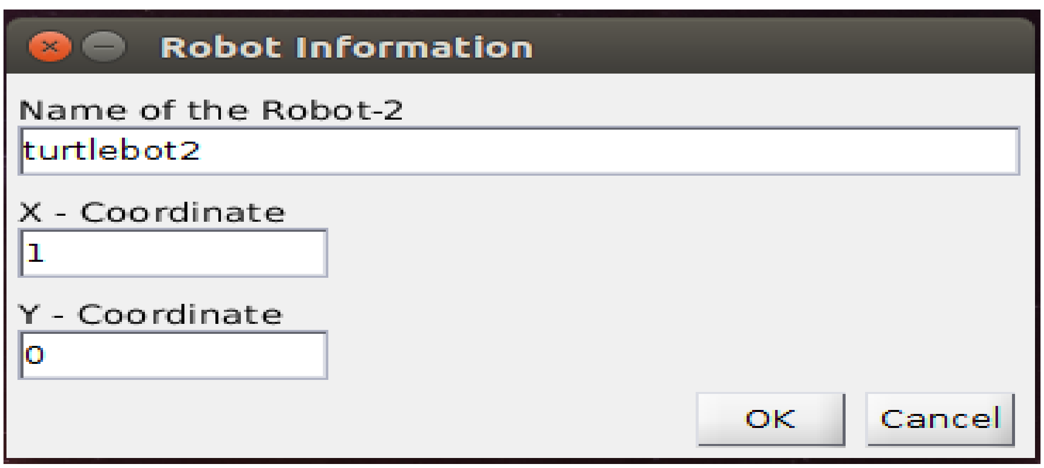
The default Gazebo launch file can spawn a TurtleBot in a Gazebo environment and publishes all the topics related to it. The system requirements needed a good simulator for a multi-robot environment. Lack of such robotic simulator was an encouraging factor to write a MATLAB script that will take the number of robots along with their respective names and locations on the map. This script will generate a launch file in XML format which can be used to spawn multiple robots in the environment. Following Figure 3.6 depict the GUI created by a MATLAB script to generate a launch file.



(a) Number of Robots



(b) Name and Position for TurtleBot



(c) Name and Position for TurtleBot

Figure 3.6: GUI for MATLAB script to produce a .launch file

A TurtleBot launched in the environment can be communicated through the topics related to it. To communicate with these two TurtleBots, there was a need for individually associated topics. This was achieved using ‘namespace’ in the launch file. It allowed us to launch different TurtleBots in Gazebo and communicate to each one individually. Figure 3.7 depicts the general concept of using namespace in the multi-robot environment.

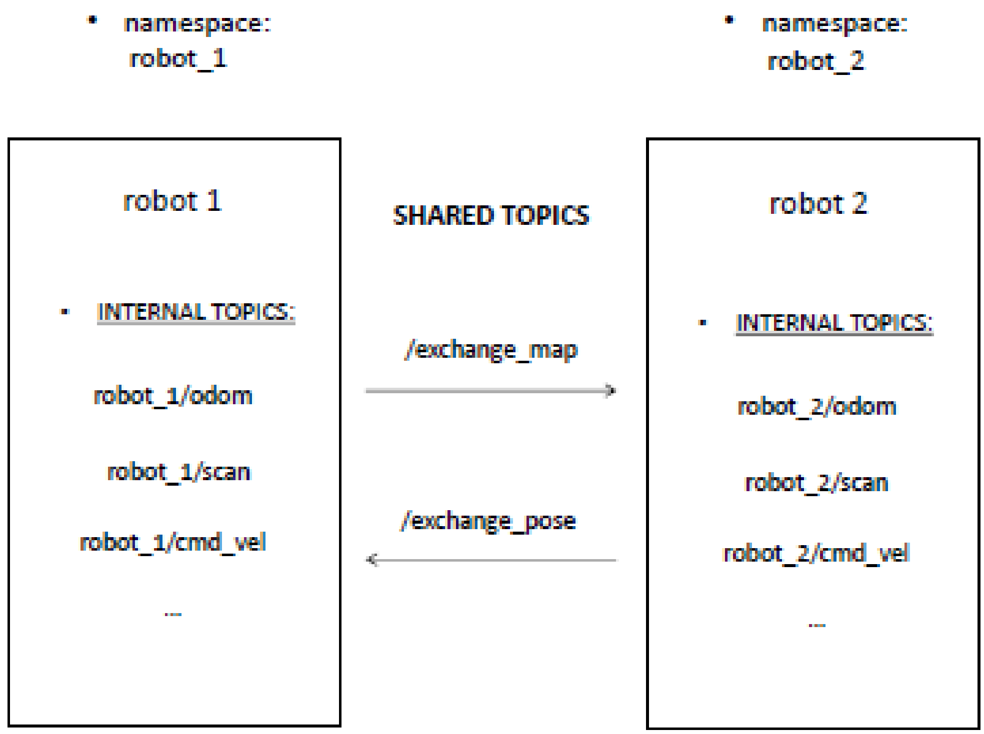


Figure 3.7: Topics namespaced for two TurtleBots in Gazebo

The launch file created by the MATLAB script is shown in Figure 3.8. It mainly comprises of a path to an empty world, names of the TurtleBots, namespacing for the respective topics and their coordinate positions. Figure 3.9 shows two TurtleBot spawned by launch file generated using the MATLAB script. Two TurtleBots were named as ‘TurtleBot’ and ‘turtlebot2’.

```xml
<?xml version="1.0" encoding="utf-8"?>
<launch>
  <arg name="base" value="$(optenv TURTLEBOT_BASE kobuki)"/>
  <arg name="battery" value="$(optenv TURTLEBOT_BATTERY /proc/acpi/battery/BAT0)"/>
  <arg name="gui" value="true"/>
  <arg name="stacks" value="$(optenv TURTLEBOT_STACKS hexagons)"/>
  <arg name="3d_sensor" value="$(optenv TURTLEBOT_3D_SENSOR kinect)"/>
  <arg name="pos_x" value="-1"/>
  <arg name="pos_y" value="0"/>
  <arg name="name" value="jv_bot"/>
  <include file="$(find gazebo_ros)/launch/empty_world.launch">
    <arg name="use_sim_time" value="true"/>
    <arg name="debug" value="false"/>
    <arg name="world_name" value="$(find turtlebot_gazebo)/worlds/empty.world"/>
  </include>
  <group ns="/turtlebot">
    <param name="tf_prefix" value="turtlebot"/>
    <include file="$(find turtlebot_gazebo)/launch/includes/$(arg base).launch.xml">
      <arg name="base" value="$(arg base)"/>
      <arg name="stacks" value="$(arg stacks)"/>
      <arg name="3d_sensor" value="$(arg 3d_sensor)"/>
      <arg name="pos_x" value="-1"/>
      <arg name="pos_y" value="0"/>
      <arg name="name" value="turtlebot"/>
      <arg name="namespace" value="turtlebot"/>
    </include>
    <node name="robot_state_publisher" pkg="robot_state_publisher" type="robot_state_publisher">
      <param name="publish_frequency" type="double" value="30.0"/>
    </node>
    <node args="manager" name="laserscan_nodelet_manager" pkg="nodelet" type="nodelet"/>
    <node args="load depthimage_to_laserscan/DepthImageToLaserScanNodelet laserscan_nodelet_manager" name="depthimage_to_laserscan" pkg="nodelet" type="nodelet">
      <param name="scan_height" value="10"/>
      <param name="output_frame_id" value="/camera_depth_frame"/>
      <param name="range_min" value="0.45"/>
      <remap from="image" to="/camera/depth/image_raw"/>
      <remap from="scan" to="/scan"/>
    </node>
  </group>
  <group ns="/turtlebot2">
    <param name="tf_prefix" value="turtlebot2"/>
    <include file="$(find turtlebot_gazebo)/launch/includes/$(arg base).launch.xml">
      <arg name="base" value="$(arg base)"/>
      <arg name="stacks" value="$(arg stacks)"/>
      <arg name="3d_sensor" value="$(arg 3d_sensor)"/>
      <arg name="pos_x" value="1"/>
      <arg name="pos_y" value="0"/>
      <arg name="name" value="turtlebot2"/>
      <arg name="namespace" value="turtlebot2"/>
    </include>
    <node name="robot_state_publisher" pkg="robot_state_publisher" type="robot_state_publisher">
      <param name="publish_frequency" type="double" value="30.0"/>
    </node>
    <node args="manager" name="laserscan_nodelet_manager" pkg="nodelet" type="nodelet"/>
    <node args="load depthimage_to_laserscan/DepthImageToLaserScanNodelet laserscan_nodelet_manager" name="depthimage_to_laserscan" pkg="nodelet" type="nodelet">
      <param name="scan_height" value="10"/>
      <param name="output_frame_id" value="/camera_depth_frame"/>
      <param name="range_min" value="0.45"/>
      <remap from="image" to="/camera/depth/image_raw"/>
      <remap from="scan" to="/scan"/>
    </node>
  </group>
</launch>
```

Figure 3.8: Launch file for two TurtleBots in Gazebo, located at [-1 0 0] and [1 0 0]

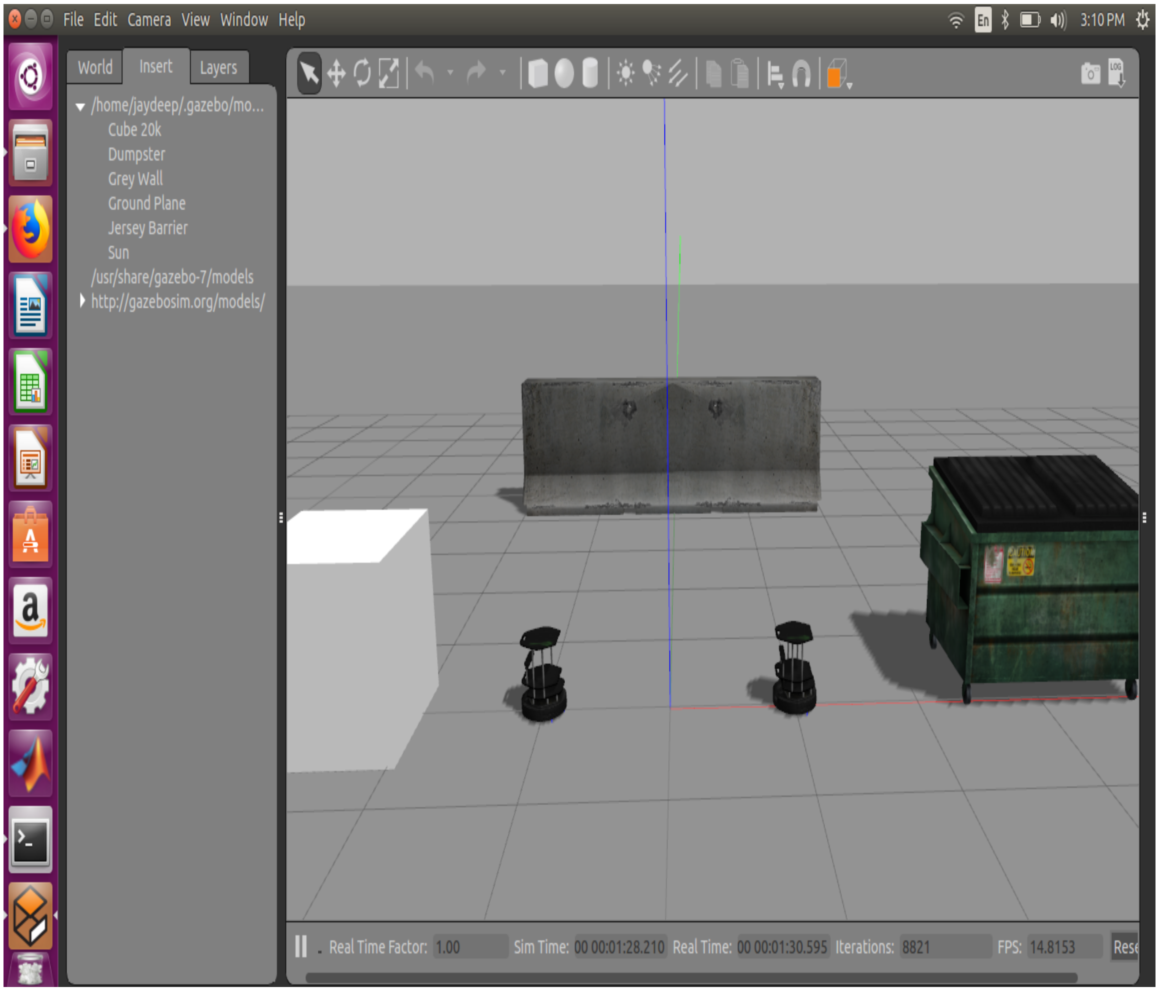
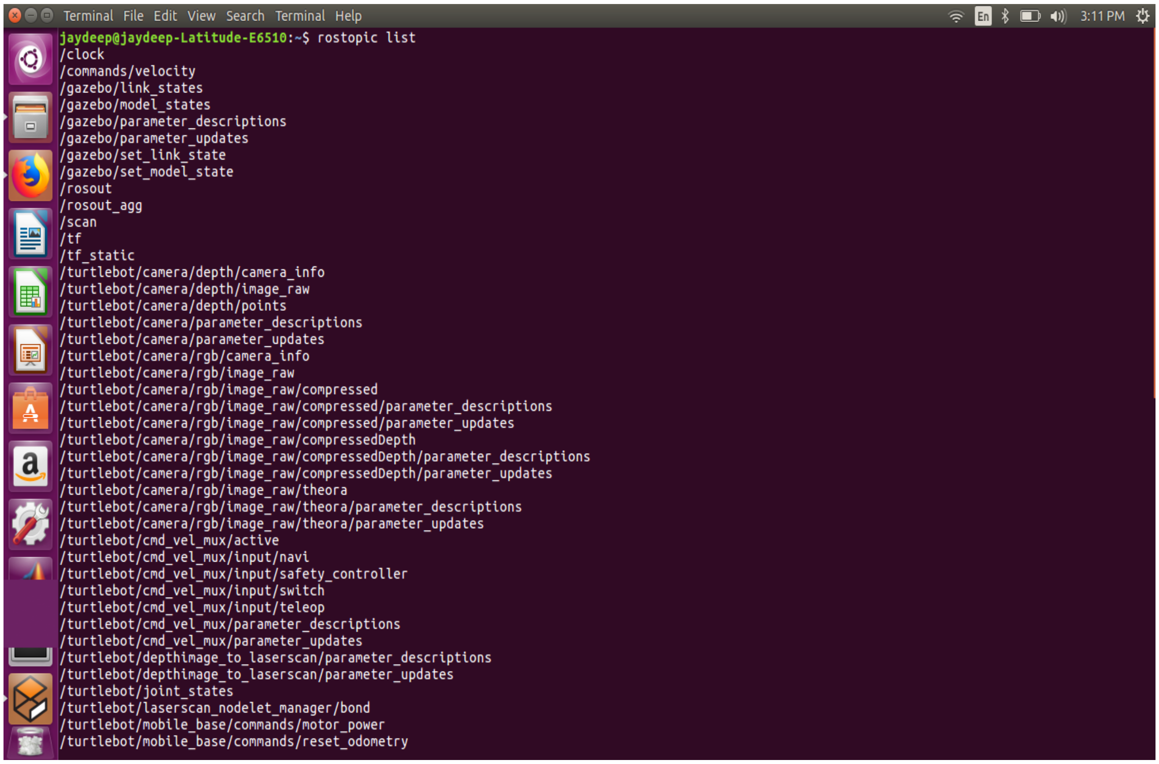


Figure 3.9: Two TurtleBots in Gazebo, located at [-1 0 0] and [1 0 0]

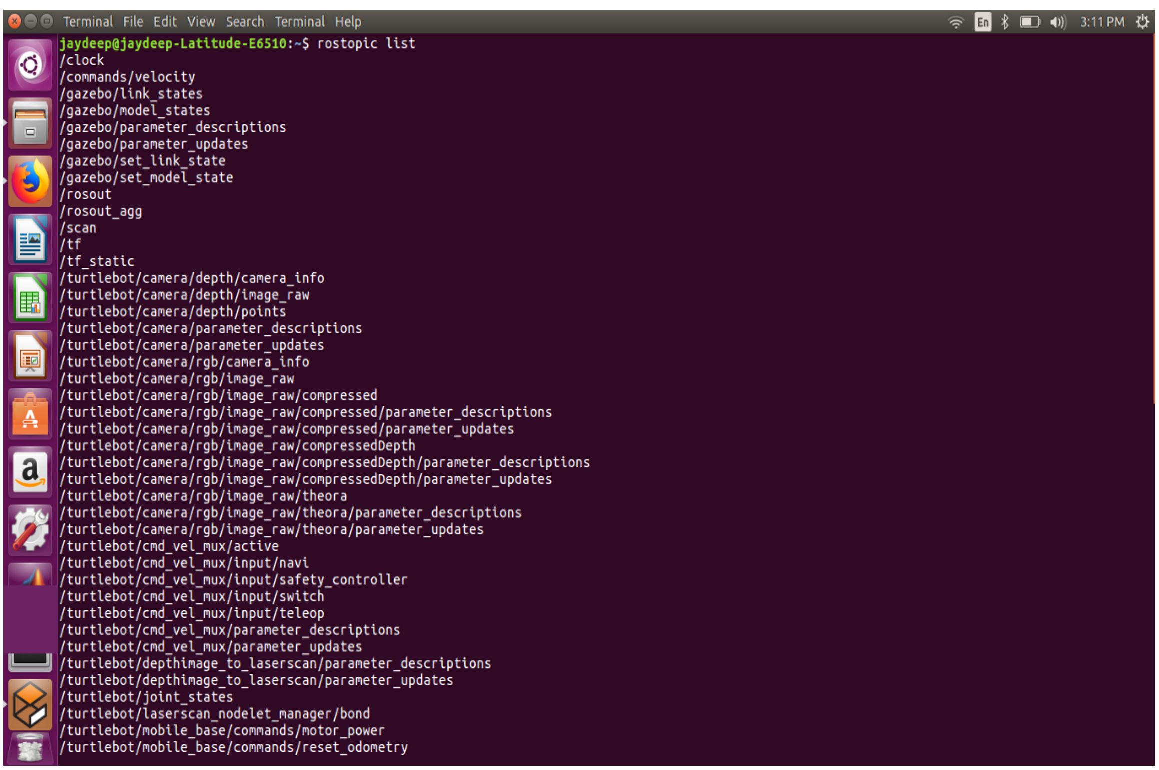
Figure 3.10 shows topics published on ROS with the namespace. All the topics related to turtlebot are namespaced under '/turtlebot' and in the same manner all the topics related to turtlebot2 are namespaced under '/turtlebot2'. To drive 'turtlebot' around, its linear and angular velocities should be published through a message to the topic '/turtlebot/mobile_base/commands/velocity' and similarly to drive 'turtlebot2', the topic used should be '/turtlebot2/mobile_base/commands/velocity'.

3.6 Implementaion of Multi-Robot System

Figure 3.11 represent the high-level block diagram of the system. This system is divided into two major parts: Development of simulation tool and Implementation of



(a) Topics for 'turtlebot'



(b) topics for 'turtlebot2'

Figure 3.10: Topics with namespacing

SLAM algorithm. The MATLAB script takes inputs from the user about the names and the positions of the TurtleBots. Then it creates a launch file which has a path to an empty world file, a path to base launch file, individual robot positions, and list of the sensors associated with a robot. The launch file can be used to create a world and to populate it with the TurtleBots at provided locations. When the simulation system is launched, algorithms like path finding [17, 18] and SLAM can be used to perceive the environment and navigate through it.

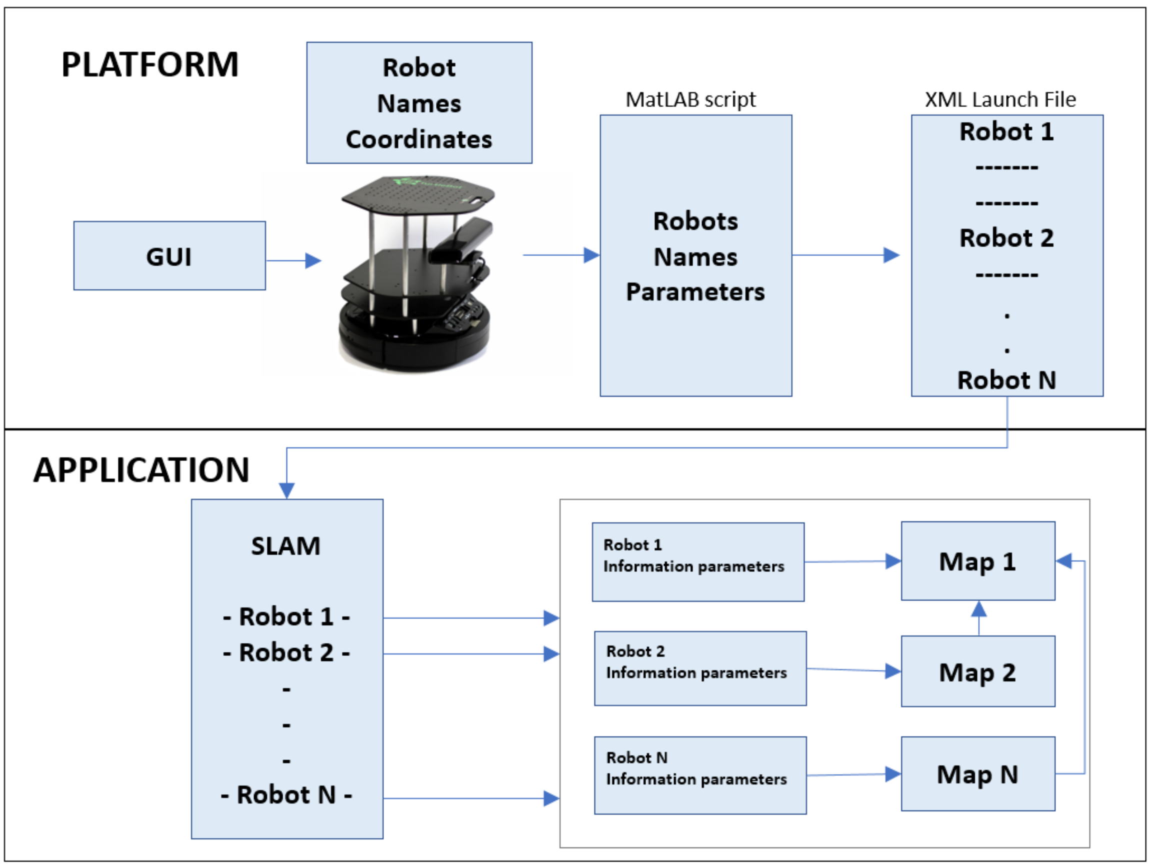


Figure 3.11: Block Diagram of a System

3.6.1 Creation of World

The world was created in Gazebo around TurtleBots through MATLAB. This world consisted of four Grey walls, two obstacles and six different landmarks. These landmarks were colored balls placed on a bar each with a different color from one another and so they were identified by the unique signature associated with it. Table 3.1 lists all the landmarks along with their color, signature and positions in the system.

Table 3.1: Landmark Placement and respective Signatures

Landmark	Color	Signature	Position
Landmark1	Red	[1 0 0]	[3.5,0.2]
Landmark2	Green	[0 1 0]	[1.8,3.0]
Landmark3	Blue	[0 0 1]	[-1.4,3.0]
Landmark4	Yellow	[1 1 0]	[-3.1,0.2]
Landmark5	Cyan	[0 1 1]	[-1.4,-2.6]
Landmark6	Magenta	[1 0 1]	[1.8,-2.6]

Figure 3.12 shows the created world. Two TurtleBots are placed such that turtlebot1 could see four landmarks and turtlebot2 could map three landmarks. Out of these landmarks, one landmark was common between them which was used to determine the transform between local references frames of the TurtleBots.

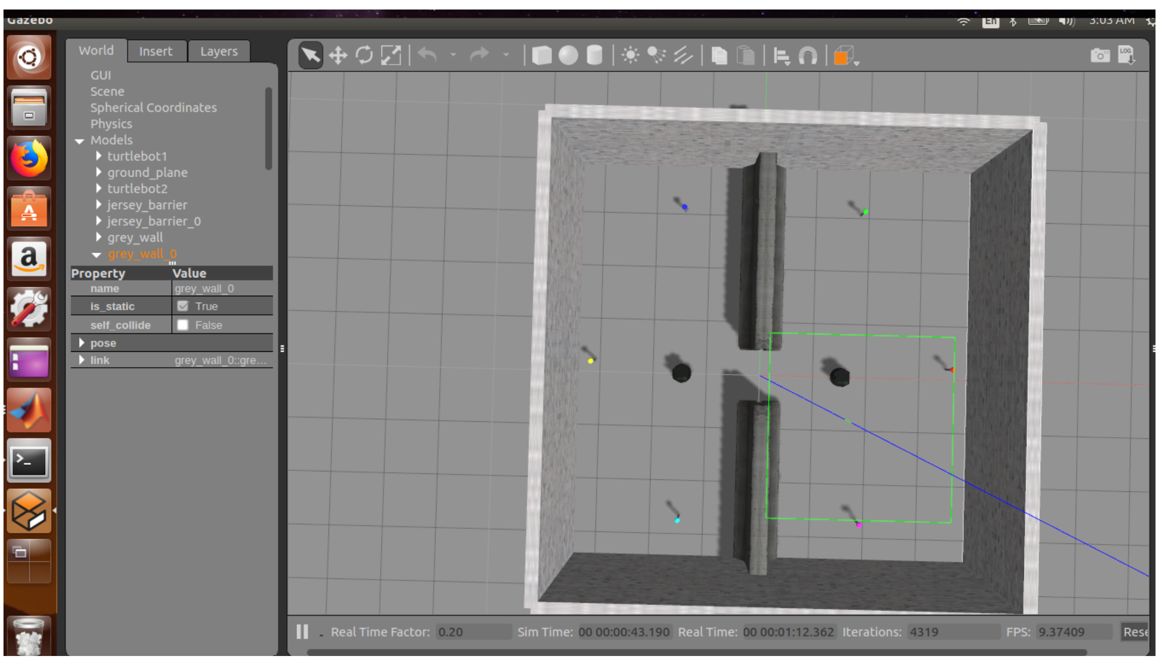


Figure 3.12: Gazebo world with walls, landmarks and TurtleBots

3.6.2 Sparse Extended Information Filter

When a system in Gazebo, with TurtleBots and landmarks, is launched, a SLAM algorithm using SEIF with known correspondence was implemented to verify the working of the system. In SEIF, posterior is represented by information matrix ‘Ω’ and information vector ϵ. Information matrix ‘Ω’ is a square matrix containing the robot and landmark states. Figure 3.13 and 3.14 shows information matrix and information vector with single robot and 4 landmarks. Here X1 denotes the robot and L1, L2, L3 and L4 are the landmarks.

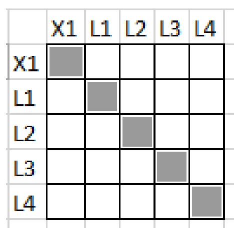


Figure 3.13: Information Matrix

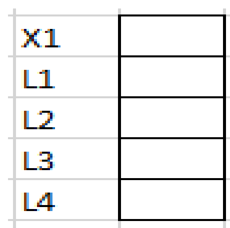


Figure 3.14: Information Vector

For multi-Robot applications, every TurtleBot runs its own SEIF to build its local map. Each robot communicates with the other using a network. This network structure was simulated in the Gazebo using a transmission and reception of ROS messages over different topics. Each individual TurtleBot maintain it’s own local posterior and a local map according to its reference frame. When a common landmark is detected in both the TurtleBots, the transformation between their local reference frames is calculated. This transformation is then applied to the posterior received from another TurtleBot to merge the data and create a global map. As stated in the survey, SEIF is divided into four steps:

- Measurement update step
- Motion update step
- Sparsification step

- State estimation step

Measurement Update: Whenever a landmark was observed by a TurtleBot through its camera, it's distance and angle from the TurtleBot was calculated by using arithmetic equations used in optics. The correspondence test was conducted by checking the index and signature of that landmark. All the observed landmarks by the individual TurtleBots were stored using the respective index of the landmark. When the correct landmark was identified and if it was never seen before, it's location was computed in the local map and it is then added to the state vector. This step introduced new links in the information matrix and information vector between a robot and the landmark. If that landmark was seen before and it was present in the maintained data structure for correspondence vector then previously estimated information was updated in the information matrix and information vector. Figure 3.15 illustrate the effect of the measurement update on information matrix and information vector. When turtlebot1 observed L1 (y1) and L2 (y3), new links were established between X1 and L1. Similarly for the turtlebot2 (x2) and L3 (y3).

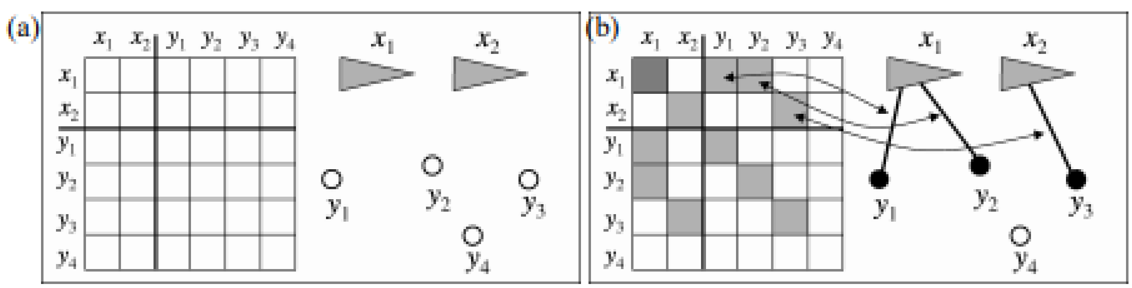


Figure 3.15: Effect of measurement update

Figure 3.15[a] and 3.15[b] depicts the information matrix before and after landmark is observed.

The algorithm depicted in Figure B.1 was used to either introduce a link or to update the information in an implementation of filter when a landmark was observed. Here Q is the zero mean white Gaussian measurement noise matrix, C is a correspondence list and μ is a state vector of the individual robot.

Motion Update: The motion command issued to any TurtleBot caused it to add some uncertainty to the information available between robot pose and landmark state. This introduced uncertainty reduced the strength of elements related to the robot pose vector. This algorithm shifted the lost information to the links between two landmarks. Figure 3.16 depicts the effects of motion in the information matrix. As it can be seen, the robot's movement reduced information between L1, L2 and turtlebot1 and also in between L3 and turtlebot2.

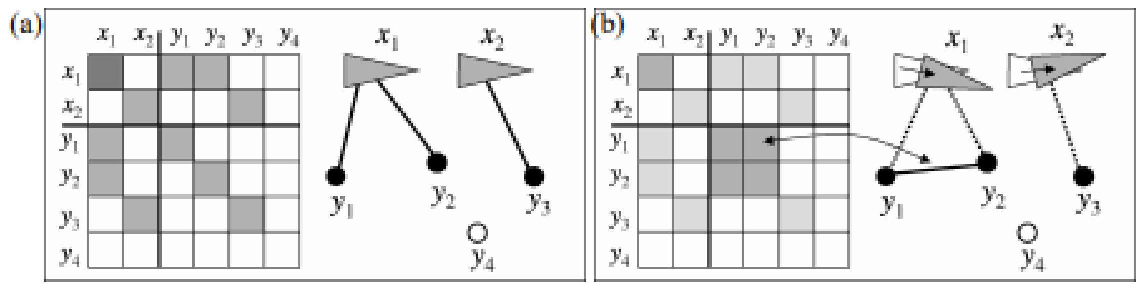


Figure 3.16: Effect of motion update

The algorithm used to implement the motion update step is in Figure B.2. Here F_x is a projection matrix which extracts robot pose elements. δ is a motion model which determines the effect of motion commands(angular velocity and linear velocity) on the current state (pose) of the robot.

Sparsification: One of the advantages of SEIF over EKF is that it is less computationally expensive and consumes less memory. For every update in EKF, all the elements in covariance matrix are populated, making its memory requirement to be quadratic to the number of landmarks i.e its memory requirements increases quadratically with an increase in the number of landmarks. Sparsification in SEIF helps to reduce these constraints by removing the unnecessary links in the landmarks. In SEIF, all the observed landmarks were divided into two main categories: 'active landmarks' and 'passive landmarks'. A robot performed update on some pre-determined number of landmarks, also called as active landmarks, making the information matrix sparsed at all the time. Figure 3.17 shows the graphical representation of the effects on the information matrix before and after Sparsification step.

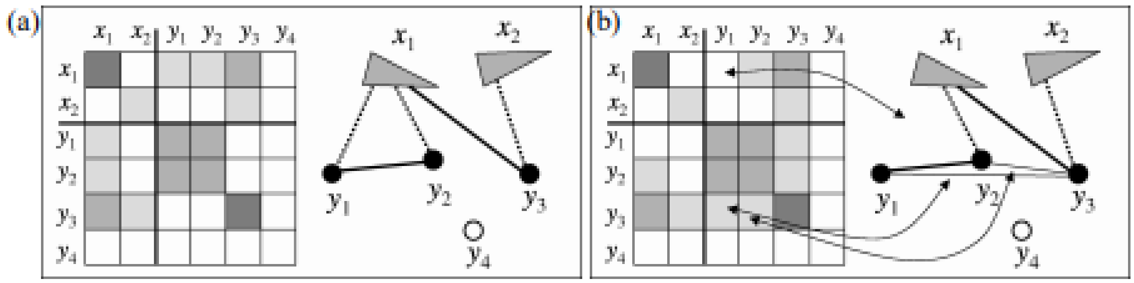


Figure 3.17: Effect of sparsification

Figure B.3 depicts the algorithm used to implement the sparsification step.

State Estimation: SEIF algorithm uses respective state vector elements for the motion as well as measurement update steps. State vector can be computed using an equation $\mu = \Omega^{-1} * \epsilon$, but this calculation includes high computational costs due to a multiplication of large matrices. Due to this reason, SEIF algorithm depicted in Figure B.4 was used to estimate state from the information vector and information matrix. In Figure B.4, F_x is a projection matrix which extracts elements respective to the robot state for the calculation of robot state estimate or the elements respective to the landmark state for the computation of landmark state estimate from the information matrix.

Graphical representation in Figures 3.15, 3.16 and 3.17 shows that the information matrix is a squared matrix of state estimates for both the robots and all the landmark state estimates, but in the actual implementation of this research, a separate state vector and information matrix were maintained for each individual robot and only contained it's own robot state and landmark states as shown in Figure 3.13 and Figure 3.14.

3.6.3 SEIF With Known Initial Pose

When SEIF SLAM was implemented with known initial position, both TurtleBots knew their own initial pose with respect to the global reference frame. Each TurtleBot predicted the coordinates of a landmark with respect to the global reference frame (A coordinate system with the origin located such that robots positions are known). In such a scenario, map merging is a simple extension of combining the information. Since both TurtleBots followed a global coordinate system, computation of transformations between their local frames was not needed. To illustrate this case, an environment was populated with landmarks located at [3,0], turtlebot1 at [-1,0] and turtlebot2 at [1,0]. Both TurtleBots predicted the landmark at [3,0] instead of [4,0] and [2,0] respectively.

3.6.4 SEIF with Unknown initial pose

When the TurtleBots were not aware of their respective initial poses, each TurtleBot mapped the environment with its own local reference frame, which had an origin located at the starting position of the robot with X-axis as the facing direction at time $t = 0$ (starting point of odometry). To illustrate this, consider two TurtleBots at [-1,0] and [1,0] and a landmark at [2,0]. For the same landmark located at '3' distance on X-axis and '0' distance on Y-axis from turtlebot1 and '1' distance on X-axis and '0' distance on Y-axis for turtlebot2 is plotted as [2,0] and [-2,0] respectively irrespective of their position in the global coordinate system. Each TurtleBot maintained posterior over its own local reference frame.

The basic difference between centralized and decentralized MR-SLAM is the distribution of controlling and data processing capabilities amongst the robots. In this implementation, a decentralized approach was taken where turtlebot1 communicated with turtlebot2 each time a landmark is seen. Data was in terms of the information matrix, information vector, correspondence vector and state vector. ROS messages were used to simulate the data transfer. For the world depicted in Figure 3.12, turtlebot1 and turtlebot2 were able to map landmarks 1, 3, 4, 5, and 1,2,6 respectively. Both TurtleBots continued to implement individual SLAM in the local reference frames unless and until a common landmark was detected. Once a correspondence was detected between them from the correspondence data shared amongst the TurtleBots, turtlebot1 started to transmit its information matrix and information filter to turtlebot2, which in turn was used to detect the transformation between two reference frames.

3.6.5 Map Merging

A key concept in MR-SLAM is map merging. When two robots have determined the correspondences and identified the common landmarks, information was mathematically manipulated such that one robot could interpret information obtained from another and this information was then amended to its own Gaussian parameters. This process of fusing the information is known as map merging and the process of aligning the posterior through mathematical manipulation is commonly known as 'Transformation of local coordinate frames'.

In an implementation of decentralized MR-SLAM system with turtlebot1 and turtlebot2, each time turtlebot1 detected a landmark, it sent a correspondence list to the other robot. This list was the array of indices for the landmarks observed be the TurtleBot. This landmark list was then compared to the correspondence list of self by turtlebot2 to detect mutual landmarks. These mutual landmarks were used to determine the transformation between two local reference frames.

For every point in turtlebot1's reference frame, it can be mapped in turtlebot2's reference frame with a translational shift and rotational shift. Both TurtleBots had some landmarks in common, which were used to determine the distance and orientation between the reference frames. Figure 3.18 depicts the general idea for the same. In order to map the data observed by another TurtleBot, all the observations from other TurtleBot had to be shifted by some distance and rotated by a particular angle.

Additive property of multivariate Gaussian parameters is exploited here to add the information regarding the common landmarks [5].

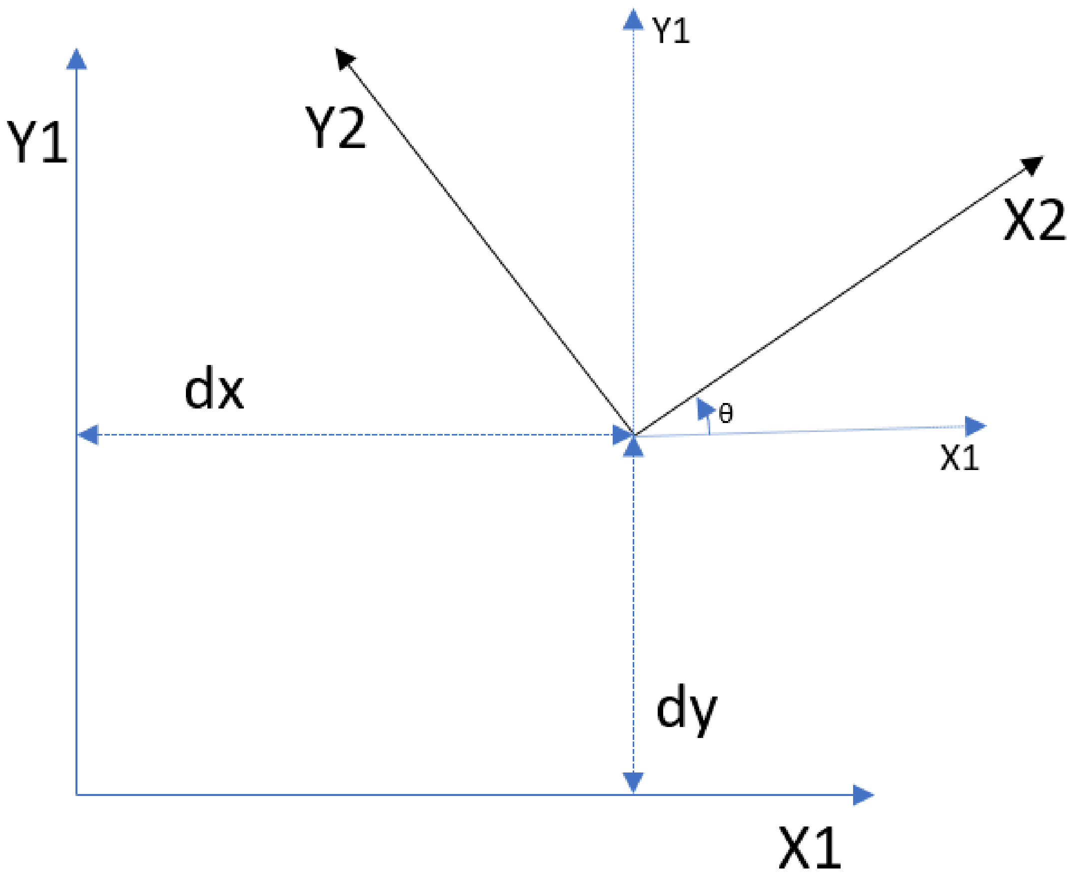


Figure 3.18: Transform between planes

$$\begin{bmatrix} x^{t1 \to t2} \\ y^{t1 \to t2} \end{bmatrix} = d + A * \begin{bmatrix} x^{t1} \\ y^{t2} \end{bmatrix} \tag{3.1}$$

$$A = \begin{bmatrix} cos\theta & -sin\theta \\ sin\theta & cos\theta \end{bmatrix} \tag{3.2}$$

$$d = \begin{bmatrix} dx \\ dy \end{bmatrix} \tag{3.3}$$

Once the distance and the rotational difference are known then the Equation 3.1 was be used to transform a point from turtlebot1's plane to the reference frame of turtlebot2. Equation 3.3 and 3.2 implies the translational shift and rotational shift between two reference frames respectively. A point [x,y] in a plane has to be shifted by dx and dy and rotated by the respective angle (θ is the counter clockwise difference between the reference frames) to represent in another plane.

Figure C.1 depicts the algorithm used to merge the maps using information matrix and information filter. Information matrix and information vectors from both the TurtleBots, displacement factor (d) and relative rotational angle ($\alpha(\theta)$) were input for the same.

Posterior from turtlebot1 was aligned with the turtlebot2 using the transform matrix and equations on line 4 and 5 in Figure C.1 by shifting and rotating the information matrix and information vector. These maps were fused by simply adding them to previous posterior. All the information related to common landmarks are added using Equations 2.3 and 2.4 by exploiting the additive property of information state. [5].

CHAPTER 4: RESULTS

This research was divided into two parts: development of a multi-robot spawning tool and implementation of SEIF to test the working of a tool. This tool enabled standard ROS environment for the multi-robot applications without any modifications in the ROS directory.

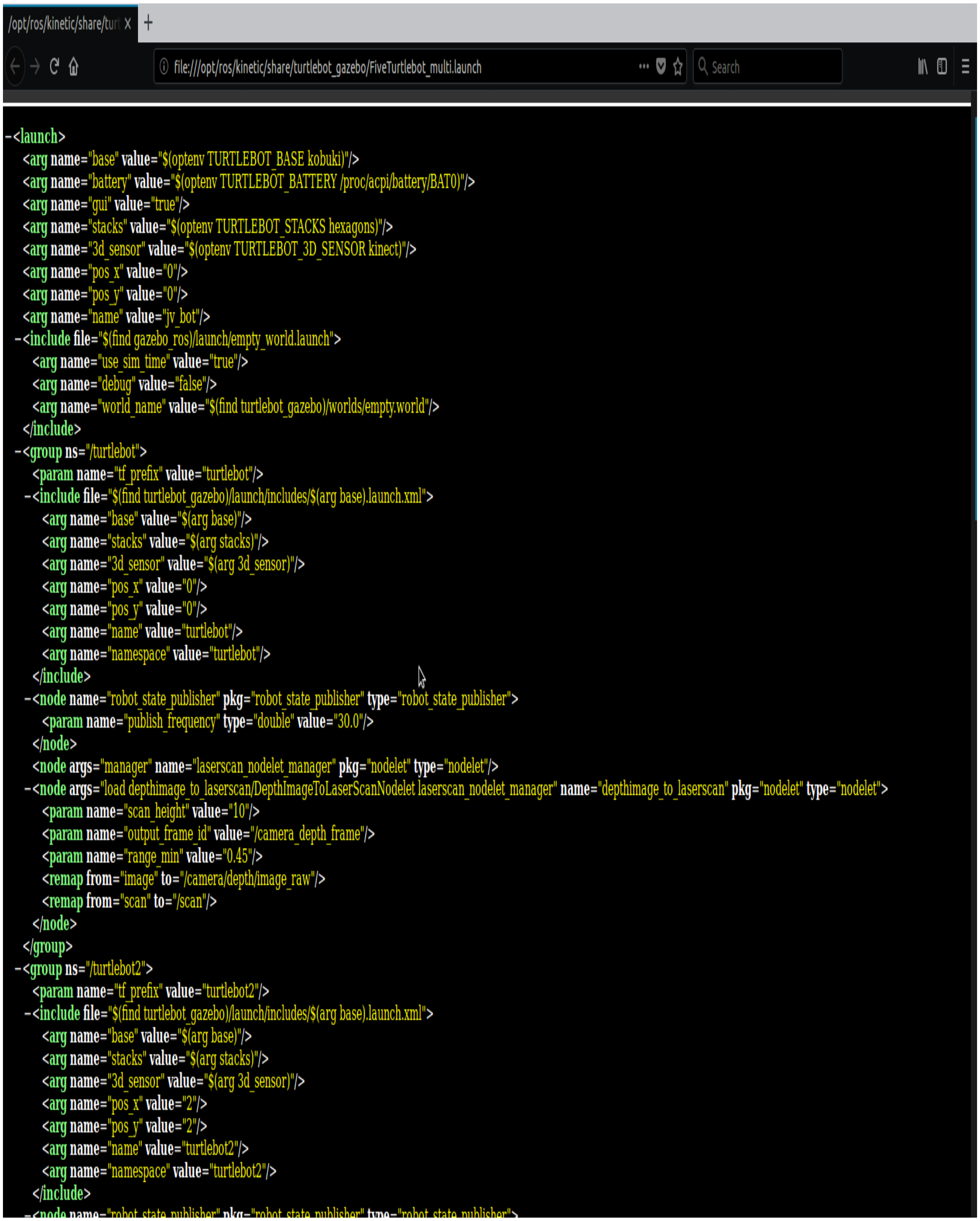
4.1 Multi TurtleBot Spawning Tool

During the implementation of the first part, a MATLAB script was written to generate an XML file. This file launched the Gazebo simulator with an environment with as many TurtleBots as an application needed. This tool was tested by spawning 5 different TurtleBots on the locations as Table 4.1 and communicating with each of them individually.

Table 4.1: Robots and their coordinates

Robot Name	Coordinates System	Position
Turtlebot	[x,y,z]	[0 0 0]
Turtlebot2	[x,y,z]	[2 2 0]
Turtlebot3	[x,y,z]	[-2 2 0]
Turtlebot4	[x,y,z]	[-2-2 0]
Turtlebot5	[x,y,z]	[2 -2 0]

This MATLAB script generated an XML launch file. A section of this file is depicted in the Figure 4.1 which lists the type of robot (Kobuki base), path to an empty world, and an information regarding turtlebot1. Figure A.1 and A.2 in the appendix A depicts the same information regarding turtlebot2, turtlebot3, turtlebot4 and turtlebot5. It has the name of TurtleBots, their respective locations along with a path to the urdf file.



/opt/ros/kinetic/share/turtlebot_gazebo/FiveTurtlebot_multi.launch


```
-<launch>
   <arg name="base" value="$(optenv TURTLEBOT_BASE kobuki)"/>
   <arg name="battery" value="$(optenv TURTLEBOT_BATTERY /proc/acpi/battery/BAT0)"/>
   <arg name="gui" value="true"/>
   <arg name="stacks" value="$(optenv TURTLEBOT_STACKS hexagons)"/>
   <arg name="3d_sensor" value="$(optenv TURTLEBOT_3D_SENSOR kinect)"/>
   <arg name="pos_x" value="0"/>
   <arg name="pos_y" value="0"/>
   <arg name="name" value="jv_bot"/>
  -<include file="$(find gazebo_ros)/launch/empty_world.launch">
     <arg name="use_sim_time" value="true"/>
     <arg name="debug" value="false"/>
     <arg name="world_name" value="$(find turtlebot_gazebo)/worlds/empty.world"/>
   </include>
  -<group ns="/turtlebot">
     <param name="tf_prefix" value="turtlebot"/>
    -<include file="$(find turtlebot_gazebo)/launch/includes/$(arg base).launch.xml">
       <arg name="base" value="$(arg base)"/>
       <arg name="stacks" value="$(arg stacks)"/>
       <arg name="3d_sensor" value="$(arg 3d_sensor)"/>
       <arg name="pos_x" value="0"/>
       <arg name="pos_y" value="0"/>
       <arg name="name" value="turtlebot"/>
       <arg name="namespace" value="turtlebot"/>
     </include>
    -<node name="robot_state_publisher" pkg="robot_state_publisher" type="robot_state_publisher">
       <param name="publish_frequency" type="double" value="30.0"/>
     </node>
     <node args="manager" name="laserscan_nodelet_manager" pkg="nodelet" type="nodelet"/>
    -<node args="load depthimage_to_laserscan/DepthImageToLaserScanNodelet laserscan_nodelet_manager" name="depthimage_to_laserscan" pkg="nodelet" type="nodelet">
       <param name="scan_height" value="10"/>
       <param name="output_frame_id" value="/camera_depth_frame"/>
       <param name="range_min" value="0.45"/>
       <remap from="image" to="/camera/depth/image_raw"/>
       <remap from="scan" to="/scan"/>
     </node>
   </group>
  -<group ns="/turtlebot2">
     <param name="tf_prefix" value="turtlebot2"/>
    -<include file="$(find turtlebot_gazebo)/launch/includes/$(arg base).launch.xml">
       <arg name="base" value="$(arg base)"/>
       <arg name="stacks" value="$(arg stacks)"/>
       <arg name="3d_sensor" value="$(arg 3d_sensor)"/>
       <arg name="pos_x" value="2"/>
       <arg name="pos_y" value="2"/>
       <arg name="name" value="turtlebot2"/>
       <arg name="namespace" value="turtlebot2"/>
     </include>
    -<node name="robot_state_publisher" pkg="robot_state_publisher" type="robot_state_publisher">
```


Figure 4.1: Section of A Launch File Generated

To launch a Gazebo simulator with an environment containing multi TurtleBots, 'roslauch' command is used along with the file name. Figure 4.2 depicts a Gazebo environment populated with five TurtleBots spawned on the given locations. The 'world' tab on the left side of the Figure 4.2 has a list of all the objects in the Gazebo environment. This environment comprises of a dumpster, a solid block, a ground plane, and five TurtleBots. A dumpster and a solid block were populated manually using the GUI of Gazebo.

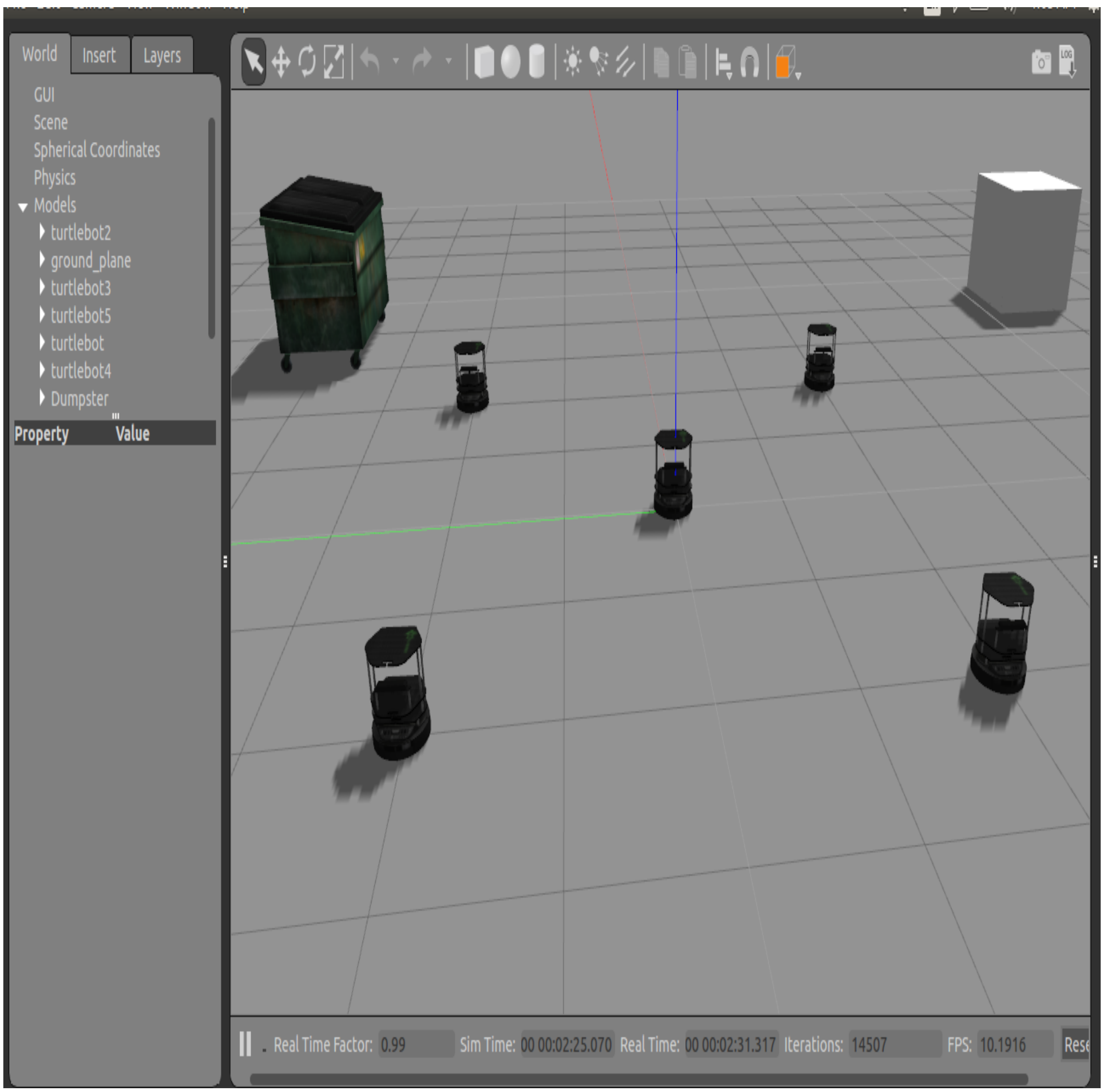


Figure 4.2: Five TurtleBots spawned in Gazebo using MATLAB script

Each individual TurtleBot can be communicated with topics associated with it. To operate each Turtlebot separately, topics should have a namespace according to the respective TurtleBot name. All the topics associated with the individual TurtleBot are shown in Figure 4.3, A.3 and A.4. Topics to be used for the 'turtlebot' are shown in Figure 4.3. Similarly, for turtlebot2 and turtlebot3 are shown in Figure A.3. Figure A.4 list all the topics related to turtlebot4 and turtlebot5.

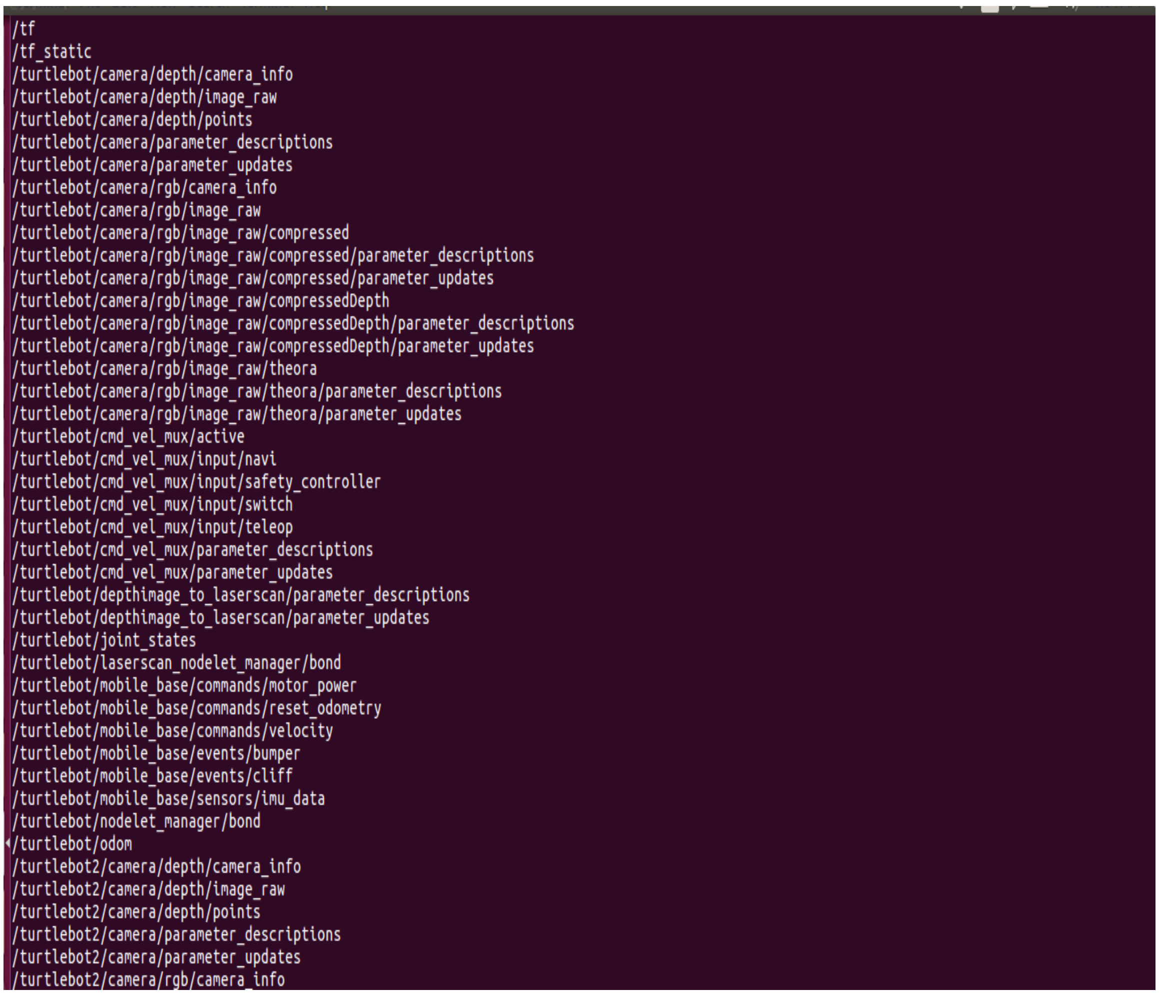


Figure 4.3: Namespacing for 'turtlebot'

This namespacing was used to control TurtleBots by publishing to the respective topic and taking camera data from the sensor by subscribing to the respective topic. Figure 4.4 shows two windows, each with processed data acquired from Kinect and visual perception of the robot.

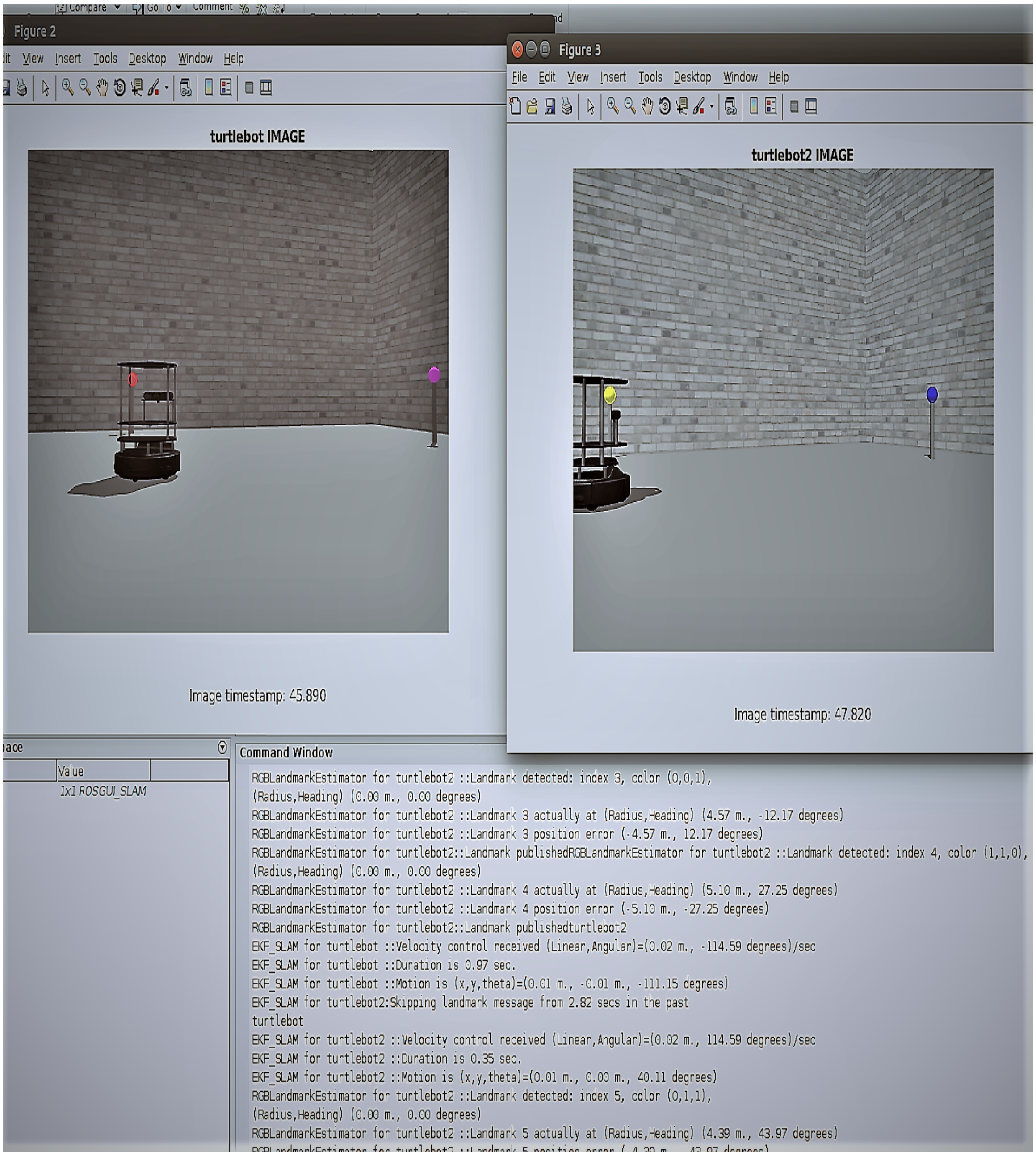


Figure 4.4: RGB camera visuals from both TurtleBots.

4.2 SLAM Using SEIF

During the second part of the research, the SEIF algorithm was implemented to test the built environment. SEIF was implemented in two cases: known initial pose and unknown initial pose. The final aim of the multi-robot system was to transfer the information from one TurtleBot to other and merge the map.

4.2.1 SEIF With Known Initial Position

When SEIF was implemented with known initial poses, a robot detected the landmarks at the positions with respect to the global coordinate system. Figure 4.5 depicts the result for this case. It can be inferred from the Figure 4.5 that each TurtleBot is implementing their own SEIF SLAM and building a map with respect to the global coordinate system. This process was important to check the functioning of a multi-robot tool in a multi-robot scenario because each TurtleBot observed the same landmarks and gave individual readings. This process helped to deduce the factors affecting the performance of a system such as time complexity.

In the Figure 4.5, starting positions of turtlebot2 and turtlebot1 are indicated by blue color diamonds at [2,0] and [-1,0] respectively. A line of red dots represents the trajectory of the TurtleBot. Gray arrows embedded by the gray circles represent the actual positions of the TurtleBots and sky blue arrows with a sky blue circles represent the predicted value of the robot states through the filter. Actual positions of the landmarks are indicated by six small circles in the environment with their respective unique colors. Arrows embedded by a circle near those landmarks represent the predicted values from each TurtleBot. The unique color of that landmark is represented by the color of an arrow at the predicted value, and color of the outer circle differentiates the predictions by the TurtleBot. All the predictions by turtlebot2 and turtlebot1 are represented by green and black circle respectively.

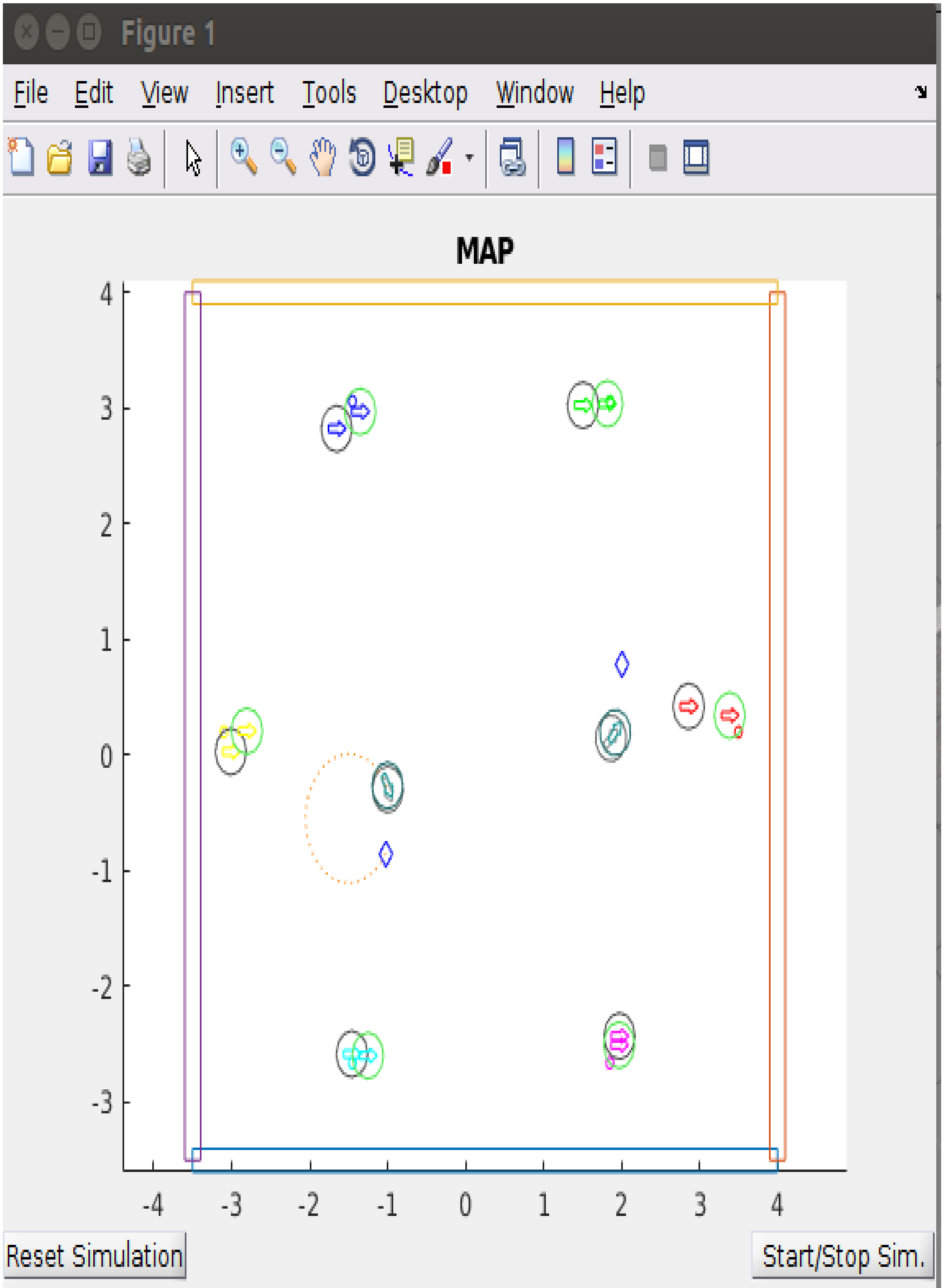


Figure 4.5: SEIF implemented on individual TurtleBots with known poses

Table 4.2 contains all the measurement values of the landmarks for both TurtleBots. It can be seen that, there is a difference between both the values which are Dependant on factors such as computation efficiency and simulation speed.

Table 4.2: Landmark prediction for TurtleBots

Landmark	Position	Turtlebot1	turtlebot2
Landmark1	[3.5,0.2]	[3.2,0.3]	[3.4,0.2]
Landmark2	[1.8,3.0]	[1.7,3]	[1.8,3]
Landmark3	[-1.4,3.0]	[-1.4,3]	[-1.6,2.28]
Landmark4	[-3.1,0.2]	[-2.85,0.2]	[-3,0.1]
Landmark5	[-1.4,-2.6]	[-1.4,-2.6]	[-1.4,-2.6]
Landmark6	[1.8,-2.6]	[1.8,-2.6]	[1.8,-2.6]

4.2.2 Update Time Per Measurement

Theoretically, EKF has quadratic time complexity compared to SEIF. The computational complexity of EKF tends to increase quadratically with the number of landmarks. Whereas, SEIF had a constant computational complexity. It can be inferred from Table 4.3 that for a small number of landmarks (6 for the implementation of this research) SEIF showed minute differences in the time required per update.

Table 4.3: Landmark prediction for TurtleBots

Landmark	Turtlebot1	turtlebot2
Landmark1	0.000000000000000000000000 sec	0.000000000000000000000000 sec
Landmark2	0.010000000000000000208167 sec	0.010000000000000000208167 sec
Landmark3	0.000000000000000000000000 sec	0.010000000000000000208167 sec
Landmark4	0.030000000000000002359224 sec	0.000000000000000000000000 sec
Landmark5	0.010000000000000000208167 sec	0.010000000000000000208167 sec
Landmark6	0.030000000000000002359224	0.010000000000000000208167 sec

4.2.3 SEIF With Unknown Initial Position

Figure 4.6 depicts the SEIF with unknown initial pose. When SEIF with unknown initial poses was implemented, each TurtleBot started mapping with respect to its own local reference frame. Its measurements are according to odometry readings and the processed image from the RGB camera. Color coding used in the Figure 4.6 is same as the case with known initial positions i.e arrow colors are landmarks colors along with the predicted locations, all the predictions by turtlebot2 and turtlebot1 are represented by green and black circle respectively. Obstacles put in the environment blocked some part of the environment for both the TurtleBots. As it can be inferred from the Figure 4.6 that turtlebot1 could see only landmark 1,3,4,5 and turtlebot2 could see landmark numbers 1,2,6.

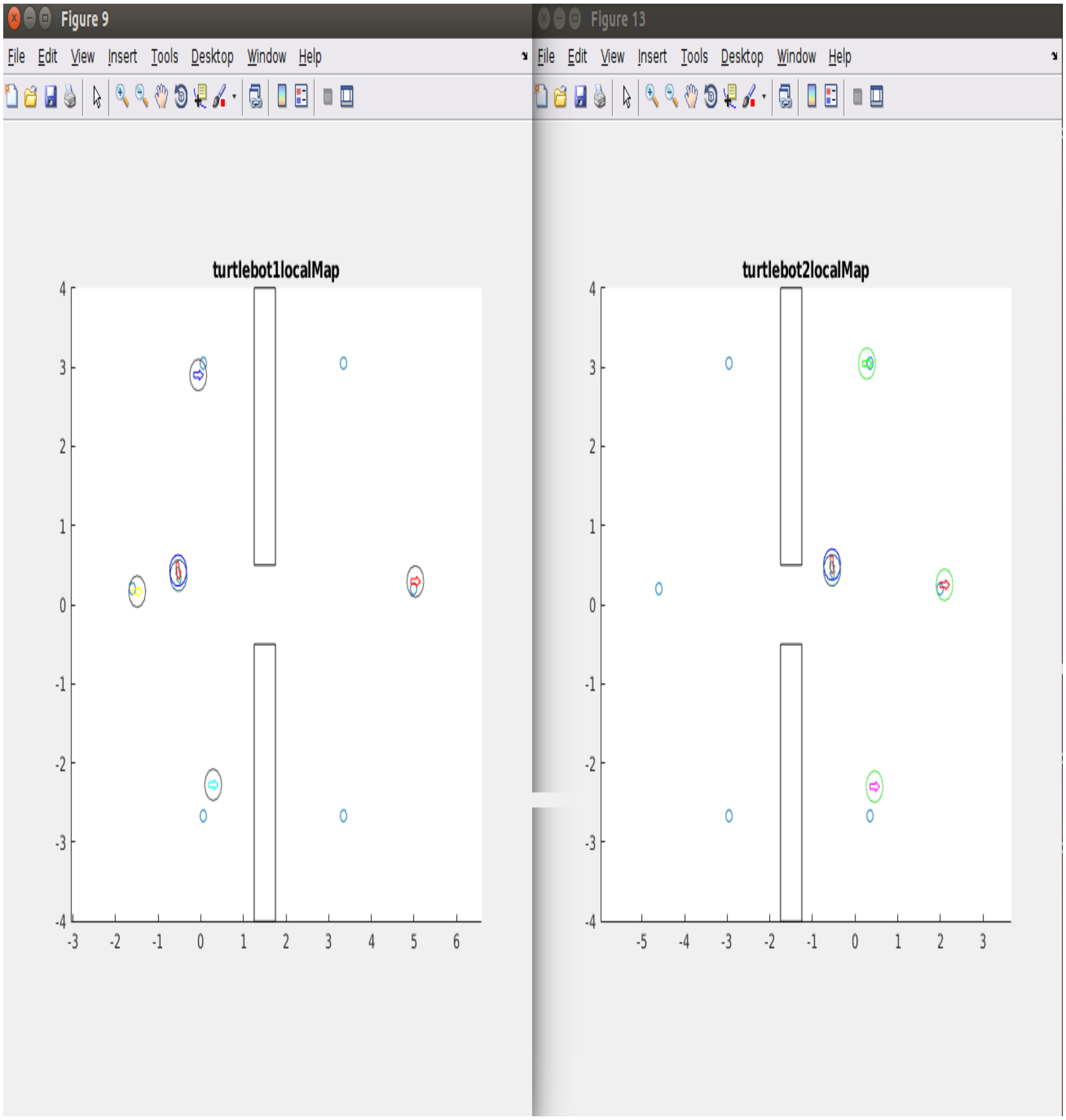


Figure 4.6: SEIF implemented on individual TurtleBots with unknown poses

4.2.4 Map Merging

For the above scenario, each TurtleBot maintains a posterior for observed landmarks. When turtlebot1 shared its posterior with turtlebot2, turtlebot2 aligned and fused it with its own posterior. It can be seen that before map merging that turtlebot2 was able to map 1,2, and 6 but map merging algorithm enabled it to map all the remaining landmarks. Figure 4.7 depicts the merged maps through the map merging algorithm.

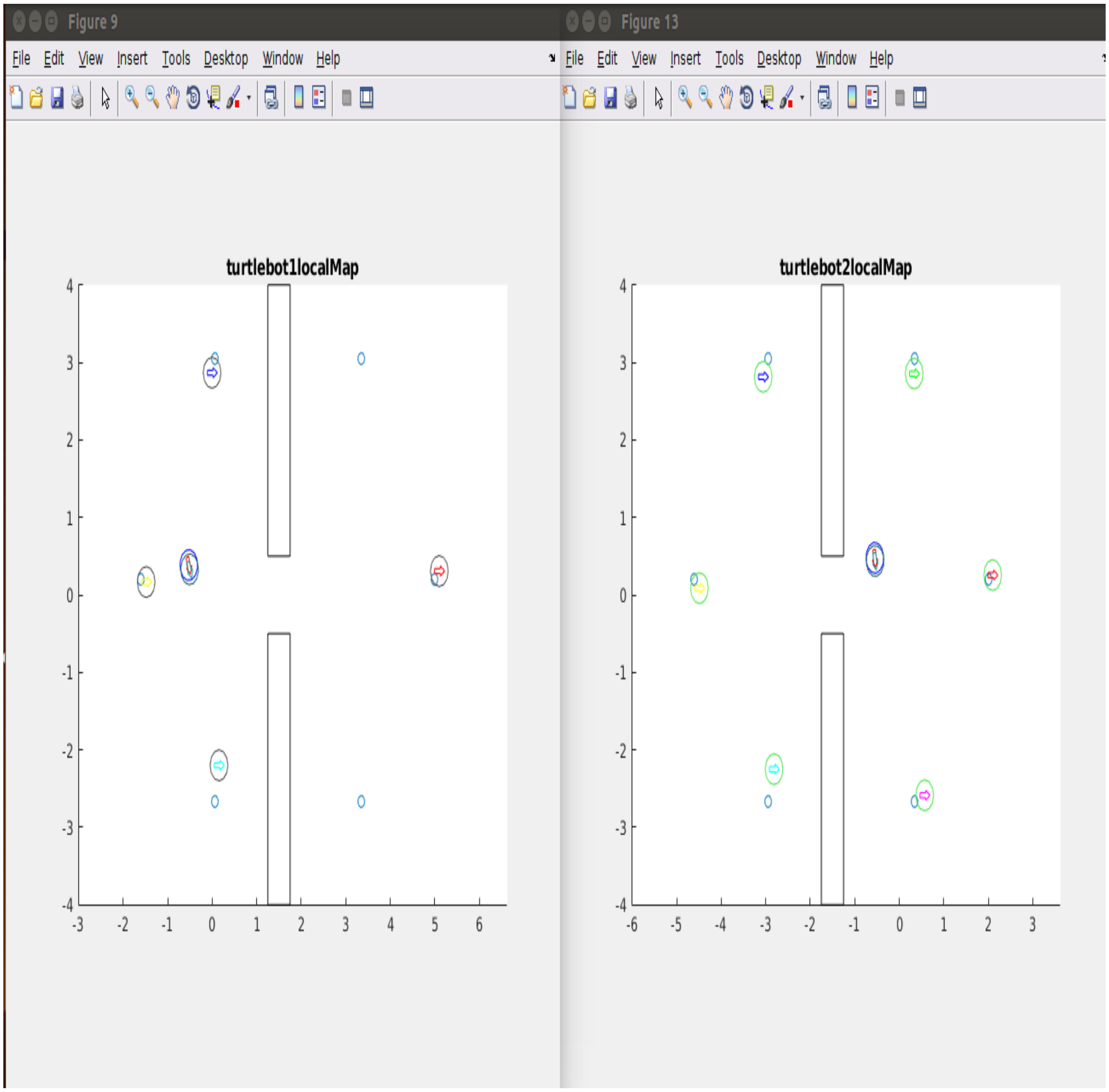


Figure 4.7: Decentralized MR-SLAM with map merging

CHAPTER 5: CONCLUSIONS AND FUTURE WORK

This chapter concludes the research carried out so far and state the future work for this research. Literature regarding the MR-SLAM and SLAM was read to understand the current technologies and to analyze the current state-of-the-art solution. Many available solutions for MR-SLAM can be categorized in centralized, decentralized and distributed systems. Implementation of centralized systems often requires an external agent or server to monitor the processing of all the present robots, which makes it the most complex system compared to others. In a centralized system the number of robots present in the environment are restricted due to the limitations on the communication bandwidth and processing power. Decentralized systems are comparatively more robust than centralized systems. Processing and controlling power is divided amongst a few robots in a team. many restrictions regarding the number of the robots and communication bandwidths are loose for it. Distributed systems are more efficient and robust where each TurtleBot is responsible for processing the data so any numbers of TurtleBot can be added without increasing the complexity of the system.

5.1 Conclusion

Many state-of-the-art solutions include the use of recursive estimators such as EKF and RBPF where they are required to localize the other TurtleBot in its own posterior. This process may include an identification of the TurtleBots while navigating in the environment. This research was focused on developing a tool which will enable Gazebo for multi-robot simulations. This tool was further tested with the implementation of a decentralized system with two TurtleBots. A lack of multi-robot simulator patch on

Gazebo for TurtleBot was noticed and a tool was developed which will take arguments such as robot names and their coordinates from the user. It then generates a '.launch' file which can be used to spawn multiple robots in the Gazebo environment. Its ability to spawn as many robots as a user wants to, make it an outstanding solution to the current multi-robot simulator problem.

In an implementation of SEIF, two TurtleBots shared their posterior in the form of information vectors and information matrices. When compared to EKF, SEIF is more computationally efficient. Every update in EKF iterates through an entire posterior and its computational complexity is quadratic in time with respect to the size of the map. SEIF, on the other hand, has constant computational complexity irrespective of the size of the map. Similarly, for the memory requirements, EKF memory requirements grow quadratically with respect to map size whereas in SEIF memory requirements grow linearly with the size of the map. SEIF maintains sparseness to regulate the update time at any point in the algorithm. This sparseness causes SEIF to approximate the recovery of the states of robot and landmarks. Due to this factor, EKF tends to generate more accurate results than SEIF. Figure D.2 and Figure D.1 depicted the effects of sparsification.

5.2 Future Work

Future work involves working with the multi-robot tool to the next level. A common GUI which takes names and coordinates as an input can be extended to a more complex and functionally advanced GUI with preemptive error findings. The inability of the current tool to detect the TurtleBots with the same name and same coordinates can be used as a motivation to extend this tool to detect those errors. The current research regarding the SEIF only considers the case with known correspondences and a limited number of landmarks. This research can be extended to a more complex algorithm to identify a landmark whose correspondence is unknown. Use of simulators put many constraints on factors such as communication, processing time and processing requirements during an implementation and analysis of the multi-robot system. Assumptions made during the implementations such as replacing network communication with messages and implementing object-oriented programming to simulate different system can be studied in a great depth during an implantation on the actual TurtleBots. Since there are many applications for multi-robot systems and current research is based on the simulation, there is an opportunity for improvement.

REFERENCES

[1] S. Thrun and Y. Liu, "Multi-robot SLAM with sparse extended information filers," in *Robotics Research. The Eleventh International Symposium*, pp. 254–266, Springer, 2005.

[2] M. Pfingsthorn, B. Slamet, and A. Visser, "A scalable hybrid multi-robot SLAM method for highly detailed maps," in *Robot Soccer World Cup*, pp. 457–464, Springer, 2007.

[3] X. S. Zhou and S. I. Roumeliotis, "Multi-robot SLAM with unknown initial correspondence: The robot rendezvous case," in *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*, pp. 1785–1792, IEEE, 2006.

[4] I. Open Source Robotics Foundation, "TurtleBot2 open-source robot development kit for apps on wheels." `"https://www.turtlebot.com/turtlebot2`.

[5] S. Thrun, W. Burgard, and D. Fox, *Probabilistic robotics.* MIT press, 2005.

[6] J. Kshirsagar, S. Shue, and J. Conrad, "Survey of multiple robot simultaneous localization and mapping," *IEEE SoutheastCon 2018*, 2018.

[7] L. Carlone, M. K. Ng, J. Du, B. Bona, and M. Indri, "Rao-Blackwellized particle filters multi robot SLAM with unknown initial correspondences and limited communication," in *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pp. 243–249, IEEE, 2010.

[8] L. Carlone, M. K. Ng, J. Du, B. Bona, and M. Indri, "Simultaneous localization and mapping using Rao-Blackwellized particle filters in multi robot systems," *Journal of Intelligent & Robotic Systems*, vol. 63, no. 2, pp. 283–307, 2011.

[9] S. B. Williams, G. Dissanayake, and H. Durrant-Whyte, "Towards multi-vehicle simultaneous localisation and mapping," in *Robotics and Automation, 2002. Proceedings. ICRA'02. IEEE International Conference on*, vol. 3, pp. 2743–2748, IEEE, 2002.

[10] S. Thrun, "A probabilistic on-line mapping algorithm for teams of mobile robots," *The International Journal of Robotics Research*, vol. 20, no. 5, pp. 335–363, 2001.

[11] A. Howard, "Multi-robot simultaneous localization and mapping using particle filters," *The International Journal of Robotics Research*, vol. 25, no. 12, pp. 1243–1256, 2006.

[12] A. Howard, G. S. Sukhatme, and M. J. Mataric, "Multirobot simultaneous localization and mapping using manifold representations," *Proceedings of the IEEE*, vol. 94, no. 7, pp. 1360–1369, 2006.

[13] D. Fox, J. Ko, K. Konolige, and B. Stewart, "A hierarchical Bayesian approach to the revisiting problem in mobile robot map building," in *Robotics Research. The Eleventh International Symposium*, pp. 60–69, Springer, 2005.

[14] J. Neira and J. D. Tardós, "Data association in stochastic mapping using the joint compatibility test," *IEEE Transactions on robotics and automation*, vol. 17, no. 6, pp. 890–897, 2001.

[15] . O. S. R. Foundation, "Gazebo ROS overview." "http://gazebosim.org/tutorials?tut=ros_overview.

[16] O. S. R. Foundation, "About ROS." "http://www.ros.org/about-ros/.

[17] S. Shue and J. M. Conrad, "Reducing the effect of signal multipath fading in RSSI-distance estimation using Kalman filters," in *Proceedings of the 19th Communications & Networking Symposium*, p. 5, Society for Computer Simulation International, 2016.

[18] A. A. Panchpor, *Implementation of Path Planning Algorithms on a Mobile Robot in Dynamic Indoor Environments*. PhD thesis, The University of North Carolina at Charlotte, 2018.

[19] G. Grisetti, G. D. Tipaldi, C. Stachniss, W. Burgard, and D. Nardi, "Fast and accurate SLAM with rao–blackwellized particle filters," *Robotics and Autonomous Systems*, vol. 55, no. 1, pp. 30–38, 2007.

[20] H. J. Chang, C. G. Lee, Y. C. Hu, and Y.-H. Lu, "Multi-robot SLAM with topological/metric maps," in *Intelligent Robots and Systems, 2007. IROS 2007. IEEE/RSJ International Conference on*, pp. 1467–1472, IEEE, 2007.

[21] A. Eliazar and R. Parr, "Dp-SLAM: Fast, robust simultaneous localization and mapping without predetermined landmarks," in *IJCAI*, vol. 3, pp. 1135–1142, 2003.

[22] J. Ko, B. Stewart, D. Fox, K. Konolige, and B. Limketkai, "A practical, decision-theoretic approach to multi-robot mapping and exploration," in *Intelligent Robots and Systems, 2003.(IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on*, vol. 4, pp. 3232–3238, IEEE, 2003.

[23] R. Vincent, D. Fox, J. Ko, K. Konolige, B. Limketkai, B. Morisset, C. Ortiz, D. Schulz, and B. Stewart, "Distributed multirobot exploration, mapping, and task allocation," *Annals of Mathematics and Artificial Intelligence*, vol. 52, no. 2-4, pp. 229–255, 2008.

[24] J. W. Fenwick, P. M. Newman, and J. J. Leonard, "Cooperative concurrent mapping and localization," in *Robotics and Automation, 2002. Proceedings. ICRA'02. IEEE International Conference on*, vol. 2, pp. 1810–1817, IEEE, 2002.

[25] S. I. Roumeliotis and G. A. Bekey, "Bayesian estimation and Kalman filtering: A unified framework for mobile robot localization," in *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*, vol. 3, pp. 2985–2992, IEEE, 2000.

[26] S. I. Roumeliotis and G. A. Bekey, "Distributed multirobot localization," *IEEE Transactions on Robotics and Automation*, vol. 18, no. 5, pp. 781–795, 2002.

[27] H. Durrant-Whyte and T. Bailey, "Simultaneous localization and mapping: part i," *IEEE robotics & automation magazine*, vol. 13, no. 2, pp. 99–110, 2006.

[28] T. Bailey and H. Durrant-Whyte, "Simultaneous localization and mapping (SLAM): Part ii," *IEEE Robotics & Automation Magazine*, vol. 13, no. 3, pp. 108–117, 2006.

[29] R. Simmons, D. Apfelbaum, W. Burgard, D. Fox, M. Moors, S. Thrun, and H. Younes, "Coordination for multi-robot exploration and mapping," in *Aaai/Iaai*, pp. 852–858, 2000.

[30] L. A. Andersson and J. Nygards, "C-sam: Multi-robot SLAM using square root information smoothing," in *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, pp. 2798–2805, IEEE, 2008.

[31] R. J. Alitappeh, G. A. Pereira, A. R. Araújo, and L. C. Pimenta, "Multi-robot deployment using topological maps," *Journal of Intelligent & Robotic Systems*, vol. 86, no. 3-4, pp. 641–661, 2017.

[32] S. Saeedi, L. Paull, M. Trentini, and H. Li, "Occupancy grid map merging for multiple robot simultaneous localization and mapping," *International Journal of Robotics and Automation*, vol. 30, no. 2, pp. 149–157, 2015.

[33] J. Jessup, S. N. Givigi, and A. Beaulieu, "Robust and efficient multirobot 3-D mapping merging with octree-based occupancy grids," *IEEE Systems Journal*, vol. 11, no. 3, pp. 1723–1732, 2017.

[34] S. S. Kia, S. F. Rounds, and S. Martínez, "A centralized-equivalent decentralized implementation of extended Kalman filters for cooperative localization," in *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*, pp. 3761–3766, IEEE, 2014.

[35] X. Chen, H. Lu, J. Xiao, H. Zhang, and P. Wang, "Robust relocalization based on active loop closure for real-time monocular SLAM," in *International Conference on Computer Vision Systems*, pp. 131–143, Springer, 2017.

[36] A. Torres-González, J. R. Martinez-de Dios, and A. Ollero, "Range-only SLAM for robot-sensor network cooperation," *Autonomous Robots*, vol. 42, no. 3, pp. 649–663, 2018.

[37] S. L. Bowman, N. Atanasov, K. Daniilidis, and G. J. Pappas, "Probabilistic data association for semantic SLAM," in *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pp. 1722–1729, IEEE, 2017.

```
/opt/ros/kinetic/share/turt X  +
file:///opt/ros/kinetic/share/turtlebot_gazebo/FiveTurtlebot_multi.launch
Search
    <remap from="scan" to="/scan"/>
  </node>
</group>
-<group ns="/turtlebot2">
  <param name="tf prefix" value="turtlebot2"/>
 -<include file="$(find turtlebot gazebo)/launch/includes/$(arg base).launch.xml">
    <arg name="base" value="$(arg base)"/>
    <arg name="stacks" value="$(arg stacks)"/>
    <arg name="3d sensor" value="$(arg 3d sensor)"/>
    <arg name="pos x" value="2"/>
    <arg name="pos y" value="2"/>
    <arg name="name" value="turtlebot2"/>
    <arg name="namespace" value="turtlebot2"/>
  </include>
 -<node name="robot state publisher" pkg="robot state publisher" type="robot state publisher">
    <param name="publish frequency" type="double" value="30.0"/>
  </node>
  <node args="manager" name="laserscan nodelet manager" pkg="nodelet" type="nodelet"/>
 -<node args="load depthimage to laserscan/DepthImageToLaserScanNodelet laserscan nodelet manager" name="depthimage to laserscan" pkg="nodelet" type="nodelet">
    <param name="scan height" value="10"/>
    <param name="output frame id" value="/camera depth frame"/>
    <param name="range min" value="0.45"/>
    <remap from="image" to="/camera/depth/image raw"/>
    <remap from="scan" to="/scan"/>
  </node>
</group>
-<group ns="/turtlebot3">
  <param name="tf prefix" value="turtlebot3"/>
 -<include file="$(find turtlebot gazebo)/launch/includes/$(arg base).launch.xml">
    <arg name="base" value="$(arg base)"/>
    <arg name="stacks" value="$(arg stacks)"/>
    <arg name="3d sensor" value="$(arg 3d sensor)"/>
    <arg name="pos x" value="-2"/>
    <arg name="pos y" value="2"/>
    <arg name="name" value="turtlebot3"/>
    <arg name="namespace" value="turtlebot3"/>
  </include>
 -<node name="robot state publisher" pkg="robot state publisher" type="robot state publisher">
    <param name="publish frequency" type="double" value="30.0"/>
  </node>
  <node args="manager" name="laserscan nodelet manager" pkg="nodelet" type="nodelet"/>
 -<node args="load depthimage to laserscan/DepthImageToLaserScanNodelet laserscan nodelet manager" name="depthimage to laserscan" pkg="nodelet" type="nodelet">
    <param name="scan height" value="10"/>
    <param name="output frame id" value="/camera depth frame"/>
    <param name="range min" value="0.45"/>
    <remap from="image" to="/camera/depth/image raw"/>
    <remap from="scan" to="/scan"/>
  </node>
</group>
-<group ns="/turtlebot4">
  <param name="tf prefix" value="turtlebot4"/>
```

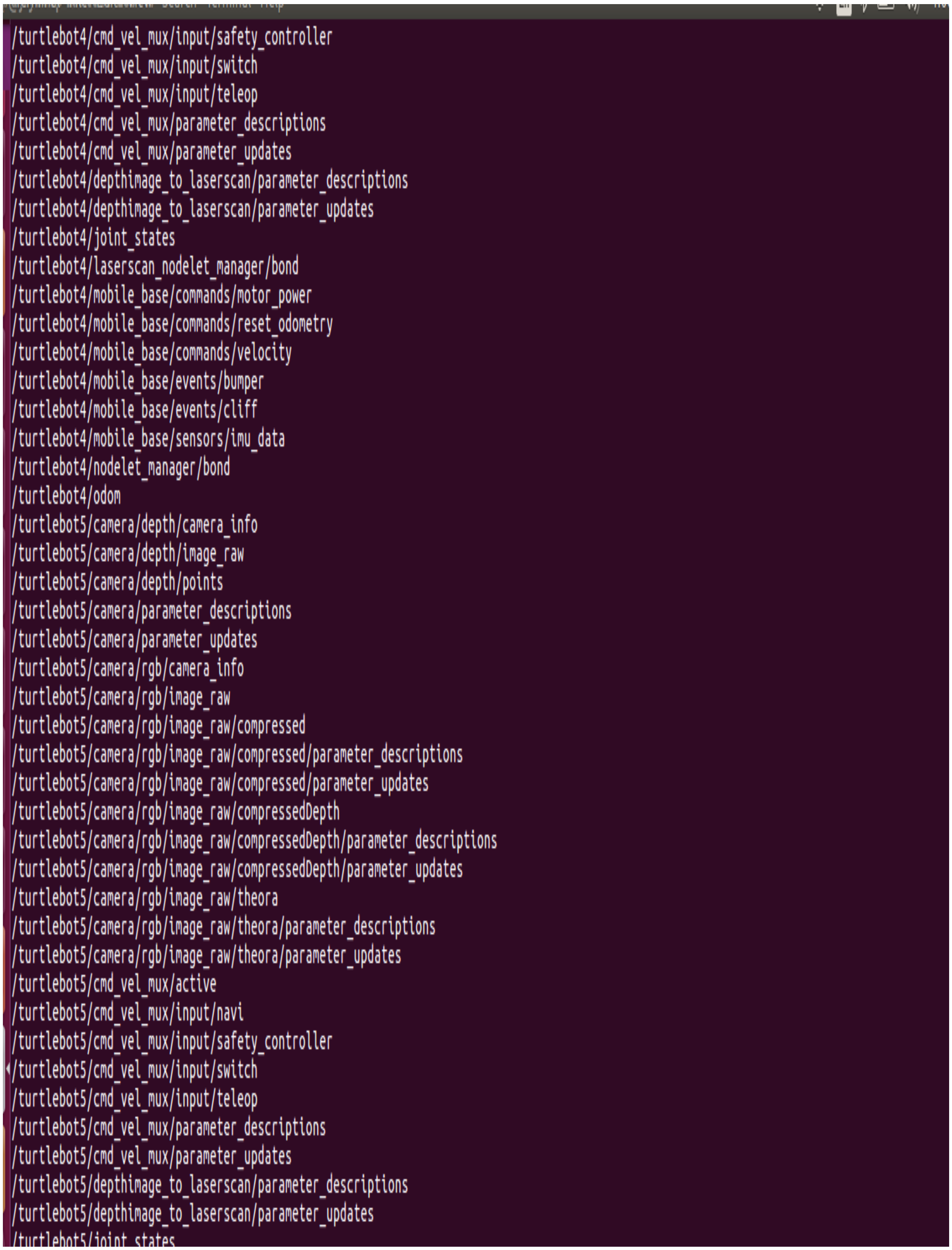
Figure A.1: Launch File Generated - section 2


```
      <remap from="scan" to="/scan"/>
    </node>
  </group>
-<group ns="/turtlebot4">
    <param name="tf_prefix" value="turtlebot4"/>
  -<include file="$(find turtlebot_gazebo)/launch/includes/$(arg base).launch.xml">
      <arg name="base" value="$(arg base)"/>
      <arg name="stacks" value="$(arg stacks)"/>
      <arg name="3d_sensor" value="$(arg 3d_sensor)"/>
      <arg name="pos_x" value="-2"/>
      <arg name="pos_y" value="-2"/>
      <arg name="name" value="turtlebot4"/>
      <arg name="namespace" value="turtlebot4"/>
    </include>
  -<node name="robot_state_publisher" pkg="robot_state_publisher" type="robot_state_publisher">
      <param name="publish_frequency" type="double" value="30.0"/>
    </node>
    <node args="manager" name="laserscan_nodelet_manager" pkg="nodelet" type="nodelet"/>
  -<node args="load depthimage_to_laserscan/DepthImageToLaserScanNodelet laserscan_nodelet_manager" name="depthimage_to_laserscan" pkg="nodelet" type="nodelet">
      <param name="scan_height" value="10"/>
      <param name="output_frame_id" value="/camera_depth_frame"/>
      <param name="range_min" value="0.45"/>
      <remap from="image" to="/camera/depth/image_raw"/>
      <remap from="scan" to="/scan"/>
    </node>
  </group>
-<group ns="/turtlebot5">
    <param name="tf_prefix" value="turtlebot5"/>
  -<include file="$(find turtlebot_gazebo)/launch/includes/$(arg base).launch.xml">
      <arg name="base" value="$(arg base)"/>
      <arg name="stacks" value="$(arg stacks)"/>
      <arg name="3d_sensor" value="$(arg 3d_sensor)"/>
      <arg name="pos_x" value="2"/>
      <arg name="pos_y" value="-2"/>
      <arg name="name" value="turtlebot5"/>
      <arg name="namespace" value="turtlebot5"/>
    </include>
  -<node name="robot_state_publisher" pkg="robot_state_publisher" type="robot_state_publisher">
      <param name="publish_frequency" type="double" value="30.0"/>
    </node>
    <node args="manager" name="laserscan_nodelet_manager" pkg="nodelet" type="nodelet"/>
  -<node args="load depthimage_to_laserscan/DepthImageToLaserScanNodelet laserscan_nodelet_manager" name="depthimage_to_laserscan" pkg="nodelet" type="nodelet">
      <param name="scan_height" value="10"/>
      <param name="output_frame_id" value="/camera_depth_frame"/>
      <param name="range_min" value="0.45"/>
      <remap from="image" to="/camera/depth/image_raw"/>
      <remap from="scan" to="/scan"/>
    </node>
  </group>
</launch>
```

Figure A.2: Launch File Generated - section 3

```
/turtlebot2/depthimage_to_laserscan/parameter_descriptions
/turtlebot2/depthimage_to_laserscan/parameter_updates
/turtlebot2/joint_states
/turtlebot2/laserscan_nodelet_manager/bond
/turtlebot2/mobile_base/commands/motor_power
/turtlebot2/mobile_base/commands/reset_odometry
/turtlebot2/mobile_base/commands/velocity
/turtlebot2/mobile_base/events/bumper
/turtlebot2/mobile_base/events/cliff
/turtlebot2/mobile_base/sensors/imu_data
/turtlebot2/nodelet_manager/bond
/turtlebot2/odom
/turtlebot3/camera/depth/camera_info
/turtlebot3/camera/depth/image_raw
/turtlebot3/camera/depth/points
/turtlebot3/camera/parameter_descriptions
/turtlebot3/camera/parameter_updates
/turtlebot3/camera/rgb/camera_info
/turtlebot3/camera/rgb/image_raw
/turtlebot3/camera/rgb/image_raw/compressed
/turtlebot3/camera/rgb/image_raw/compressed/parameter_descriptions
/turtlebot3/camera/rgb/image_raw/compressed/parameter_updates
/turtlebot3/camera/rgb/image_raw/compressedDepth
/turtlebot3/camera/rgb/image_raw/compressedDepth/parameter_descriptions
/turtlebot3/camera/rgb/image_raw/compressedDepth/parameter_updates
/turtlebot3/camera/rgb/image_raw/theora
/turtlebot3/camera/rgb/image_raw/theora/parameter_descriptions
/turtlebot3/camera/rgb/image_raw/theora/parameter_updates
/turtlebot3/cmd_vel_mux/active
/turtlebot3/cmd_vel_mux/input/navi
/turtlebot3/cmd_vel_mux/input/safety_controller
/turtlebot3/cmd_vel_mux/input/switch
/turtlebot3/cmd_vel_mux/input/teleop
/turtlebot3/cmd_vel_mux/parameter_descriptions
/turtlebot3/cmd_vel_mux/parameter_updates
/turtlebot3/depthimage_to_laserscan/parameter_descriptions
/turtlebot3/depthimage_to_laserscan/parameter_updates
/turtlebot3/joint_states
/turtlebot3/laserscan_nodelet_manager/bond
/turtlebot3/mobile_base/commands/motor_power
/turtlebot3/mobile_base/commands/reset_odometry
/turtlebot3/mobile_base/commands/velocity
/turtlebot3/mobile_base/events/bumper
```

Figure A.3: Namespacing for turtlebot2 and turtlebot3



```
/turtlebot4/cmd_vel_mux/input/safety_controller
/turtlebot4/cmd_vel_mux/input/switch
/turtlebot4/cmd_vel_mux/input/teleop
/turtlebot4/cmd_vel_mux/parameter_descriptions
/turtlebot4/cmd_vel_mux/parameter_updates
/turtlebot4/depthimage_to_laserscan/parameter_descriptions
/turtlebot4/depthimage_to_laserscan/parameter_updates
/turtlebot4/joint_states
/turtlebot4/laserscan_nodelet_manager/bond
/turtlebot4/mobile_base/commands/motor_power
/turtlebot4/mobile_base/commands/reset_odometry
/turtlebot4/mobile_base/commands/velocity
/turtlebot4/mobile_base/events/bumper
/turtlebot4/mobile_base/events/cliff
/turtlebot4/mobile_base/sensors/imu_data
/turtlebot4/nodelet_manager/bond
/turtlebot4/odom
/turtlebot5/camera/depth/camera_info
/turtlebot5/camera/depth/image_raw
/turtlebot5/camera/depth/points
/turtlebot5/camera/parameter_descriptions
/turtlebot5/camera/parameter_updates
/turtlebot5/camera/rgb/camera_info
/turtlebot5/camera/rgb/image_raw
/turtlebot5/camera/rgb/image_raw/compressed
/turtlebot5/camera/rgb/image_raw/compressed/parameter_descriptions
/turtlebot5/camera/rgb/image_raw/compressed/parameter_updates
/turtlebot5/camera/rgb/image_raw/compressedDepth
/turtlebot5/camera/rgb/image_raw/compressedDepth/parameter_descriptions
/turtlebot5/camera/rgb/image_raw/compressedDepth/parameter_updates
/turtlebot5/camera/rgb/image_raw/theora
/turtlebot5/camera/rgb/image_raw/theora/parameter_descriptions
/turtlebot5/camera/rgb/image_raw/theora/parameter_updates
/turtlebot5/cmd_vel_mux/active
/turtlebot5/cmd_vel_mux/input/navi
/turtlebot5/cmd_vel_mux/input/safety_controller
/turtlebot5/cmd_vel_mux/input/switch
/turtlebot5/cmd_vel_mux/input/teleop
/turtlebot5/cmd_vel_mux/parameter_descriptions
/turtlebot5/cmd_vel_mux/parameter_updates
/turtlebot5/depthimage_to_laserscan/parameter_descriptions
/turtlebot5/depthimage_to_laserscan/parameter_updates
/turtlebot5/joint_states
```

Figure A.4: Namespacing for turtlebot4 and turtlebot5

APPENDIX B: SEIF Algorithm

Algorithm SEIF_measurement_update($\bar{\xi}_t, \bar{\Omega}_t, \mu_t, z_t, c_t$):

$$Q_t = \begin{pmatrix} \sigma_r & 0 & 0 \\ 0 & \sigma_\phi & 0 \\ 0 & 0 & \sigma_s \end{pmatrix}$$

for all observed features $z_t^i = (r_t^i \;\; \phi_t^i \;\; s_t^i)^T$ *do*

$j = c_t^i$

if landmark j *never seen before*

$$\begin{pmatrix} \mu_{j,x} \\ \mu_{j,y} \\ \mu_{j,s} \end{pmatrix} = \begin{pmatrix} \mu_{t,x} \\ \mu_{t,y} \\ s_t^i \end{pmatrix} + r_t^i \begin{pmatrix} \cos(\phi_t^i + \mu_{t,\theta}) \\ \sin(\phi_t^i + \mu_{t,\theta}) \\ 0 \end{pmatrix}$$

endif

$$\delta = \begin{pmatrix} \delta_x \\ \delta_y \end{pmatrix} = \begin{pmatrix} \mu_{j,x} - \mu_{t,x} \\ \mu_{j,y} - \mu_{t,y} \end{pmatrix}$$

$q = \delta^T \delta$

$$\hat{z}_t^i = \begin{pmatrix} \sqrt{q} \\ \operatorname{atan2}(\delta_y, \delta_x) - \mu_{t,\theta} \\ \mu_{j,s} \end{pmatrix}$$

$$H_t^i = \frac{1}{q} \begin{pmatrix} \sqrt{q}\delta_x & -\sqrt{q}\delta_y & 0 & 0\cdots0 & -\sqrt{q}\delta_x & \sqrt{q}\delta_y & 0 & 0\cdots0 \\ \delta_y & \delta_x & -1 & 0\cdots0 & -\delta_y & -\delta_x & 0 & 0\cdots0 \\ 0 & 0 & 0 & \underbrace{0\cdots0}_{3j-3} & 0 & 0 & 1 & \underbrace{0\cdots0}_{3j} \end{pmatrix}$$

endfor

$\xi_t = \bar{\xi}_t + \sum_i H_t^{iT} Q_t^{-1} [z_t^i - \hat{z}_t^i - H_t^i \; \mu_t]$

$\Omega_t = \bar{\Omega}_t + \sum_i H_t^{iT} Q_t^{-1} H_t^i$

return ξ_t, Ω_t

Figure B.1: Measurement update algorithm [5]

1: **Algorithm SEIF_motion_update**($\xi_{t-1}, \Omega_{t-1}, \mu_{t-1}, u_t$):

2: $$F_x = \begin{pmatrix} 1 & 0 & 0 & 0 \cdots 0 \\ 0 & 1 & 0 & 0 \cdots 0 \\ 0 & 0 & 1 & \underbrace{0 \cdots 0}_{2N} \end{pmatrix}$$

3: $$\delta = \begin{pmatrix} -\frac{v_t}{\omega_t} \sin \mu_{t-1,\theta} + \frac{v_t}{\omega_t} \sin(\mu_{t-1,\theta} + \omega_t \Delta t) \\ \frac{v_t}{\omega_t} \cos \mu_{t-1,\theta} - \frac{v_t}{\omega_t} \cos(\mu_{t-1,\theta} + \omega_t \Delta t) \\ \omega_t \Delta t \end{pmatrix}$$

4: $$\Delta = \begin{pmatrix} 0 & 0 & \frac{v_t}{\omega_t} \cos \mu_{t-1,\theta} - \frac{v_t}{\omega_t} \cos(\mu_{t-1,\theta} + \omega_t \Delta t) \\ 0 & 0 & \frac{v_t}{\omega_t} \sin \mu_{t-1,\theta} - \frac{v_t}{\omega_t} \sin(\mu_{t-1,\theta} + \omega_t \Delta t) \\ 0 & 0 & 0 \end{pmatrix}$$

5: $\Psi_t = F_x^T \, [(I + \Delta)^{-1} - I] \, F_x$

6: $\lambda_t = \Psi_t^T \, \Omega_{t-1} + \Omega_{t-1} \, \Psi_t + \Psi_t^T \, \Omega_{t-1} \, \Psi_t$

7: $\Phi_t = \Omega_{t-1} + \lambda_t$

8: $\kappa_t = \Phi_t \, F_x^T (R_t^{-1} + F_x \, \Phi_t \, F_x^T)^{-1} \, F_x \, \Phi_t$

9: $\bar{\Omega}_t = \Phi_t - \kappa_t$

10: $\bar{\xi}_t = \xi_{t-1} + (\lambda_t - \kappa_t) \, \mu_{t-1} + \bar{\Omega}_t \, F_x^T \, \delta_t$

11: $\bar{\mu}_t = \mu_{t-1} + F_x^T \, \delta$

12: *return* $\bar{\xi}_t, \bar{\Omega}_t, \bar{\mu}_t$

Figure B.2: Motion update algorithm [5]

1: **Algorithm SEIF_sparsification**(ξ_t, Ω_t):

2: *define* F_{m_0}, F_{x,m_0}, F_x *as projection matrices from* y_t *to* m_0, $\{x, m_0\}$, *and* x, *respectively*

3: $$\begin{aligned} \tilde{\Omega}_t = {} & \Omega_t - \Omega_t^0 \, F_{m_0} \, (F_{m_0}^T \, \Omega_t^0 \, F_{m_0})^{-1} \, F_{m_0}^T \, \Omega_t^0 \\ & + \Omega_t^0 \, F_{x,m_0} \, (F_{x,m_0}^T \, \Omega_t^0 \, F_{x,m_0})^{-1} \, F_{x,m_0}^T \, \Omega_t^0 \\ & - \Omega_t \, F_x \, (F_x^T \, \Omega_t F_x)^{-1} \, F_x^T \, \Omega_t \end{aligned}$$

4: $\tilde{\xi}_t = \xi_t + \mu_t \, (\tilde{\Omega}_t - \Omega_t)$

5: *return* $\tilde{\xi}_t, \tilde{\Omega}_t$

Figure B.3: Sparsification algorithm [5]

Algorithm SEIF_update_state_estimate($\bar{\xi}_t, \bar{\Omega}_t, \bar{\mu}_t$):

for a small set of map features m_i *do*

$$F_i = \begin{pmatrix} 0\cdots0 & 1 & 0 & 0\cdots0 \\ \underbrace{0\cdots0}_{2(N-i)} & 0 & 1 & \underbrace{0\cdots0}_{2(i-1)x} \end{pmatrix}$$

$$\mu_{i,t} = (F_i\ \Omega_t\ F_i^T)^{-1}\ F_i\ [\xi_t - \Omega_t\ \bar{\mu}_t + \Omega_t\ F_i^T\ F_i\ \bar{\mu}_t]$$

endfor

for all other map features m_i *do*

$$\mu_{i,t} = \bar{\mu}_{i,t}$$

endfor

$$F_x = \begin{pmatrix} 1 & 0 & 0 & 0\cdots0 \\ 0 & 1 & 0 & 0\cdots0 \\ 0 & 0 & 1 & \underbrace{0\cdots0}_{2N} \end{pmatrix}$$

$$\mu_{x,t} = (F_x\ \Omega_t\ F_x^T)^{-1}\ F_x\ [\xi_t - \Omega_t\ \bar{\mu}_t + \Omega_t\ F_x^T\ F_x\ \bar{\mu}_t]$$

return μ_t

Figure B.4: State estimation algorithm [5]

APPENDIX C: Map Merging Algorithm

Algorithm SEIF_map_fusion($\Omega^j, \xi^j, \Omega^k, \xi^k, d, \alpha, \mathcal{C}$):

$\Delta = (d_x \ d_y \ \alpha \ d_x \ d_y \ 0 \ \cdots \ d_x \ d_y \ 0)^T$

$$\mathcal{A} = \begin{pmatrix} \cos\alpha & \sin\alpha & 0 & & & & \cdots & 0 \\ -\sin\alpha & \cos\alpha & 0 & & & & & \vdots \\ 0 & 0 & 1 & & & & & \\ & & & \ddots & & & & \\ & & & & & \cos\alpha & \sin\alpha & 0 \\ \vdots & & & & & -\sin\alpha & \cos\alpha & 0 \\ 0 & & \cdots & & & 0 & 0 & 1 \end{pmatrix}$$

$\Omega^{j\to k} = \mathcal{A}\,\Omega^j\,\mathcal{A}^T$

$\xi^{j\to k} = \mathcal{A}\,(\xi^j - \Omega^{j\to k}\,\Delta)$

$$\Omega = \begin{pmatrix} \Omega^k & 0 \\ 0 & \Omega^{j\to k} \end{pmatrix}$$

$$\xi = \begin{pmatrix} \xi^k \\ \xi^{j\to k} \end{pmatrix}$$

for any pair $(m_j, m_k) \in \mathcal{C}^{j,k}$ *do*

$$F = \begin{pmatrix} 0\cdots0 & 1 & 0 & 0 & 0\cdots0 & -1 & 0 & 0 & 0\cdots0 \\ 0\cdots0 & 0 & 1 & 0 & 0\cdots0 & 0 & -1 & 0 & 0\cdots0 \\ 0\cdots0 & \underbrace{0 \quad 0}_{m_j} & & 1 & 0\cdots0 & \underbrace{0 \quad 0}_{m_k} & & -1 & 0\cdots0 \end{pmatrix}$$

$$\Omega \longleftarrow \Omega + F^T \begin{pmatrix} \infty & 0 & 0 \\ 0 & \infty & 0 \\ 0 & 0 & \infty \end{pmatrix} F$$

endfor

return Ω, ξ

Figure C.1: Map merging algorithm [5]

APPENDIX D: Effects of Sparsification in SEIF

SEIF estimated states of the robots and landmarks using the information matrix and information vector. When the sparsification was used in an implementation, it generated the approximated results as shown in Figure D.1. It can be compared to the state estimation of an implementation without the sparsification step as shown in Figure D.2.

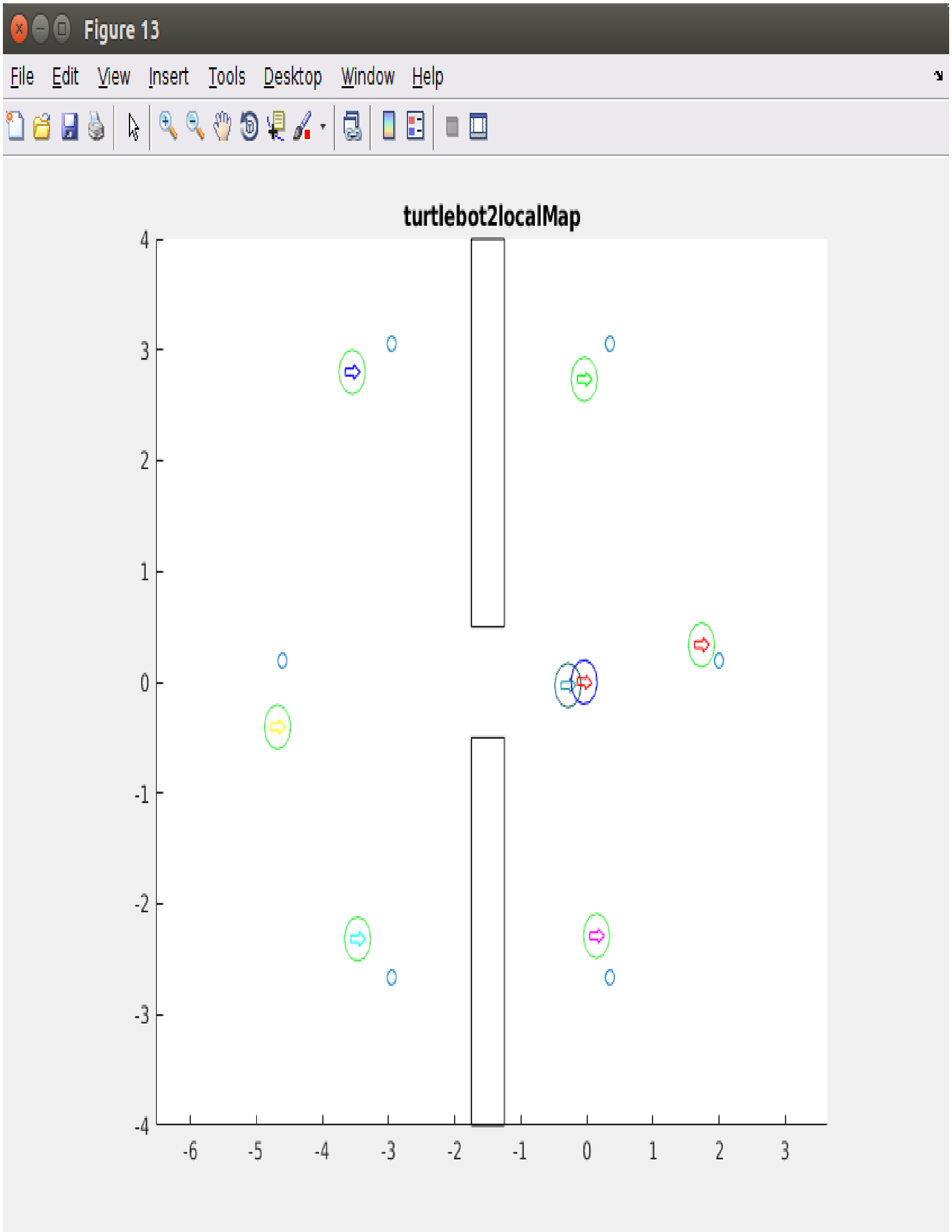


Figure D.1: Decentralized MR-SLAM with map merging with sparsification

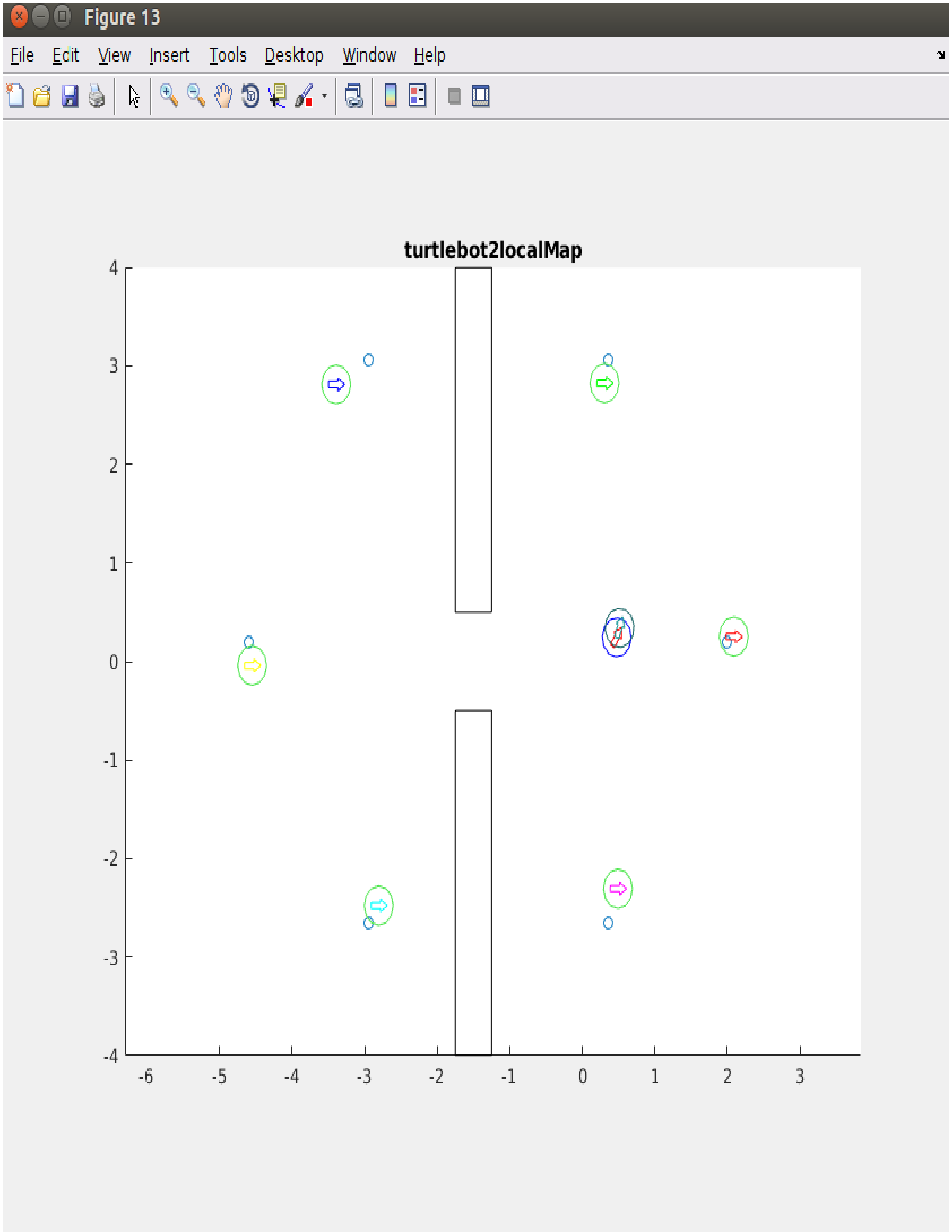


Figure D.2: Decentralized MR-SLAM with map merging without sparsification