MULTI-OBJECTIVE OPTIMISATION OF RTAB-MAP PARAMETERS USING GENETIC ALGORITHM FOR INDOOR 2D SLAM

by

Nagamalar Nagarajan

Submitted in partial fulfilment of the requirements
for the degree of Master of Applied Science

at

Dalhousie University
Halifax, Nova Scotia
December 2020

Dedicated to my family.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Currently, there are multiple packages available to implement different Simultaneous Localisation And Mapping (SLAM) approaches in Robot Operating System (ROS). To effectively obtain sensor data, these packages use parameters whose values are set from prior knowledge and experience working with robots and SLAM. In this research, using a Multi-Objective Genetic Algorithm (MOGA) to optimise the values for these parameters is proposed. Using MOGA allows trade-offs between the objectives using Pareto dominance technique. Three parameters from the RTAB-Map package are considered for optimisation using three different MOGA mechanisms, Dominance Count, Dominance Rank and Switching Fitness. The quality of the map generated for every set of parameters is taken as the indicator of its performance. The number of corners, number of contours and the proportion of occupied cells in the map are used as quantitative measures of map quality. Finally, results obtained from the algorithm are tested on a Quanser QBot2 robot.

LIST OF ABBREVIATIONS USED

SLAM	Simultaneous Localization And Mapping
AGV	Autonomous Ground Vehicle
GPS	Global Positioning System
EKF	Extended Kalman filter
PF	Particle Filter
ROS	Robot Operating System
GA	Genetic Algorithm
RGB-D	Red, Green, Blue-Depth
MOGA	Multi-Objective Genetic Algorithm
LASER	Light Amplification by Stimulated Emission of Radiation
RTAB-Map	Real-Time Appearance-Based Mapping
LISP	List Processing
MATLAB	Matrix Laboratory
OS	Operating System
SPA	Sparse Pose Adjustment
LIDAR	Light Detection And Ranging
RBPF	Rao-Blackwellised Particle Filter
2D	Two Dimensional
3D	Three Dimensional
LM	Levenberg-Marquardt
SIFT	Scale Invariant Feature Transform
SURF	Speeded Up Robust Features
CPU	Central Processing Unit
GB	Giga Byte
GHz	Giga Hertz
SSD	Solid State Drive
VM	Virtual Machine
RAM	Random Access Memory
LTS	Long-Term Support

SDK	Software Development Kit
USB	Universal Serial Bus
IMU	Inertial Measurement Unit
NUI	Natural User Interface
GP	Genetic Programming
LGP	Linear Genetic Programming
SOO	Single Objective Optimisation
MOO	Multi Objective Optimisation
AFL	Adaptive Fuzzy Logic
DC	Dominance Count
DR	Dominance Rank
SF	Switching Fitness
GUI	Graphical User Interface
OpenCV	Open Computer Vision

ACKNOWLEDGEMENTS

I would like to thank my supervisor Dr. Jason Gu for giving me the opportunity to work with him as a graduate student and for his constant motivation and guidance. I would also like to thank my supervisory committee, Dr. Kamal El-Sankary and Dr. William Phillips for their support and guidance. I would like to extend my gratitude to the late Dr. Mohamed El-Hawary for introducing me to fuzzy systems, Dr. Mae Seto for piquing my interest in the SLAM problem and Dr. Malcolm Heywood for his immense patience and motivation in developing my fascination with Genetic Algorithms.

I would like to thank Dr. Mathieu Labbe for sharing his knowledge on RTAB-Map and answering my many questions. I would also like to thank my husband for his unconditional support and encouragement.

Finally, I would like to thank all the faculty, staff and students of the Electrical and Computer Engineering department for making my Master's degree an enjoyable journey.

CHAPTER 1   INTRODUCTION

Initially robots were developed for industrial manufacturing and were stationary. However, now they are engaged in tasks that are less repetitive and in environments that are less structured for example, medical surgery, ocean and space explorations, search and rescue operations, etc. Using an autonomous robot reduces risk to people and increases persistence and reach. Autonomous Ground Vehicles (AGVs) can be used for search and rescue missions in collapsed mines or under debris from natural calamities, to scope hostile land in military applications, and as electric vehicles. In such cases, having a-priori knowledge of the environment might not be very useful due to these changes and in some cases, the environment will be dynamic and continuously changing.

Navigation is one of the most challenging features required of a mobile robot. A major flaw in most navigation and localization techniques is the assumption that the environment is known in advance. However, an autonomous mobile robot is required to start from an unknown initial point and independently explore the environment. Using its sensors, a mobile robot must not only determine its position in the environment, but also build a map of the area. This method of autonomous map building is called Simultaneous Localization And Mapping or SLAM [1]. It is especially useful in cases where GPS is unavailable, so the maps developed by SLAM can be used for exploration and motion planning. At the heart of most of the commonly used SLAM techniques, lies a Bayesian probability filter like the Gaussian Extended Kalman filter (EKF) or the nonparametric Particle filter (PF).

To avoid exerting effort in duplicating existing technology, Robot Operating System (ROS) is a useful platform which makes many opensource packages and programming codes readily available for research and educational purposes. Currently, there are multiple ROS packages available that implement SLAM, like GMapping that uses PF, Hector SLAM that uses EKF, RTAB-Map and Karto SLAM that use a simple Bayesian filter for loop closure to implement a graph-based SLAM approach. There are numerous parameters used by these packages for effective localisation and map generation. The values and ranges for these parameters are defined from previous experiences with robots and SLAM. However, to avoid ambiguity in setting the values for these parameters and to customise them for specific research needs, using a Genetic Algorithm (GA) for optimising these parameters would be a good choice. This is especially useful in cases where map accuracy is of utmost importance and in long term projects that require a robot to perform exploration repetitively.

In currently available literature, the use of GA for optimal implementation of SLAM is limited and using MOGA in SLAM is uncharted territory; therefore, making the approach proposed in this thesis unique. When using GA, multiple parameters can be optimised simultaneously by implementing a Multi-Objective GA (MOGA). This allows identifying parameter values that are best suited for the robot, its sensor set and the environment to be employed in the task, and prevents having to rely on historical data obtained by outdated technology. Using MOGA for this optimisation prevents premature convergence to local optima and solves bootstrap problem, in other words, the absence of selective pressure [2]. Unlike single objective GA, MOGA eliminates the need to define a weight for the multiple objectives a-priori as it allows expressing the multiple objectives using Pareto dominance techniques. Furthermore, the results obtained via Pareto dominance techniques offer multiple trade-offs between the objectives which will be useful in making a-posteriori choices between solutions.

In this thesis, a MOGA is employed to identify optimal values of select parameters used by the RTAB-Map package for generating maps of the environment. The maps are evaluated based on their number of corners, number of contours and proportion of occupied spaces, to promote evolution of parameter sets that produce better maps. Different Pareto dominance techniques have been explored to compare the promotion of a single fittest individual versus the promotion of a pool of individuals that provide equally good but different solutions.

The robot used for the different sets of experiments in this research, Quanser QBot2, is an autonomous ground robot (AGV) with a Yujin Robot Kobuki platform, a Microsoft Kinect RGB camera and depth sensor, and a Quanser data acquisition device with a wireless embedded computer. The RTAB-Map package has been chosen from multiple

SLAM packages available in ROS primarily due to its ability to support a camera-based robot out of the box and the continuous support available for the package. For simulation purposes, a Turtlebot2 robot, which has a similar configuration as the QBot2, has been used.

1.1 THESIS OUTLINE

This thesis consists of 7 chapters, where chapter 1 is the introduction. The other chapters are organized in the following manner:

- Chapter 2 provides an overview of the SLAM problem. It introduces ROS and lists the available ROS packages that implement different SLAM techniques.
- Chapter 3 introduces the AGV, its various sensors used to solve the SLAM problem in this thesis and the simulated robot.
- Chapter 4 explains how the virtual machines were setup for the simulation experiments and provides details on ROS installation and the different ROS packages used.
- Chapter 5 is a basic outline of the RTAB-Map and genetic algorithms. It discusses the RTAB-Map's memory management model and the parameters chosen for optimisation. It also introduces GA, how the first GA SLAM was implemented, different MOGA mechanisms and the objectives used to optimise maps produced by RTAB-Map.
- Chapter 6 presents the implementation of the MOGA methods developed for parameter optimisation and details the observations made by the robot and the results comparing the different MOGA approaches employed in the different setups.
- Chapter 7 discusses the future scope of the algorithm and other ways MOGA can be incorporated in SLAM.

CHAPTER 2 SIMULTANEOUS LOCALIZATION AND MAPPING AND ROS

Mobile robots need to be able to explore unknown environments, especially those that are remote or too hazardous. This necessitates that mobile robots be autonomous. In autonomous mobile robotics, it is essential that a mobile robot is able to start from an arbitrary initial point and autonomously explore the environment with its on-board sensors, gain knowledge about it, interpret the scene, build an appropriate map, and localize itself relative to this map [1]. This problem is called simultaneous localization and mapping.

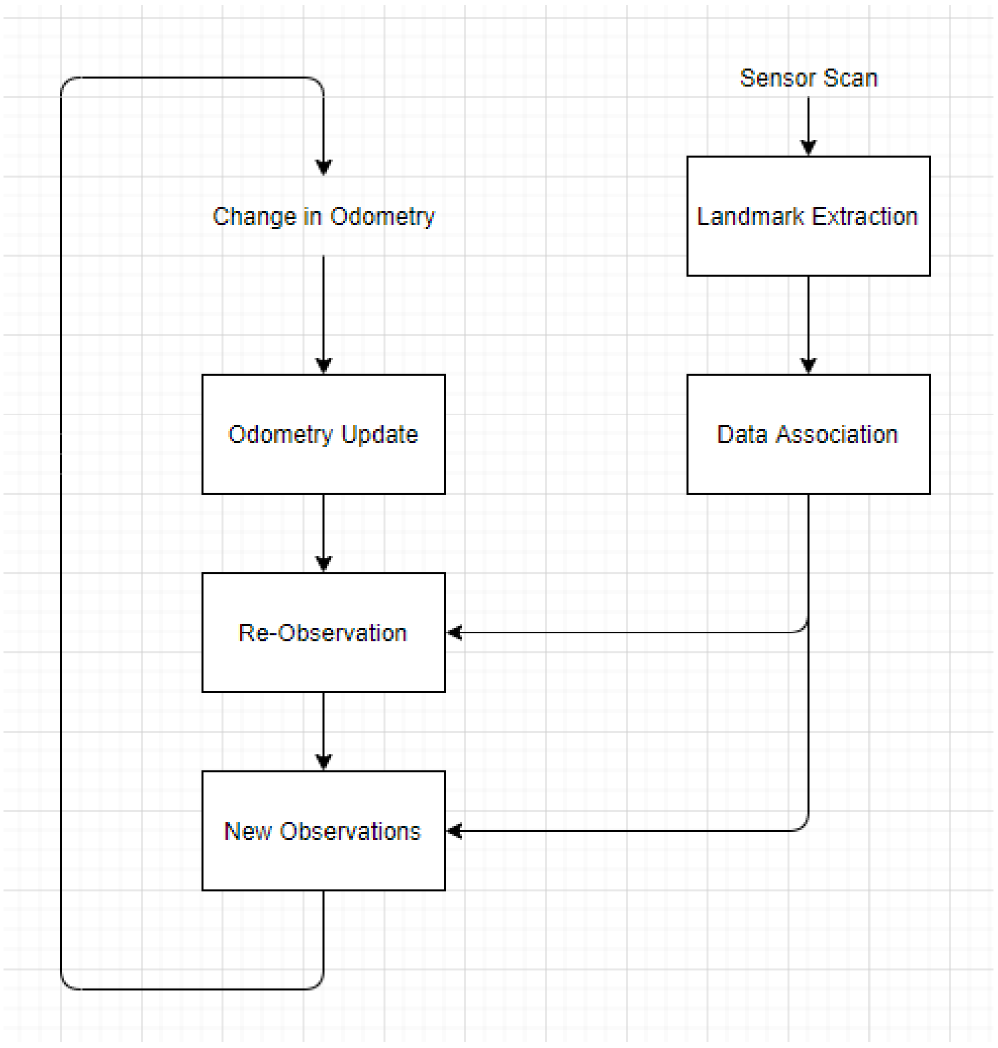


Figure 1 Overview of the SLAM process.

Since its introduction in [3] SLAM has been implemented in multiple ways, an example is shown in Figure 1 where the SLAM process consists of landmark extraction, data association, state estimation, state update and landmark update with multiple ways to solve each step [6]. The changes in odometry and observations are updated in the filter, based on scans from different robot sensors and data association, i.e., it determines if sensor measurements taken at different times correspond to the same physical object. As shown in Figure 2, the prediction and measurement updates at each step yield a map where the uncertainty at each step is compounded by the uncertainty in estimation and the noise in sensor measurements and the environment. This necessitates loop closure detection as it shrinks the robot pose uncertainty.

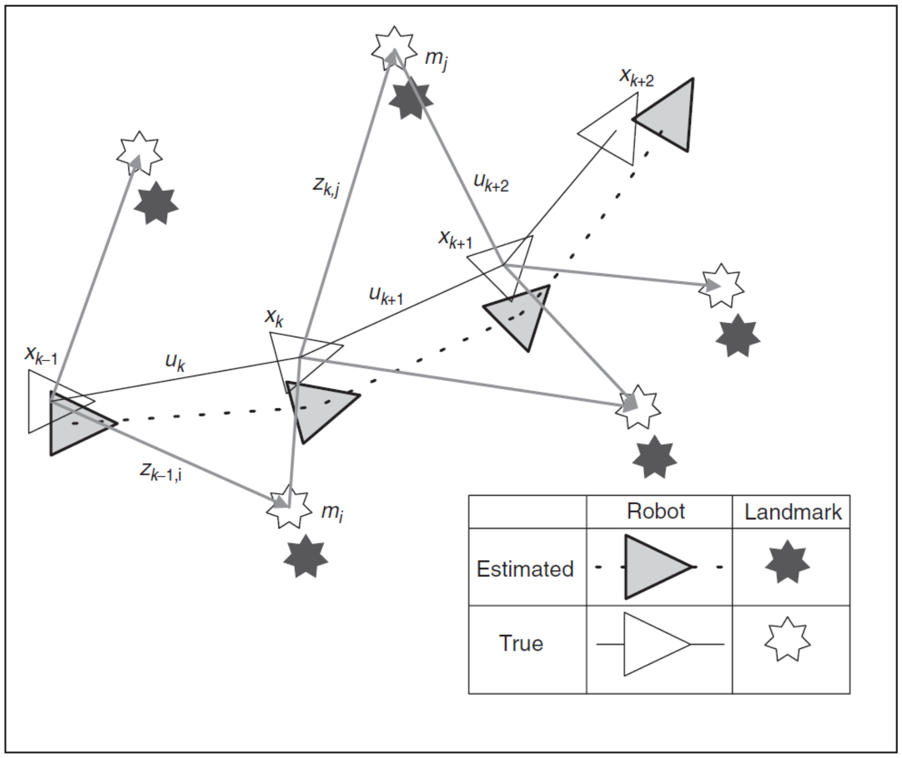


Figure 2 The SLAM problem [4].

Data association is the process of matching observed landmarks from different scans from the robot sensor/s with each other, i.e., it determines if sensor measurements taken at different times correspond to the same physical object [4]. This can also be referred to as re-observing landmarks. In most SLAM algorithms with unknown data association, i.e., ambiguous landmarks, multiple tracks/paths need to be maintained so that a search for the appropriate track can be conducted. This increases the computational complexity of these algorithms [5]. Errors in data association commonly occur due to measurement error and motion error. As shown in Figure 3, if two landmarks are very close to each other, observations could come from either landmark.

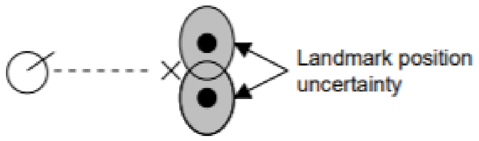


Figure 3 Measurement error [6].

If a robot skids or its pose changes by a small amount, observations from a single landmark could be associated with different landmarks as shown in Figure 4 [6].

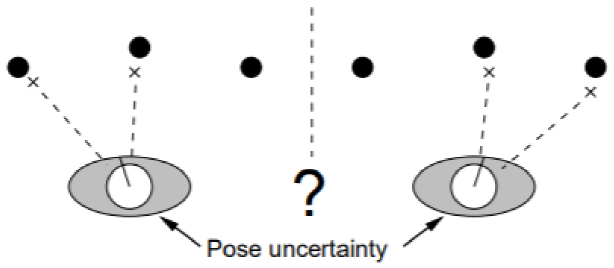


Figure 4 Motion error [6].

Probabilistic SLAM is sensitive to incorrect data associations and increasing the complexity of environments makes it run out of computational time and storage space eventually.

2.1 ROBOT OPERATING SYSTEM

ROS is an open-source framework for creating robot software. It provides tools, libraries and conventions across a wide variety of robotic platforms that aim to simplify complex programming tasks, so that developers and researchers can focus on new algorithm development without having to cope with trivial low-level hardware communication problems [7], [8]. It was introduced in 2007 by Stanford University and Willow Garage. ROS is multilingual and supports languages with client libraries like Python, C++, LISP and MATLAB [7]. The three typical components of ROS are illustrated in Figure 5. In ROS, applications running in the landscape are called nodes or nodelets. The distinction being, a node is mapped to a single OS process and ROS nodelets reside as threads inside a process called ROS Nodelet Manager [9]. Topics are publish/subscribe methods of exchange, services establish request/response communication model, and these three components communicate via messages [8].

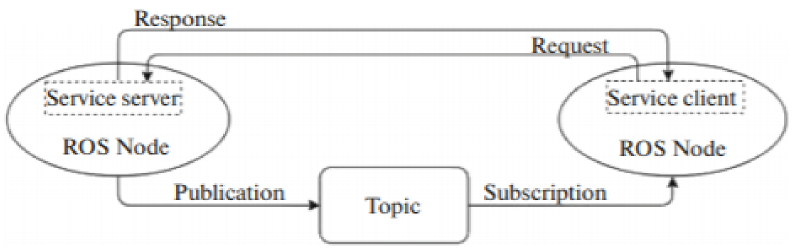


Figure 5 Basic ROS computational graph [8].

The nodelets communicate through a link between the ROS layers as indicated in Figure 6. Messages across nodes are transmitted through the operating system network layers [9].

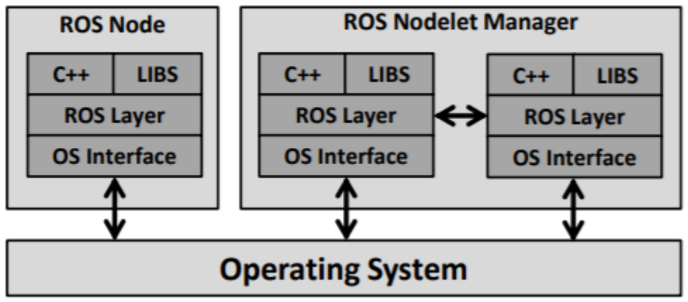


Figure 6 ROS Application Architecture [9].

Currently ROS only runs on Unix-based platforms like the Ubuntu and Mac OS X systems and is fully supported. More recently, it can also be installed on Microsoft Windows 10 through the Windows Subsystem for Linux, however the support is limited with minimal or no access to hardware and subpar performance [10].

2.2 SLAM PACKAGES IN ROS

The most popular ROS packages that implement SLAM are *gmapping* which is a LASER based SLAM approach, *hector_slam* which is LIDAR based, *slam_karto* which is again LASER based, and *rtabmap* which is a camera (RGBD, stereo) based SLAM approach. These packages are discussed in the following sections.

2.2.1 Hector SLAM

As shown in Figure 7, the Hector SLAM package uses an EKF filter for navigation where it combines information from different sensors to provide a consistent 3D state estimation and the robot’s position heading information are provided by a 2D SLAM system.

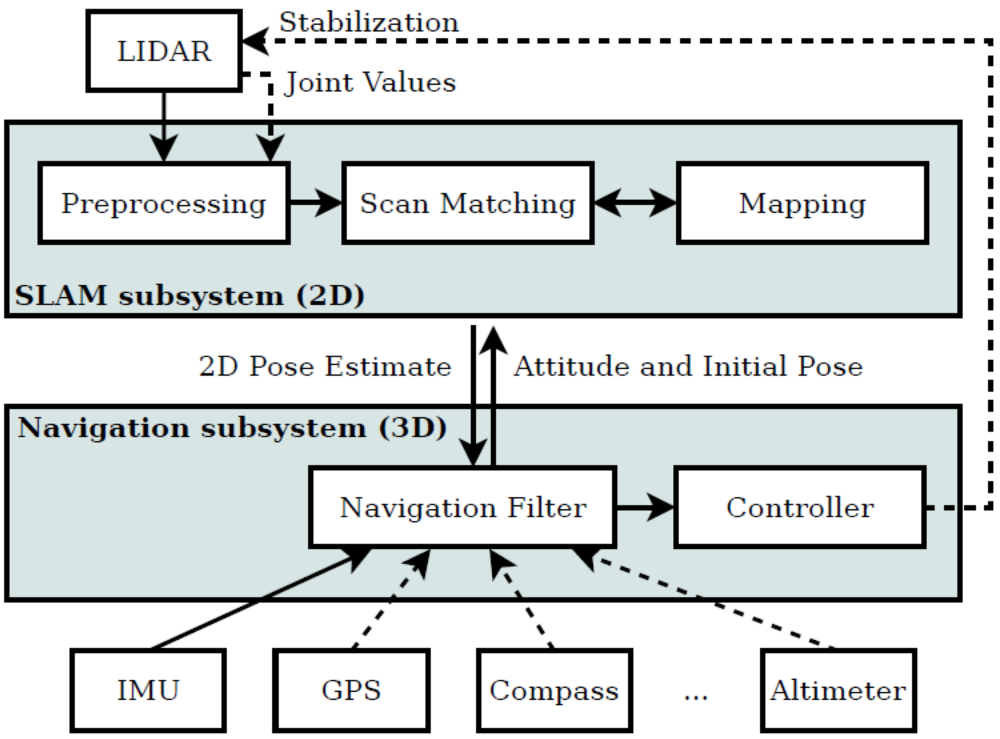


Figure 7 Overview of the ROS hector_slam mapping and navigation system [11].

EKF relaxes the linear assumption of the Kalman filter to include realistic non-linear functions, however, it still assumes Gaussian noise. With the environment being unknown, the robot's pose at its initial point is taken as the origin. According to [12], the initial mean and covariance will be:

$$\mu_0 = (0 \quad 0 \quad 0 \ldots 0)^T \tag{2.1}$$

$$\Sigma_0 = \begin{pmatrix} 0 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 0 & \infty & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & \infty \end{pmatrix}. \tag{2.2}$$

When the robot moves the state vector changes,

$$y_t = y_{t-1} + \begin{pmatrix} -\frac{v_t}{\omega_t}\sin\theta + \frac{v_t}{\omega_t}\sin(\theta + \omega_t\Delta t) \\ \frac{v_t}{\omega_t}\cos\theta - \frac{v_t}{\omega_t}\cos(\theta + \omega_t\Delta t) \\ \omega_t\Delta t + \gamma_t\Delta t \\ 0 \\ \vdots \\ 0 \end{pmatrix} \tag{2.3}$$

and only the first three elements in the above motion model update are non-zero as the landmarks remain fixed. So,

$$y_t = y_{t-1} + F_x{}^T \begin{pmatrix} -\frac{v_t}{\omega_t}\sin\theta + \frac{v_t}{\omega_t}\sin(\theta + \omega_t\Delta t) \\ \frac{v_t}{\omega_t}\cos\theta - \frac{v_t}{\omega_t}\cos(\theta + \omega_t\Delta t) \\ \omega_t\Delta t + \gamma_t\Delta t \end{pmatrix} \tag{2.4}$$

where, x, y and θ denote the pose of the robot at time t-1. v_t is the translational velocity and ω_t is the rotational velocity with Δt being the time frame of the robot motion. F_x is 3×(3N+3) matrix,

$$F_x = \begin{pmatrix} 1 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 1 & 0 & \cdots & 0 \end{pmatrix}. \tag{2.5}$$

The noise-free full motion model with a random noise part is given by,

$$y_t = \underbrace{y_{t-1} + F_x^T \begin{pmatrix} -\frac{v_t}{\omega_t}\sin\theta + \frac{v_t}{\omega_t}\sin(\theta + \omega_t \Delta t) \\ \frac{v_t}{\omega_t}\cos\theta - \frac{v_t}{\omega_t}\cos(\theta + \omega_t \Delta t) \\ \omega_t \Delta t \end{pmatrix}}_{g(u_t, y_{t-1})} + N(0, F_x^T R_t F_x) \qquad 2.6$$

where $F_x^T R_t F_x$ extends the covariance matrix to the dimension of the full state vector squared. In EKF, linearization approximates the motion function, g, using a first-degree Taylor expansion,

$$g(u_t, y_{t-1}) \approx g(u_t, \mu_{t-1}) + G_t(y_{t-1} - \mu_{t-1}) \qquad 2.7$$

where the function $g(u_t, \mu_{t-1})$ is the estimation and G_t is derivative of g at the control input u_t and μ_{t-1} with respect to y_{t-1}. Equation 2.6 allows us to decompose the Jacobian into an identity matrix and a low dimensional Jacobian g_t that gives the robot pose change as:

$$G_t = I + F_x^T g_t F_x \qquad 2.8$$

with

$$g_t = \begin{pmatrix} 0 & 0 & -\frac{v_t}{\omega_t}\cos\mu_{t-1,\theta} + \frac{v_t}{\omega_t}\cos(\mu_{t-1,\theta} + \omega_t \Delta t) \\ 0 & 0 & -\frac{v_t}{\omega_t}\sin\mu_{t-1,\theta} + \frac{v_t}{\omega_t}\sin(\mu_{t-1,\theta} + \omega_t \Delta t) \\ 0 & 0 & 0 \end{pmatrix}. \qquad 2.9$$

Substituting these in EKF gives the mean and covariance at time t,

$$\bar{\mu}_t = \mu_{t-1} + F_x^T \begin{pmatrix} -\frac{v_t}{\omega_t}\sin\mu_{t-1,\theta} + \frac{v_t}{\omega_t}\sin(\mu_{t-1,\theta} + \omega_t \Delta t) \\ \frac{v_t}{\omega_t}\cos\mu_{t-1,\theta} - \frac{v_t}{\omega_t}\cos(\mu_{t-1,\theta} + \omega_t \Delta t) \\ \omega_t \Delta t \end{pmatrix} \qquad 2.10$$

$$\bar{\Sigma}_t = G_t \Sigma_{t-1} G_t^T + F_x^T R_t F_x \,. \qquad 2.11$$

EKF also requires a measurement model with a noise component,

$$z_t^i = \begin{pmatrix} r_t^i \\ \phi_t^i \\ s_t^i \end{pmatrix} = \underbrace{\begin{pmatrix} \sqrt{\left(m_{j,x} - x\right)^2 + \left(m_{j,y} - y\right)^2} \\ atan2\left(m_{j,y} - y, m_{j,x} - x\right) - \theta \\ m_{j,s} \end{pmatrix}}_{h(y_t, j)} + N\left(0, \underbrace{\begin{pmatrix} \sigma_r & 0 & 0 \\ 0 & \sigma_\phi & 0 \\ 0 & 0 & \sigma_s \end{pmatrix}}_{Q_t}\right) \quad 2.12$$

where $m_{j,x}$ and $m_{j,y}$ denote the landmark coordinates at time t and $m_{j,s}$ is its signature. ϕ is the bearing and r is the range of landmark j, which is the ith component in the measurement vector and s is its signature as observed by the robot. Therefore, the Taylor approximation gives us,

$$h(y_t, j) \approx h(\bar{\mu}_t, j) + H_t^i(y_t - \bar{\mu}_t)\,. \quad 2.13$$

Here H_t^i is the derivative of h with respect to the full state vector y_t. Since the measurement function h, only depends on the robot pose x_t and the location of the jth landmark m_j, the derivative factors into a low-dimensional Jacobian h_t^i and a matrix $F_{x,j}$, which maps h_t^i into a matrix of the dimension of the full state vector:

$$H_t^i = h_t^i F_{x,j} \quad 2.14$$

where, $h_t^i$is the Jacobian of the function $h(y_t, j)$ at $\bar{\mu}_t$ calculated with respect to x_t and m_j:

$$h_t^i = \begin{pmatrix} \frac{\bar{\mu}_{t,x} - \bar{\mu}_{j,x}}{\sqrt{q_t}} & \frac{\bar{\mu}_{t,y} - \bar{\mu}_{j,y}}{\sqrt{q_t}} & 0 & \frac{\bar{\mu}_{j,x} - \bar{\mu}_{t,x}}{\sqrt{q_t}} & \frac{\bar{\mu}_{j,y} - \bar{\mu}_{t,y}}{\sqrt{q_t}} & 0 \\ \frac{\bar{\mu}_{j,y} - \bar{\mu}_{t,y}}{q_t} & \frac{\bar{\mu}_{t,x} - \bar{\mu}_{j,x}}{q_t} & -1 & \frac{\bar{\mu}_{t,y} - \bar{\mu}_{j,y}}{q_t} & \frac{\bar{\mu}_{j,x} - \bar{\mu}_{t,x}}{q_t} & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}. \quad 2.15$$

The scalar $q_t = (\bar{\mu}_{j,x} - \bar{\mu}_{t,x})^2 + (\bar{\mu}_{j,y} - \bar{\mu}_{t,y})^2$, and $j = c_t^i$ is the landmark corresponding to z_t^i. $F_{x,j}$ is a 6×(3N+3) matrix which maps h_t^i into a 3×(3N+3) matrix.

$$F_{x,j} = \begin{pmatrix} 1 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & 0 & \cdots & 0 & 0 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 1 & 0 & \cdots & 0 & 0 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 & 1 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 1 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 & 1 & 0 & \cdots & 0 \end{pmatrix}. \qquad 2.16$$

Inserting these equations in EKF gives us the Kalman gain and allows addition of observations into the filter in equations 2.18 and 2.19,

$$K_t^i = \bar{\Sigma}_t H_t^{iT} \left(H_t^i \bar{\Sigma}_t H_t^{iT} + Q_t \right)^{-1} \qquad 2.17$$

$$\bar{\mu}_t = \bar{\mu}_t + K_t^i \left(z_t^i - \hat{z}_t^i \right) \qquad 2.18$$

$$\bar{\Sigma}_t = \left(I - K_t^i H_t^i \right) \bar{\Sigma}_t \,. \qquad 2.19$$

From the above derivation, it can be concluded that EKF is computationally expensive. Additionally, it requires memory and update time that is quadratic in N, the number of landmarks [12]. Also, EKF functions on single hypothesis data association, i.e., it has no mechanism to represent uncertainty in data association. If a landmark is incorrectly associated, it can never be undone [13], therefore using the 2D SLAM system for scan matching in the hector_slam ROS package prevents overconfidence in the EKF estimates [11].

2.2.2 GMapping

The GMapping package uses a Rao-Blackwellized Particle Filter (RBPF) with raw laser data from long range LASER scanners and odometry to create 2D grid maps. Unlike the Gaussian Kalman filter and EKF, a PF is well suited for multimodal beliefs. In PFs the posterior belief $bel(x_t)$ is represented by a random set of state samples or particles drawn from the parametric probability distribution of the posterior and as explored in [12] is denoted by,

$$x_{0:t}^{[m]} = x_0^{[m]}, x_1^{[m]}, \ldots, x_t^{[m]} \,. \qquad 2.20$$

This algorithm can be easily modified by adding $x_t^{[m]}$ to $x_{0:t-1}^{[m]}$. The posterior is calculated over all state sequences as opposed to the current state sequence:

$$bel(x_{0:t}) = p(x_{0:t}|u_{1:t}, z_{1:t}) \tag{2.21}$$

where, u_t is the control variable and z_t is the measurement. Based on the Bayes filter algorithm,

$$p(x_{0:t}|u_{1:t}, z_{1:t}) = \eta p(z_t|x_{0:t}, z_{1:t-1}, u_{1:t}) p(x_{0:t}|z_{1:t-1}, u_{1:t}) \tag{2.22}$$

based on the Markov assumption,

$$= \eta p(z_t|x_t) p(x_{0:t}|z_{1:t-1}, u_{1:t}) \tag{2.23}$$

$$= \eta p(z_t|x_t) p(x_t|x_{0:t-1}, z_{1:t-1}, u_{1:t}) p(x_{0:t-1}|z_{1:t-1}, u_{1:t}) \tag{2.24}$$

$$p(x_{0:t}|u_{1:t}, z_{1:t}) = \eta p(z_t|x_t) p(x_t|x_{t-1}, u_t) p(x_{0:t-1}|z_{1:t-1}, u_{1:t-1})\,. \tag{2.25}$$

Here, the constant η is the normalization factor. Maintaining all states in the posterior leads to the absence of integral signs. Assuming the particle set at time t-1 is distributed according to $bel(x_{0:t-1})$, for the mth particle in the set $x_{0:t-1}^{[m]}$, the sample $x_t^{[m]}$ is generated from the proposal distribution:

$$p(x_t|x_{t-1}, u_t) bel(x_{0:t-1}) = p(x_t|x_{t-1}, u_t) p(x_{0:t-1}|z_{1:t-1}, u_{1:t-1}) \tag{2.26}$$

with

$$w_t^m = \frac{target\ distribution}{proposal\ distribution} \tag{2.27}$$

$$= \frac{\eta p(z_t|x_t) p(x_t|x_{t-1}, u_t) p(x_{0:t-1}|z_{1:t-1}, u_{1:t-1})}{p(x_t|x_{t-1}, u_t) p(x_{0:t-1}|z_{1:t-1}, u_{1:t-1})} \tag{2.28}$$

$$= \eta p(z_t|x_t)\,. \tag{2.29}$$

By resampling particles with probability proportional to importance weight w_t^m, the resulting particles are distributed according to,

$$\eta w_t^m p(x_t|x_{t-1}, u_t) p(x_{0:t-1}|z_{1:t-1}, u_{1:t-1}) = bel(x_{0:t}) . \qquad 2.30$$

If $x_{0:t}^{[m]}$ is distributed according to $bel(x_{0:t})$, then $x_t^{[m]}$ is distributed according to $bel(x_t)$.

[12] discusses some practical considerations in the PF algorithm:

- variability due to random sampling,
- variation is amplified by resampling,
- choosing a set of samples from a distribution introduces sampling bias,
- exponential computation cost, and
- absence of particles in the correct state vicinity.

The RBPF method as introduced in [14] reduces computation effort as SLAM can potentially have millions of dimensions to make it more suitable for the SLAM problem. Using [15] and [16], the gmapping approach uses a PF in which each particle carries an individual map of the environment. To reduce the number of particles, a selective resampling strategy based on the effective sample size is applied considering the movement of the robot and the most recent observation.

2.2.3 Karto SLAM

Karto SLAM uses a graph-based SLAM approach like the one shown in Figure 8.

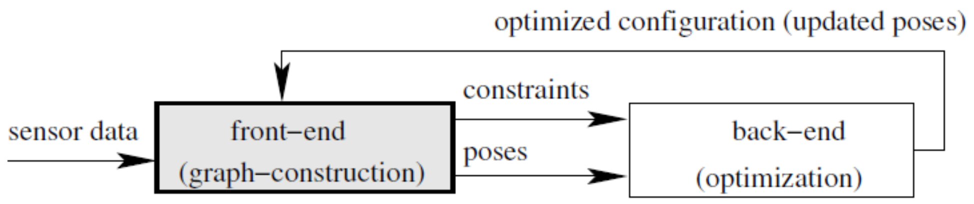


Figure 8 Typical graph-based SLAM system [17].

The front-end of a graph-based SLAM deals with sensor data whereas the backend is where the robot poses are corrected for an efficient environment map. As discussed in [17], Karto SLAM uses the Levenberg-Marquardt (LM) algorithm as a framework to optimise a set of poses, **c** which is a collection of the robot's translation 't' and angle 'θ' as given by,

$$c_i = [t_i, \theta_i] = [x_i, y_i, \theta_i] \qquad 2.31$$

and constraints, which are the measurement of a node, c_i, from another, c_j. Their offset,

$$h(c_i, c_j) \equiv \begin{cases} R_i^T(t_j - t_i) \\ \theta_j - \theta_i \end{cases} \qquad 2.32$$

is the measurement equation, where R_i is given by the 2×2 rotation matrix of θ_i. The error associated with each constraint is,

$$e_{ij} \equiv \bar{z}_{ij} - h(c_i, c_j) \qquad 2.33$$

and the total error is,

$$\chi^2(\boldsymbol{c}, \boldsymbol{p}) \equiv \sum_{ij} e_{ij}^T \Lambda_{ij} e_{ij} \qquad 2.34$$

where, $\bar{z}_{ij}$ is the measured offset between c_i and c_j in c_i's frame and Λ_{ij} is the precision matrix. Each iteration of the LM algorithm,

- sets up the linear system by assigning the values of **c** to vector *x* and error function to vector *e*,

$$\Lambda \equiv \begin{bmatrix} \Lambda_{ab} & & \\ & \ddots & \\ & & \Lambda_{mn} \end{bmatrix} \qquad 2.35$$

$$\mathbf{J} \equiv \frac{\partial e}{\partial x} \qquad 2.36$$

$$\mathbf{H} \equiv \mathbf{J}^T \Lambda \mathbf{J} \qquad 2.37$$

- decomposes **H**,

$$(\mathbf{H} + \lambda\, diag\mathbf{H})\Delta \mathrm{x} = \mathbf{J}^T \Lambda e \qquad 2.38$$

where λ is a small multiplier and **J** is the Jacobian of the measurement function *h* given by,

$$\frac{\partial e_{ij}}{\partial t_i} \equiv \begin{bmatrix} -R_i^T \\ 0 \quad 0 \end{bmatrix} \qquad \frac{\partial e_{ij}}{\partial \theta_i} \equiv \begin{bmatrix} {\partial R_i^T}/{\partial \theta_i}\,(t_j - t_i) \\ -1 \end{bmatrix}$$
$$\frac{\partial e_{ij}}{\partial t_j} \equiv \begin{bmatrix} R_i^T \\ 0 \quad 0 \end{bmatrix} \qquad \frac{\partial e_{ij}}{\partial \theta_i} \equiv \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}^T \tag{2.39}$$

For the Jacobian J_i of e_{ij} with respect to c_i, **H** is formed by adding 4 components for each measurement,

$$\begin{matrix} \ddots & & & \\ & J_i^T \Lambda_{ij} J_i & \cdots & J_i^T \Lambda_{ij} J_j \\ & \vdots & \ddots & \vdots \\ & J_j^T \Lambda_{ij} J_i & \cdots & J_j^T \Lambda_{ij} J_j \\ & & & & \ddots \end{matrix} \tag{2.40}$$

- and computes Δx by solving the linear equation. Adding this increment to the current value yields,

$$t_i = t_i + \Delta t_i \qquad \theta_i = \theta_i + \Delta\theta_i \tag{2.41}$$

Karto SLAM applies a simple Bayesian filter as explored in [18] for scan matching and recognizing previously visited locations, also known as loop closure, which then removes irrelevant and poor-quality observations.

2.2.4 Real-Time Appearance-Based Mapping

RTAB-Map, as introduced in [19], is a graph-based SLAM approach which uses a discrete Bayesian filter to track its appearance-based loop closure detection with an efficient memory management approach [20] shown in Figure 9. Appearance based methods use data collected from vision sensors for localization and building a map of the environment.

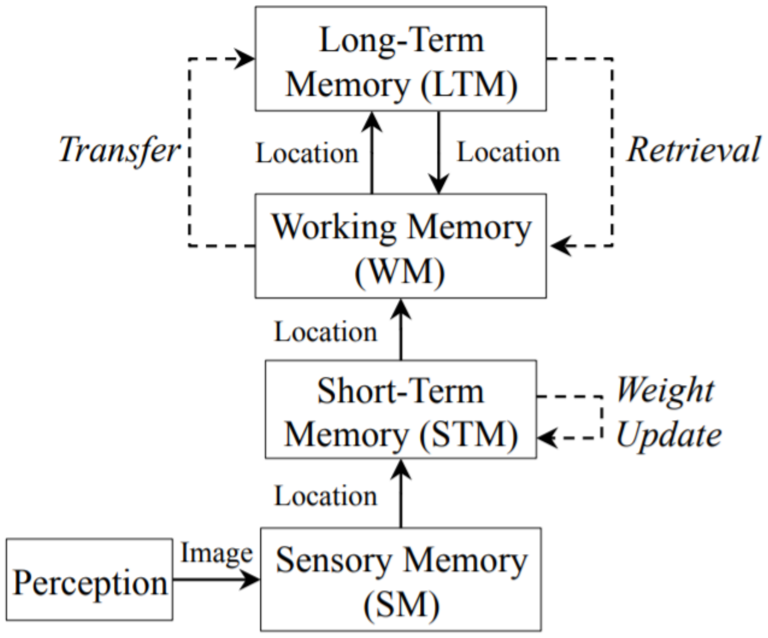


Figure 9 RTAB-Map Memory management model [20].

As discussed in [19], [20] and [21], RTAB-Map uses SIFT/SURF local feature descriptor to extract information from images captured by the vision sensor. The discrete Bayesian filter tracks loop closures by calculating the probability that the current location has already been visited using:

$$\boldsymbol{p}(S_t|L^t) = \underbrace{\eta \boldsymbol{p}(L_t|S_t)}_{\text{Observation}} \underbrace{\sum_{i=-1}^{t_n} \underbrace{\boldsymbol{p}(S_t|S_{t-1}=i)}_{\text{Transition}} p(S_{t-1}=i|L^{t-1})}_{\text{Belief}} \;. \qquad 2.38$$

Here, S_t is the variable that represents the loop closure state, L_t is the current location, t_n is the time index for the newest location, η is used for normalization and L^t is the set of all observed locations. The observation model is estimated using:

$$p(L_t|S_t = j) = \mathfrak{L}(S_t = j|L_t) = \begin{cases} \dfrac{s_j - \sigma}{\mu}, & \text{if } s_j \geq \mu + \sigma \\ 1, & \text{otherwise} \end{cases} \qquad 2.39$$

where, $\mathfrak{L}(S_t|L_t)$ is the likelihood function, s_j is the score with σ standard deviation and μ mean.

As the robot ventures deeper in the environment the number of images to compare will increase making loop closure detection slower. To make the process more effective and real-time, RTAB-Map's memory management model uses only a certain number of locations, stored in its working memory, for loop closure detection and rest of the locations are stored in its long-term memory. This means that only the current map of the environment is built locally while the global map is updated online. Loop closure detection, like in all SLAM algorithms, helps optimise the map built. RTAB-Map will be discussed further in chapter 5.

2.3 PARAMETER OPTIMISATION USING GA

In the packages discussed above, multiple parameters, like the ones mentioned in Table 1, with a range of values are used to produce effective maps. For example, in Karto SLAM the radius of search area and the number of consecutive nodes can be used to finetune the resultant map. The values for these parameters are set using prior experience and knowledge in the field. Using a GA to get optimal values for these parameters not only allows testing various combinations of parameter values, but also most of these tests can be run in simulation. This means real-time testing can be limited to the optimal set of values which makes it ideal when physical resources are limited or available for a limited amount of time.

Table 1 Partial list of parameters used in different SLAM packages.

Hector SLAM	GMapping	Karto SLAM	RTAB-Map
~map_update_distance_thresh	~throttle_scans	~throttle_scans	~rgbd_cameras
~map_update_angle_thresh	~map_update_interval	~map_update_interval	~queue_size
~laser_min_dist	~maxUrange	~minimum_travel_distance	~map_filter_angle
~laser_max_dist	~iterations	~minimum_travel_heading	~iterations

Hector SLAM	GMapping	Karto SLAM	RTAB-Map
~scan_subscriber_queue_size	~linearUpdate	~scan_buffer_size	~Vis/MinInliers
~map_resolution	~angularUpdate	~loop_search_maximum_distance	~Mem/RehearsalSimilarity
~map_size	~particles	~loop_search_space_dimension	~Kp/MaxDepth

CHAPTER 3 QBOT2 SENSORS AND THEIR APPLICATIONS

This chapter introduces the robot used for this research and briefly discusses its various sensors. This setup uses a Quanser QBot2, shown in Figure 10, which is an AGV that is mounted on a Kobuki mobile robot platform. Table 2 lists the main specifications of the robot.

Table 2 QBot2 specifications [22].

Symbol	Description	Value	Unit
D	Diameter of the QBot 2	0.35	m
d	Distance between the left and right wheels	0.235	m
h	Height of the QBot 2 (with Kinect mounted)	0.27	m
ν_{max}	Maximum speed of the QBot 2	0.7	m/s
m	Total mass of the QBot 2	3.79	kg

Robots use a wide variety of sensors to obtain information about their surroundings. These can be classified into sensors that sense a robot's internal measurements, i.e., proprioceptive and sensors that obtain data from the environment, i.e., exteroceptive; sensors that measure surrounding energy, i.e., passive and sensors that interact with the territory, i.e., active [1]. The QBot2 has wheel encoders, cliff sensors, bumper sensors, wheel drop sensors, overcurrent sensors, a 3-axis gyroscope, a battery voltage sensor and a Microsoft Kinect RGB camera and depth sensor [22]. Here, the wheel encoders, gyroscope and battery voltage sensors are examples of proprioceptive whereas the bumper sensors and the Kinect sensor are exteroceptive. Additionally, bumper sensors, gyroscope and Kinect sensor are passive whereas wheel encoders and cliff sensors are active sensors. As mentioned in [22], the QBot2 contains three digital bumper sensors, left, right and centre, located on the front frame of the Kobuki base. They return binary outputs indicating contact or collision based on whether the frame is compressed or not.

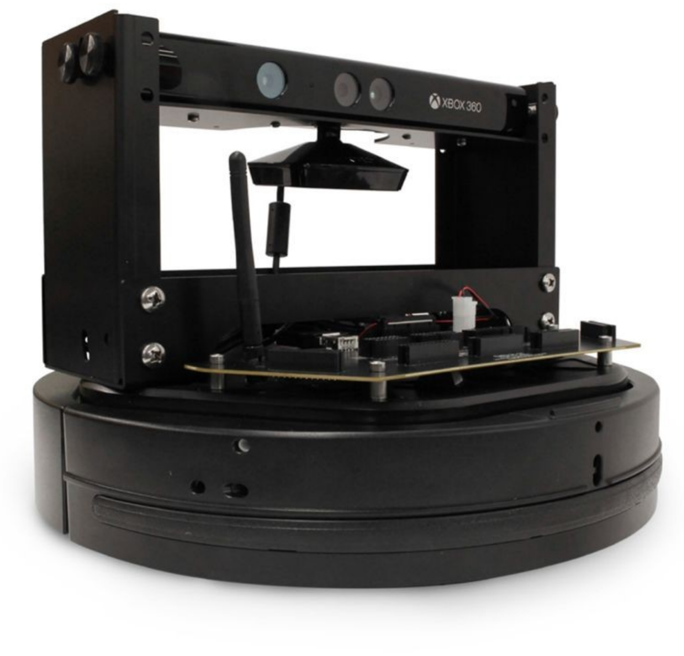


Figure 10 The Quanser QBot2 [22].

There are three analog and digital cliff sensors, left, right and centre, located below the Kobuki frame, that detect cliffs by identifying changes in the distance between the robot base and the floor. The two, left and right, wheel drop sensors identify whether a wheel has dropped or not, enabling the robot to perceive uneven ground.

3.1 WHEEL ENCODERS

The Kobuki robot platform is driven by two differential drive wheels with inbuilt high accuracy wheel encoders that sense 2578.33 ticks/wheel revolution and 11.7 ticks/mm. Wheel encoders are proprioceptive sensors used to estimate robot position over time and this process is known as Odometry. The odometry data provides an approximate position of the robot to serve as the initial estimate of where the robot might be in the SLAM process. Inaccuracies due to drift and slippage make motion estimation prone to error. For the QBot2, a differential drive robot, the motion can be modelled using *forward kinematics*.

Kinematics is the study of the mathematics of motion without considering the forces that affect the motion [23]. The forward kinematics of a differential drive robot can be derived as discussed in [24] and [1]. For a robot moving with speed V and angular velocity ω, let ϕ_r and ϕ_l be the rotational velocity of right and left motors respectively, when sampling at an interval of Δt. The QBot2 has a wheel radius r of 0.035(m) and half wheel distance *l* of 0.235/2(m). In Figure 11, I is used to define arbitrary inertial basis, therefore, X_I, Y_I axes are the axes for the global reference frame.

$$\xi_I = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} \qquad 3.1$$

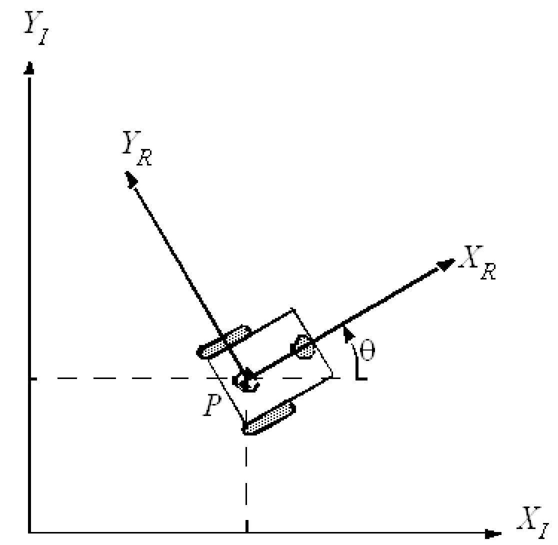


Figure 11 Posture definition in the global reference frame and the robot local reference frame for a wheeled differential drive robot [24].

where, θ is the robot's heading angle. The robot's reference frame/local reference frame R is,

$$R(\theta) = \begin{bmatrix} cos\theta & sin\theta & 0 \\ -sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad 3.2$$

Using this to map motion from global to local reference frame,

$$\dot{\xi_R} = R(\theta)\dot{\xi_I} \qquad 3.3$$

$$= \begin{bmatrix} cos\theta & sin\theta & 0 \\ -sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} \qquad 3.4$$

$$= \begin{bmatrix} \dot{x}cos\theta + \dot{y}sin\theta \\ -\dot{x}sin\theta + \dot{y}cos\theta \\ \dot{\theta} \end{bmatrix}. \qquad 3.5$$

From the forward kinematic model, the overall speed of robot in the global reference frame,

$$\dot{\xi}_I = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = f\left(l, r, \theta, \dot{\phi}_r, \dot{\phi}_l\right) \qquad 3.6$$

$$\dot{\xi}_I = R(\theta)^{-1}\dot{\xi}_R\,. \qquad 3.7$$

When the robot moves forward along the goal, if one wheel spins while the other is stationary, since P is halfway between the 2 wheels, the robot will move instantaneously with half the speed.

$$V_r = \frac{r\dot{\phi}_r}{2} \qquad 3.8$$

$$V_l = \frac{r\dot{\phi}_l}{2}\,. \qquad 3.9$$

Consider, the forward spin of right wheel results in anti-clockwise rotation,

$$\omega_r = \frac{r\dot{\phi}_r}{2l} \qquad 3.10$$

because the wheel is instantaneously moving along the arc of a circle with radius 2*l*. For the left wheel, forward spin results in clockwise rotation,

$$\omega_l = \frac{r\dot{\phi}_l}{2l}\,. \qquad 3.11$$

The kinematic model for a differential drive robot,

$$\dot{\xi}_I = R(\theta)^{-1}\begin{bmatrix}\frac{r\dot{\phi}_r}{2}+\frac{r\dot{\phi}_l}{2}\\ 0\\ \frac{r\dot{\phi}_r}{2l}-\frac{r\dot{\phi}_l}{2l}\end{bmatrix} \qquad 3.12$$

$$R(\theta)^{-1} = \begin{bmatrix}cos\theta & -sin\theta & 0\\ sin\theta & cos\theta & 0\\ 0 & 0 & 1\end{bmatrix} \qquad 3.13$$

assuming robot moves towards goal along θ.
Therefore,

$$\dot{\xi}_I = \begin{bmatrix}cos\theta & -sin\theta & 0\\ sin\theta & cos\theta & 0\\ 0 & 0 & 1\end{bmatrix}\begin{bmatrix}\frac{r\dot{\phi}_r}{2}+\frac{r\dot{\phi}_l}{2}\\ 0\\ \frac{r\dot{\phi}_r}{2l}-\frac{r\dot{\phi}_l}{2l}\end{bmatrix} \qquad 3.14$$

$$\dot{\xi}_I = \begin{bmatrix}cos\theta\left(\frac{r\dot{\phi}_r}{2}+\frac{r\dot{\phi}_l}{2}\right)\\ sin\theta\left(\frac{r\dot{\phi}_r}{2}+\frac{r\dot{\phi}_l}{2}\right)\\ \frac{r\dot{\phi}_r}{2l}-\frac{r\dot{\phi}_l}{2l}\end{bmatrix} = \begin{bmatrix}\dot{x}\\ \dot{y}\\ \dot{\theta}\end{bmatrix}. \qquad 3.15$$

This approach to kinematic modeling provides information about the motion of the robot given its component wheel speeds.

3.2 GYROSCOPE

Gyroscopes are passive proprioceptive heading sensors used to determine the robot's orientation and inclination. They measure angular acceleration and therefore can be used to measure the robot's orientation in relation to a fixed reference frame [23]. The QBot2 uses a L3G4200D 3-axis digital gyroscope manufactured by STMicroelectronics [25]. It measures in the ±250 degrees/s range where the Yaw axis (angular rotation along the Z-axis) is factory calibrated within the range of ±20 deg/s to ±100 deg/s. It is also relatively immune to shocks and vibrations [25]. However, gyroscopes accumulate small measurement errors over time resulting in gradual loss of accuracy.

3.3 VISUAL SENSORS

Recently visual sensors have grown in popularity in robotic applications. A visual sensor or camera captures an image of the surrounding space and this digital image is processed in order to obtain information like depth, motion, colour and feature [1]. Digital cameras capture light and process it into digital images. Placing red, green or blue filters over the images allows them to measure light intensity of each colour and produce a colour image. Ranging is an important aspect in sensing as it provides information vital to collision avoidance. Generally, cameras measure scene structure as opposed to the distance between the object and the robot in lasers and sonars [23]. Depth sensors in cameras make ranging and recreating 3D objects from the 2D images easier.

3.3.1 Microsoft Kinect Sensor

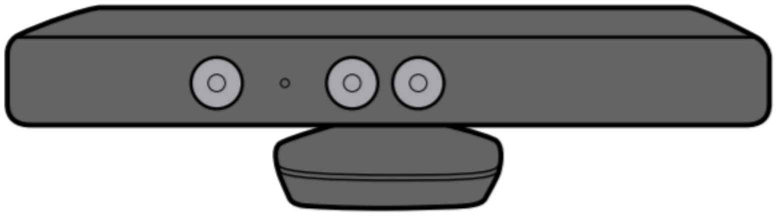


Figure 12 Xbox 360 Kinect sensor [26].

The QBot2 comes equipped with a Microsoft Kinect sensor, as shown in Figure 12, that includes an RGB camera for capturing image data as well as a 3D depth sensor that captures 11-bit depth data at the resolution of 680 × 480 and several frame-rates [22]. The depth sensor consists of a monochrome infrared camera and an infrared projector which evaluates distance from each point on an object based on the time of flight of infrared light. As presented in [22], the depth sensor has a range of 0.5 m to 6 m. Once it is mounted on the QBot2, the Kinect sensor can be tilted up and down whenever needed as shown in Figure 13. The QBot2 Kinect sensor is optimal for indoor applications and in locations without direct sunlight due to the type of infrared sensor used.

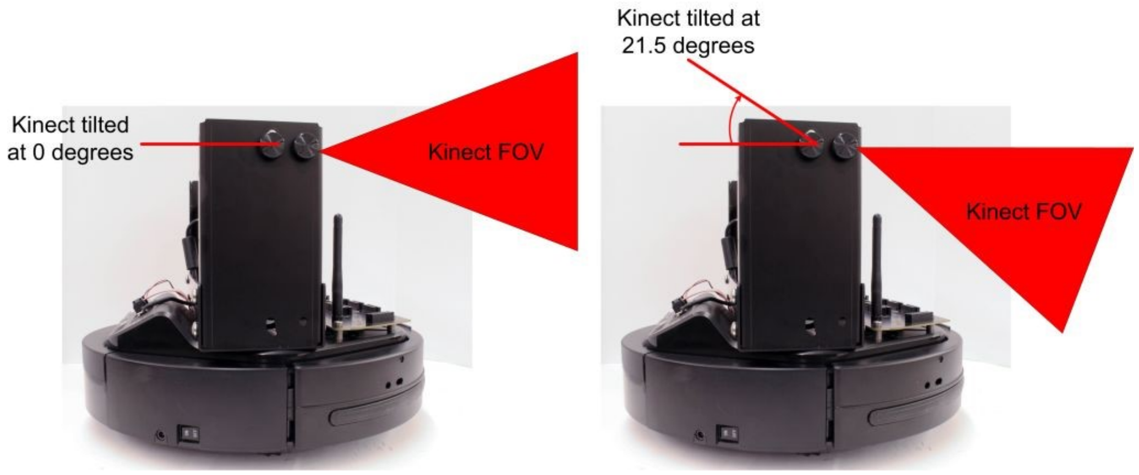


Figure 13 QBot2 with tilted Kinect sensor [22].

3.3.2 Visual Odometry

The process of determining robot position using a sequence of images is called Visual Odometry [27]. It can be used for map building and obstacle avoidance even in spaces where little or no a-priori knowledge is available. Visual Odometry was successfully implemented for the first time in [28] using monocular (one camera) and stereo (two cameras) setups. When using two cameras the measurements can be obtained directly in the global reference frame. However, when using a single camera, the global reference frame needs to be determined in collusion with other sensors or from a-priori knowledge of objects and surroundings [1]. In [28], it was observed that the stereo vision scheme works better when compared to the monocular scheme as it can operate efficiently even without camera motion. Like wheel encoders based odometry, visual odometry also inevitably suffers from motion drift over time, however, this can be overcome when the robot revisits an area. When compared to other sensors, vision sensors are capable of place or location recognition which is like loop closure, commonly discussed in the SLAM problem.

3.3.3 Feature Detection

When sensors, including cameras, detect objects, they identify either appearance or features. Most common odometry and SLAM techniques use feature detection as opposed to appearance detection since detecting features is faster and more accurate than detecting appearances [29]. The two feature detection techniques reviewed in this section are Scale Invariant Feature Transform or SIFT and Speeded Up Robust Features or SURF.

SIFT was introduced in [30] as a method to generate image features that transform an image into a large collection of local feature vectors, invariant to translation, rotation and scaling and partially invariant to illumination changes and 3D projection. As explored in [31] there are four major stages in generating image features.

Scale space extrema detection:
Identifying locations and scales that can be repeatedly assigned under different views of the same object enables Keypoint detection. The scale space of an image is defined as the function:

$$L(x, y, \sigma) = G(x, y, \sigma) * I(x, y) \qquad 3.16$$

where, * is the convolution operation in *x* and *y* and *I(x, y)* is the input image. The variable-scale Gaussian is given by:

$$G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{-(x^2+y^2)/2\sigma^2}. \qquad 3.17$$

The difference between two nearby scales, separated by a constant multiplicative factor *k,* enables efficient detection of stable keypoint locations in scale space,

$$\begin{aligned} D(x, y, \sigma) &= \big(G(x, y, k\sigma) - G(x, y, \sigma)\big) * I(x, y) \\ &= L(x, y, k\sigma) - L(x, y, \sigma). \end{aligned} \qquad 3.18$$

The values of the difference-of-Gaussian function and the scale-normalized Laplacian of Gaussian, $\sigma^2\nabla^2 G$, are approximately close to each other. Here σ^2 provides true scale invariance. From the heat diffusion equation:

$$\frac{\partial G}{\partial \sigma} = \sigma \nabla^2 G. \qquad 3.19$$

Therefore,

$$\sigma \nabla^2 G = \frac{\partial G}{\partial \sigma} \approx \frac{G(x, y, k\sigma) - G(x, y, \sigma)}{k\sigma - \sigma} \qquad 3.20$$

and,

$$G(x, y, k\sigma) - G(x, y, \sigma) \approx (k-1)\sigma^2 \nabla^2 G. \qquad 3.21$$

This shows that the difference-of Gaussian is already a scale-invariant Laplacian.

Keypoint localization:

Once a keypoint has been identified, location, scale and ratio of principal curvatures need to be determined to reject low contrast points or points poorly localized on edges. For the sample point to be the origin, the Taylor expansion of scale space function, *D(x, y, σ)*, is used:

$$D(x) = D + \frac{\partial D}{\partial \mathbf{x}}^T \mathbf{x} + \frac{1}{2}\mathbf{x}^T \frac{\partial^2 D}{\partial \mathbf{x}^2}\mathbf{x} \qquad 3.22$$

where, **x** = $(x, y, \sigma)^T$ is offset from the sample point. The local extremum is,

$$\hat{\mathbf{x}} = -\frac{\partial^2 D}{\partial \mathbf{x}^2}^{-1} \frac{\partial D}{\partial \mathbf{x}}\ . \qquad 3.23$$

For rejecting unstable extrema,

$$D(\hat{\mathbf{x}}) = D + \frac{\partial D}{\partial \mathbf{x}}^T \hat{\mathbf{x}} \qquad 3.24$$

is obtained by substituting equation 3.23 into equation 3.22.

Poorly determined locations along edges need to be eliminated as they are susceptible to small noises. The difference-of-Gaussian function will have a large principal curvature across the edge but a small perpendicular one, which can be determined by,

$$\mathbf{H} = \begin{bmatrix} D_{xx} & D_{xy} \\ D_{xy} & D_{yy} \end{bmatrix} \qquad 3.25$$

where, the components of the Hessian matrix **H** are estimated by the differences between the neighbouring sample points. The sum and product of eigenvalues are computed from the trace and determinant of **H** respectively as,

$$Tr(\mathbf{H}) = D_{xx} + D_{yy} = \alpha + \beta \tag{3.26}$$

$$Det(\mathbf{H}) = D_{xx} + D_{yy} - \left(D_{xy}\right)^2 = \alpha\beta \tag{3.27}$$

where, α is the largest magnitude eigenvalue and β is the smaller one. The point will be discarded for not being an extremum when the determinant is negative.

$$\frac{Tr(\mathbf{H})^2}{Det(\mathbf{H})} = \frac{(\alpha + \beta)^2}{\alpha\beta} = \frac{(r+1)^2}{r} \tag{3.28}$$

where the ratio r = α/β. The below equation is verified to check if the ratio of principal curvatures is below threshold,

$$\frac{Tr(\mathbf{H})^2}{Det(\mathbf{H})} < \frac{(r+1)^2}{r}\ . \tag{3.29}$$

Orientation assignment:

To accomplish image rotation invariance, a consistent orientation is assigned to each keypoint. For scale-invariant computations, the keypoint scale is used to select the closest scale Gaussian smoothed image, L. Using pixel differences, the gradient magnitude is precomputed as,

$$m(x,y) = \sqrt{(L(x+1,y) - L(x-1,y))^2 + \left(L(x,y+1) - L(x,y-1)\right)^2} \tag{3.30}$$

and the orientation as,

$$\theta(x,y) = \tan^{-1}\left(\frac{L(x,y+1) - L(x,y-1)}{L(x+1,y) - L(x-1,y)}\right) \tag{3.31}$$

where L(x,y) is the image sample.

Keypoint descriptor:

In the above stages, a keypoint has been assigned an image location, scale and orientation. Finally, a highly distinctive and invariant descriptor for the local image region is computed that makes the keypoint immune to light and 3D viewpoint changes as detailed in [31].

SURF was introduced in [32] as a faster technique for feature detection that balances reducing dimensions and complexity with maintaining distinction. SURF constitutes a detector and a descriptor that draws heavily from SIFT. The Hessian matrix-based detector uses a basic approximation, similar to the difference-of -Gaussian function in SIFT, that relies on integral images for image convolutions to reduce the computation time. The SIFT descriptor is distinctive and relatively fast; however, high dimensionality makes the matching step inefficient. Therefore, the SURF descriptor describes a distribution of Haar-wavelet responses within the neighbourhood of interest point and exploits integral images for speed. The dimensions are restricted to 64 and based on the sign of the Laplacian a new indexing step is presented to reduce feature computation and matching time and increase robustness. A further improvement suggested to SURF is Upright-SURF, which is suited for applications with horizontally positioned cameras, is even faster to compute.

3.4 ROBOT SIMULATION

A TurtleBot2 as shown in Figure 14, uses a Kinect 360 RGB-D sensor like the QBot2, which means their cameras have the same resolutions and depth sensors. Because both robots use a Kobuki base, same range of sensors like the cliff sensor, bumper sensor, wheel encoders and gyroscope, and a similar payload of approximately 4.5 kg to 5 kg, the TurtleBot2 robot is a good stand-in for the QBot2. Therefore, for easier implementation we use the pre-existing TurtleBot2 ROS packages instead of creating new simulation packages for a QBot2. The only minor differences are that the QBot2 uses a Raspberry Pi whereas the TurtleBot2 uses Intel Core i3-4010U as their onboard computers and the TurtleBot2 is taller than the QBot2..

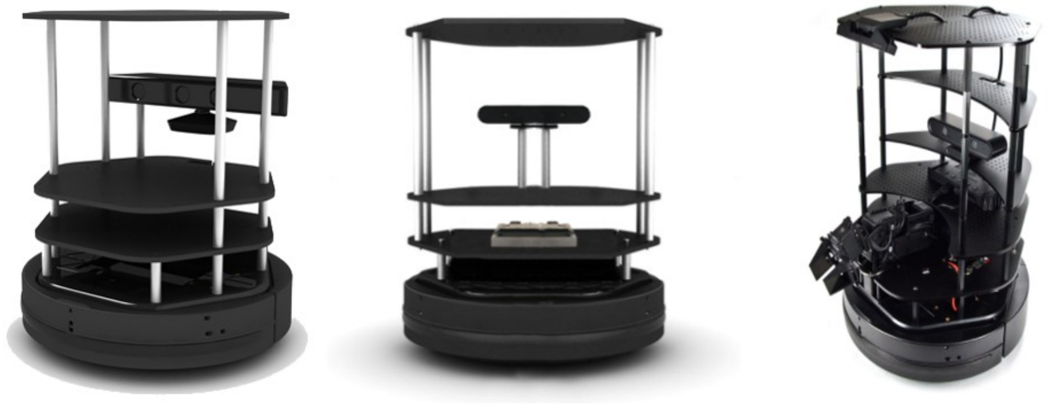


Figure 14 TurtleBot2, TurtleBot 2e and TurtleBot 2i [33].

CHAPTER 4 ROS SETUP

This chapter discusses the step by step instructions and the commands for the installation and setup of

- Oracle VirtualBox VM
- ROS- Installation and Programming
- ROS packages for simulation experiments
- ROS packages for QBot2 experiments.

4.1 CREATING VIRTUALBOX VM AND ROS INSTALLATION

To setup ROS for the purpose of the simulation portion of this research, multiple Oracle VirtualBox VMs were created using the following steps:

- Oracle VM VirtualBox version 6.1 was downloaded and installed using instructions from [34]. Ubuntu 16.04 LTS was downloaded from [35].
- A new VM was created in the VirtualBox with 1 core CPU, 2 GB RAM and 30 GB fixed hard disk. As detailed in [36] the VM was powered on and Ubuntu 16.04 LTS was installed as the operating system on the VM. This step was repeated to created multiple VMs so the different experiments could be conducted simultaneously.
- The Kinetic Kame version of ROS was installed on these VMs along with its dependencies and Ubuntu updates by following the instructions from table 3. Kinetic Kame is the recommended ROS version for Ubuntu 16.04 LTS. It has extended support till 2021 and with comprehensive documentation available.

Table 3 ROS Kinetic Installation Instructions [10].

```
$    sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) main" >
     /etc/apt/sources.list.d/ros-latest.list'
$    sudo apt-key adv --keyserver 'hkp://keyserver.ubuntu.com:80' --recv-key
     C1CF6E31E6BADE8868B172B4F42ED6FBAB17C654
$    sudo apt-get update
$    sudo apt-get install ros-kinetic-desktop-full
$    echo "source /opt/ros/kinetic/setup.bash" >> ~/.bashrc
$    source ~/.bashrc
$    sudo apt install python-rosdep python-rosinstall python-rosinstall-generator
     python-wstool build-essential
$    sudo apt install python-rosdep
$    sudo rosdep init
$    rosdep update
```

4.2 PROGRAMMING WITH ROS

Before programming in ROS, a workspace needs to be created. The folder called ROS workspace or *catkin workspace* is where new ROS packages are created, existing packages are installed, and new executables are built and created [36]. *catkin* is a set of tools that ROS uses to generate programs, libraries, scripts and interfaces that other code can use [1]. To create a *catkin* workspace use [10]:

```
$ mkdir -p ~/catkin_ws/src
$ cd ~/catkin_ws/
$ catkin_make
```

A ROS package is a combination of code, data and documentation [7]. Instead of having to change the source directory to the *catkin* workspace created above, every time a new ROS package is created or an existing ROS package is edited, source the *catkin_ws* and overlay it on your environment using:

```
$ source devel/setup.bash
```

One useful ROS package for programming robots is the *tf package* where *tf* stands for transform. A robotic system typically has many 3D coordinate frames such as a world frame, base frame, head frame, etc., that change over time as shown in Figure 15. The RGB cylinders in the figure represent the X, Y, and Z axes of the coordinate frames. *tf* keeps track of all these frames and maintains the relationship between coordinate frames in a tree structure buffered in time [10].

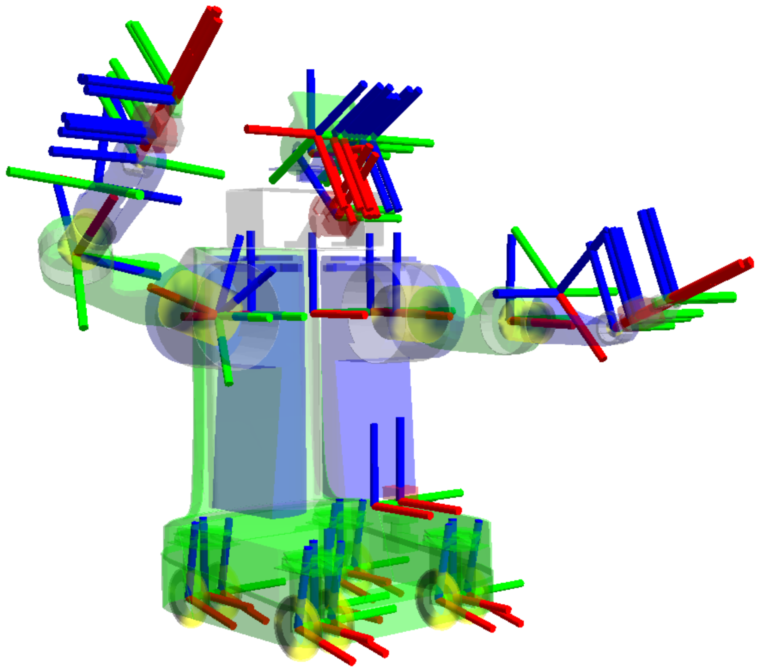


Figure 15 The different 3D coordinate frames of Willow Garage's PR2 robot [37].

The *tf* library was designed to provide a standard way to keep track of coordinate frames and transform data from one system to another, in such a way that individual component users can be confident that the data is in the coordinate frame that they want, without requiring knowledge of all the coordinate frames in the system [37].

4.3 ROS PACKAGES USED

4.3.1 Gazebo

Gazebo is a tool in which 3D environments along with different types of robots can be simulated with a high degree of precision [38]. The *gazebo_ros* package provides an interface between Gazebo and ROS. It incorporates many, ready to work, simulated sensors and robots and allows usage of existing standard ROS plugins for Gazebo. Being able to control the world and the robot with command-line interface makes it the ideal choice for this research. Currently Gazebo version 11.0 is available, however, ROS *kinetic* supports Gazebo7 and comes preinstalled with Gazebo 7.0. For seamless operation Gazebo7 needs to be updated to the latest version, which is currently version 7.16, using the commands in Table 4.

Table 4 Commands to update Gazebo from version 7.0 to 7.16.

```
$  sudo sh -c 'echo "deb http://packages.osrfoundation.org/gazebo/ubuntu-stable
   `lsb_release -cs` main" > /etc/apt/sources.list.d/gazebo-stable.list'
$  wget http://packages.osrfoundation.org/gazebo.key -O - | sudo apt-key add -

$  sudo apt-get update && sudo apt-get install gazebo7 -y
```

4.3.2 Turtlebot

Install the ROS packages required to use TurtleBot2 with Gazebo using the following commands [33]:

$ sudo apt-get install ros-kinetic-turtlebot ros-kinetic-turtlebot-apps ros-kinetic-turtlebot-interactions ros-kinetic-turtlebot-simulator ros-kinetic-kobuki-ftdi ros-kinetic-ar-track-alvar-msgs

To be able to access the code and make changes to the Turtlebot package, it is built from source following the instructions in Table 5.

Table 5 Commands to build Turtlebot package from source in ROS Kinetic [39].

```
$  mkdir ~/rocon && cd ~/rocon
$  wstool init -j5 src https://raw.github.com/robotics-in-
   concert/rocon/release/indigo/rocon.rosinstall
$  source /opt/ros/indigo/setup.bash
$  rosdep install --from-paths src -i -y
$  catkin_make
```

```
$  mkdir ~/kobuki
$  cd ~/kobuki
$  wstool init src -j5
   https://raw.github.com/yujinrobot/yujin_tools/master/rosinstalls/indigo/kobuki.rosi
   nstall
$  source ~/rocon/devel/setup.bash
$   rosdep install --from-paths src -i -y
$   catkin_make
```

```
$   mkdir ~/turtlebot
$   cd ~/turtlebot
$   wstool init src -j5
    https://raw.github.com/yujinrobot/yujin_tools/master/rosinstalls/indigo/turtlebot.ro
    sinstall
$   source ~/kobuki/devel/setup.bash
$   rosdep install --from-paths src -i -y
$   catkin_make
```

The *rocon*, *kobuki* and *turtlebot* workspaces created using the commands from the above table are chained and source information from each other.

4.3.3 rtabmap and rtabmap_ros

The *rtabmap* and *rtabmap_ros* packages [40] build a step-by-step map of the environment that is then optimized by loop closure detection, as illustrated by Figure 16. By default, the maps are stored in "~/.ros/rtabmap.db". *The rtabmap_ros* package contains the parameters for ROS and RTAB-Map. The ROS parameters connect the RTAB-Map library, which constitutes RTAB-Map parameters, with ROS. Like the turtlebot package in the previous section, to be able to access the code and make changes to the *rtabmap_ros* package, it is built from source as shown in Table 6. To install the dependencies for these packages, use the following commands:

$ sudo apt-get install ros-kinetic-rtabmap ros-kinetic-rtabmap-ros

$ sudo apt-get remove ros-kinetic-rtabmap ros-kinetic-rtabmap-ros

Table 6 Commands to build rtabmap and rtabmap_ros packages from source in ROS Kinetic [40].

```
$   cd ~
$   git clone https://github.com/introlab/rtabmap.git rtabmap
$    cd rtabmap/build
$    cmake ..
$    make
$    sudo make install
```

```
$    cd ~/catkin_ws
$    git clone https://github.com/introlab/rtabmap_ros.git src/rtabmap_ros
$    catkin_make -j1
```

4.3.4 rviz

RViz is ROS's 3D visualising tool. This is where the robot's movements and map building process can be observed in real time. To transform data from the robot's local reference frame to the global reference, *rviz* uses the *tf* package. The command [41] used to install *rviz* is:

$ sudo apt-get install ros-kinetic-rviz

It also allows users to interactively set intermediate goals and exploration boundaries for the robot using the 2D Nav goal setting.

4.3.5 explore_lite

To automate the robot movement, instead of having to continuously interact with the robot using the keyboard or a joystick as shown in section 4.4, we use the *explore_lite* ROS package. It is a frontier-based approach for autonomous exploration [42], where based on the information of the world obtained via images from Kinect sensor the algorithm processes intermediate goals for the robot. As detailed in [42], a robot will start at the origin and will look for frontiers, i.e., boundaries between unknown and unoccupied spaces in the environment. Once frontiers are detected, the nearest possible frontier will be set as the intermediate goal for the robot. For path planning, this package uses a depth first search to find the shortest path from the robot's current location to its intermediate goal. Once a frontier has been visited, it is added to a list of known locations. *explore_lite* allows the robot to search its world greedily until no new frontier is detected. As shown in Figure 16, the blue arrows indicate unexplored areas or frontiers in the environment. The *explore_lite* package [43] is more cost efficient than other similar packages as it obtains map information from the *rtabmap_ros* package and odometry information from the *turtlebot* package. Using this data, it sends commands to the robot base for movement, so it does not need to create its own map. To install explore_lite the command below is used:

```
$ sudo apt install ros-kinetic-multirobot-map-merge ros-kinetic-explore-lite
```

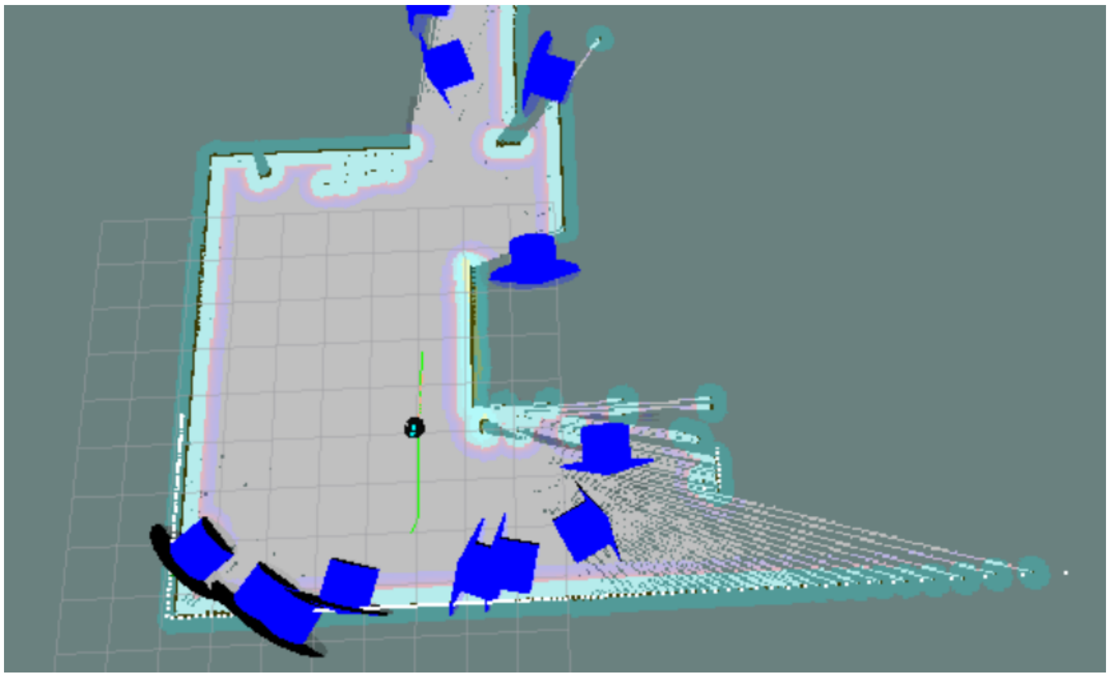


Figure 16 Unexplored frontiers detected by explore_lite package [43].

4.4 PACKAGES FOR QBOT2 TESTING

To setup the platform to test QBot2 robot, an Ubuntu laptop with 64-bit 16.04 LTS OS, 8.00 GB RAM, 256 GB SSD storage, Intel Core i5 7200 2.50 GHz to 2.71 GHz processor was used. Following the instructions from Table 2 ROS Kinetic Kame was installed and from section 4.2 a *catkin_ws* was created. Following the instructions from tables 6 and 7 the turtlebot, *rtabmap* and *rtabmap_ros* packages were built from source. The *explore_lite* package was also installed to enable autonomous navigation with the QBot2.

4.4.1 Kinect And Kobuki Software Installation

To access the XBOX 360 Kinect sensor in Ubuntu, *freenect_stack* is built from source as shown in table 7. This acts as a ROS interface to Microsoft Kinect using the libfreenect library. It contains the *freenect_camera* and *freenect_launch* packages which in turn contain the camera drivers and launch files, respectively.

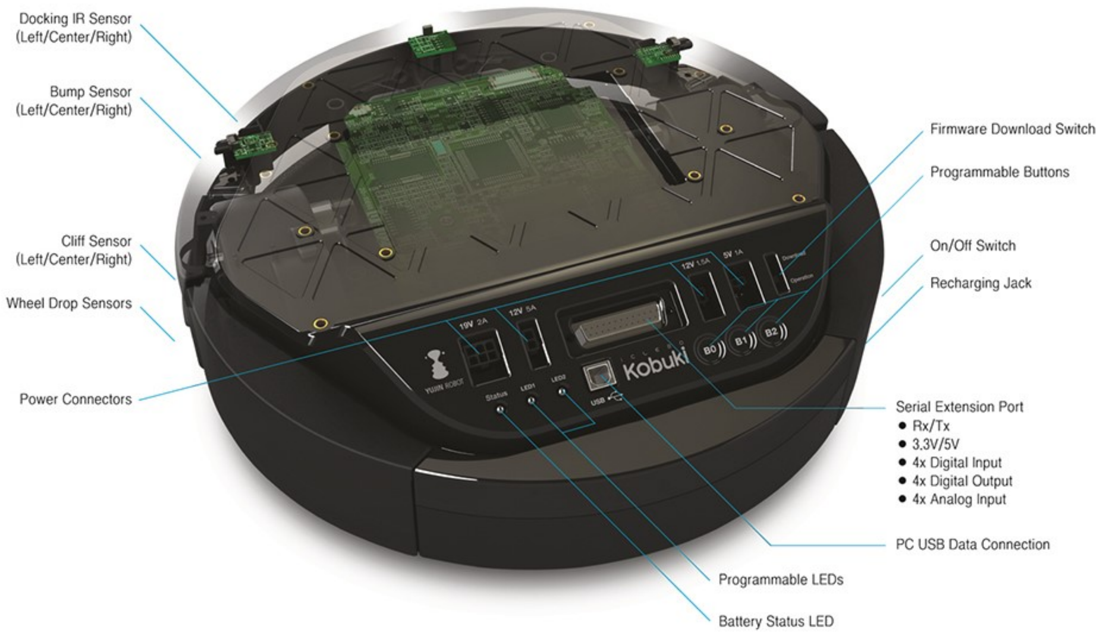


Figure 17 The Kobuki mobile robot platform [22].

Table 7 Instructions for Freenect Installation in ROS Kinetic [44].

```
$   cd ~/catkin_ws/src
$   git clone https://github.com/ros-drivers/freenect_stack/tree/master
$   cd ..
$   catkin_make -j1
```

The QBot2 is seated on a Kobuki base like the one shown in Figure 17, therefore Kobuki ROS packages need to be installed in the ubuntu laptop for connection to the QBot2 using a USB A-B cable. Table 8 contains the list of commands for Kobuki installation and controlling the QBot2 using the laptop keyboard.

Table 8 Instructions for Kobuki Installation in ROS Kinetic [39].

```
$   sudo apt-get install ros-kinetic-kobuki ros-kinetic-kobuki-core
$   sudo usermod -a -G dialout $USER
$   rosrun kobuki_ftdi create_udev_rules
    # Insert Kobuki's USB cable and open a new shell
$   . /opt/ros/kinetic/setup.bash
$   roslaunch kobuki_node minimal.launch --screen
    # In a second shell
$   . /opt/ros/kinetic/setup.bash
$   roslaunch kobuki_keyop keyop.launch --screen
```

4.4.2 Testing Connections and Sensors

In preparation for conducting the experiments on QBot2, some basic initial tests were performed. The QBot2 was tested in random walker mode B0 to confirm operating condition. As discussed in [39], to test the connections and sensors the following commands were executed.

- To check if the bumpers are working, one of the bumpers was tapped:
  $ rostopic echo /mobile_base/events/bumper
- To check the wheel drop sensors, the QBot2 was lifted and then put down:
  $ rostopic echo /mobile_base/events/wheel_drop
- To confirm that the QBot2 IMU (Gyroscope) is working and the robot can detect its orientation, the robot was turned:
  $ rostopic echo /mobile_base/sensors/imu_data
- To make the robot move around using the laptop keyboard:
  $ roslaunch kobuki_node minimal.launch --screen
  $ roslaunch kobuki_keyop safe_keyop.launch –screen

When using *safe_keyop* command the QBot2 uses the bumpers, cliff and wheel drop sensors to ensure safe operation.

Furthermore, to test if the RGBD sensor works, connect the USB cable from the camera and execute the following command:

```
$ roslaunch freenect_launch freenect.launch depth_registration:=true
```

In the logs ‘Xbox NUI Camera(2ae) from Microsoft’ should show up as a device, in another terminal execute:

```
$ roslaunch rtabmap_ros rtabmap.launch rtabmap_args:="--delete_db_on_start"
```

The RTAB-Map application should open-up with a window displaying the camera output.

CHAPTER 5 RTAB-MAP AND GENETIC ALGORITHM

In this chapter, the RTAB-Map package and the parameters chosen for optimisation are discussed. Also, Genetic Algorithm is introduced along with a glimpse of the first GA SLAM and the MOGA. As the proposed approach employs a MOGA to optimise RTAB-Map parameters so that the map generated as a result is more efficient and truer to the environment, the objectives used to optimise RTAB-Map parameters are explored briefly.

5.1 RTAB-MAP

RTAB-Map SLAM package uses inputs from wheel odometry and a RGBD camera to create a map. While Hector SLAM and GMapping can be programmed to use the depth sensor in the RGBD camera as a fake LIDAR or a fake laser respectively, RTAB-Map works with a camera out of the box which makes it ideal to work with a QBot2 robot.

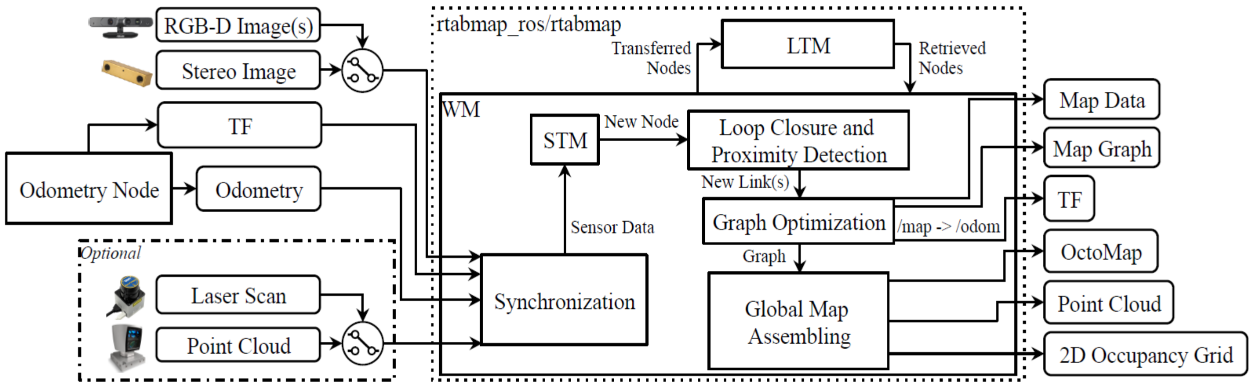


Figure 18 Block diagram of RTAB-Map ROS node [21].

As more areas of the environment are explored, the time required to process the data to assemble the map also increases. In RTAB-Map to avoid such delays, a memory management technique, depicted in Figures 9 and 18, is used where information from sensors is obtained by the perception module and sent to the sensory memory. Here the image is allotted a signature using the bag-of-words approach and a local map is created. [20] and [21] detail how the short-term memory is used to observe similarities between

]

consecutive images and the working memory is used for loop closure detection. To make loop closures less time-consuming, locations that are less likely to cause loop closures are transferred to the long-term memory. In RTAB-Map the global map is assembled online, therefore these local maps are added to its cache and global occupancy grid is updated with new poses periodically. The overall process to detect loop closures is outlined in Table 9.

Table 9 RTAB-Map Algorithm [20].

```
1: $time \leftarrow \textsc{TimeNow}()$          ▷ $\textsc{TimeNow}()$ returns current time
2: $I_t \leftarrow$ acquired image
3: $L_t \leftarrow \textsc{LocationCreation}(I_t)$
4: if $z_t$ (of $L_t$) is a bad signature (using $T_{\text{bad}}$) then
5:     Delete $L_t$
6: else
7:     Insert $L_t$ into STM, adding a neighbor link with $L_{t-1}$
8:     Weight Update of $L_t$ in STM (using $T_{\text{similarity}}$)
9:     if STM's size reached its limit ($T_{\text{STM}}$) then
10:        Move oldest location of STM to WM
11:    end if
12:    $p(S_t|L^t) \leftarrow$Bayesian Filter Update in WM with $L_t$
13:    Loop Closure Hypothesis Selection ($S_t = i$)
14:    if $S_t = i$ is accepted (using $T_{\text{loop}}$) then
15:        Add loop closure link between $L_t$ and $L_i$
16:    end if
17:    Join $trash$'s thread          ▷ Thread started in $\textsc{Transfer}()$
18:    $\textsc{Retrieval}(L_i)$          ▷ LTM $\rightarrow$ WM
19:    $pTime \leftarrow \textsc{TimeNow}() - time$          ▷ Processing time
20:    if $pTime > T_{\text{time}}$ then
21:        $\textsc{Transfer}()$          ▷ WM $\rightarrow$ LTM
22:    end if
23: end if
```

The *rtabmap* package provides a vast list of parameters that can be modified for an optimal mapping and localization depending on the environment and hardware. For the purpose of this research, the following parameters are considered:

- Mem/RehearsalSimilarity: As explained in [19] Rehearsal is the number of times a location has been matched or consecutively viewed. Hence, Rehearsal Similarity is the ratio between the number of matched word pairs between the locations being compared and the total number of words in the signature of the location with more words. It sets the threshold which determines if a location is unique enough to be kept in the short-term memory or very much like another location that they can be merged. The parameter key 'Mem/' in its name implies that changing this parameter would make changes to the memory of the framework. Rehearsal similarity ranges from 0 to 1 and is by default set to 0.3 in the *rtabmap* package.

- Kp/MaxDepth: Max Depth filters extracted keypoints by depth. The parameter key 'Kp/' in its name implies that this parameter is keypoint based. A feature's depth is estimated with depth image, changing the MaxDepth will change how loop closures are detected. Depending on the number of loop closures that are detected, the robot's trajectory might be different. MaxDepth can range from 0 to infinity and is by default set to 4.0m.
- Vis/MinInliers: It is the minimum number of feature correspondences required to accept a loop closure. The parameter key 'Vis/' indicates that this is a Visual registration parameter. Decreasing the value of Vis/MinInliers can help to accept more localizations whereas increasing it would increase accuracy at the cost of less localizations in cases with less matching visual features. By default, it is set to 15 in the *rtabmap* versions 0.19.6 and 0.20.3.

5.2 GENETIC ALGORITHM

Genetic Algorithms are optimisation algorithms that mimic natural biological evolutionary behaviour and are based on the Darwinian concept of "competition for a finite resource". Genetic programming (GP) is defined as the direct evolution or breeding of computer programs for the purpose of intuitive learning [45]. GPs might be linear (Linear Genetic Programming), tree (Syntactic Closure) or graph-based GP (Neural Network). In a GA, as illustrated in Figure 19, every generation has its own fixed population, and a selection operator decides which individuals get to be the parents in each generation. A variation operator, like crossover or mutation, is used to produce offspring relative to parents. Replacement operator is a type of selection operator that replaces the current population with the children from the previous step, to be the population for the next generation.

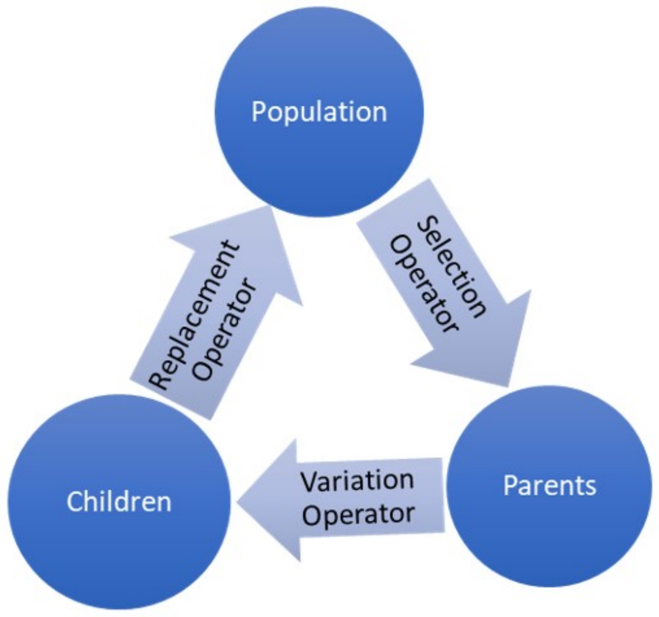


Figure 19 The Generic Evolutionary Model.

5.2.1 Selection Operator

At initialisation of a GA, the population is a random sample of candidate solutions. Selection operators define processes for selection and replacement based on Darwin's natural selection. Fitness function is used to select individuals from the population and compare them with respect to their fitness. The *fitness* measure defines the problem the algorithm is expected to solve [45]. As candidates are evaluated, information is retained regarding where the search effort should be directed, thus biasing the selection of individuals, the way children are created and the formulation of a new population. Efficiency of the process is dependent on the ability to utilize partial information gleaned from the distribution of individuals and their fitness across the search landscape.

5.2.2 Variation Operator

In genetic algorithm, variation operators direct the creation of new individuals. Variation operators can be of three types: Local search – too expensive in practice, Crossover, also known as recombination, and Mutation. In crossover substructures are exchanged between two, in some cases more individuals resulting in as many children.

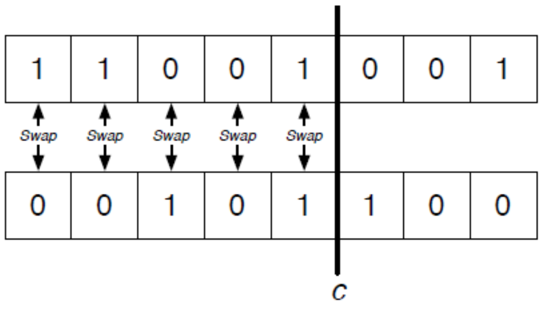


Figure 20 Single-point string Crossover [46].

It cannot introduce new genetic material that is not in the gene pool of the population, therefore, is exploitative. It assumes that the material necessary for an optimal solution exists in the population. Crossover can be single-point or multi-point. *Single-point crossover* is when a point is chosen on the strings and everything after that point is crossed over, so that everything in the first string after the point ends up in the second string and vice-versa, as shown in Figure 20. Figure 21 shows crossover in tree-based GP where subtrees in parents are selected and exchanged and Figure 22 illustrates the *two-point string crossover* used in LGP for recombining two individuals.

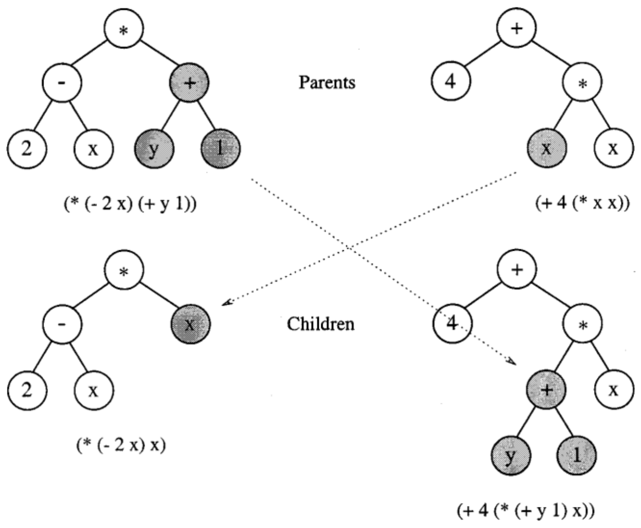


Figure 21 Crossover in tree-based GP [45].

A segment is selected in each of the two parents and exchanged. If one of the resulting children exceeds the maximum length, crossover is aborted and restarted with exchanging equally sized segments [45]. When using crossover, a solution associated with one peak in the search space can be replaced by the offspring associated with a completely different peak, which can potentially result in the loss of an entire mode of search space.

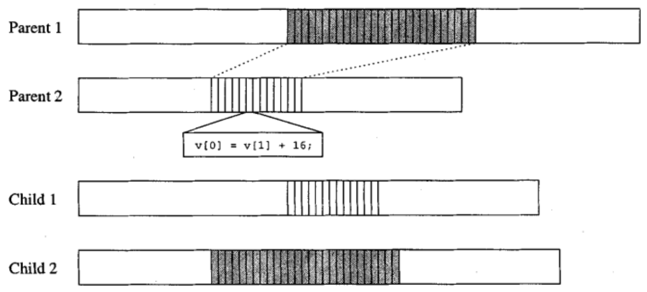


Figure 22 Two-point string Crossover in LGP [45].

The *mutation* operation, on the other hand, randomly replaces a substructure, i.e., a waypoint, a sub-path, etc. Mutation Operates on a single individual, the parent, producing a single new individual, the child. It generates new individuals whose representation (potentially) does not currently exist in the population, i.e., mutation can introduce new genetic material, hence it is explorative. Introduction of the new genetic material is regulated by controlling the probability of application.

5.2.3 Objective

The fitness function and the objective for any GA form its cost function. The objective finds the value of *x* that maximises the function $y = f(x)$ and a problem can be defined with a single objective or multiple objectives. When dealing with Single Objective Optimisation (SOO) the aim is to find the best or approximately the best solution available [47], for example the setups in [48], [49] and [50]. However, most real-world problems have multiple objectives, which are sometimes in conflict with each other, that require optimisation, like for the construction of a bridge minimum cost and maximum safety is needed [47]. These problems with two or more objective functions are called “multi-objective’’, like the setup in [2] that discusses Multi-Objective Optimisation (MOO).

5.3 FIRST GENETIC ALGORITHM SLAM

Enhancing SLAM using a GA was introduced in [48] to consider SLAM as a global optimisation problem, where the setup used a Pioneer I mobile robot with a SICK laser scanner. Using data from a lone laser sensor the robot perceived the environment and built the map illustrated in Figure 23. This odometer trace was divided into segments of fixed length and for each segment $\boldsymbol{k}$**,** the chromosome contained two floating point numbers $-d_{max} \leq \Delta\delta_k \leq +d_{max}$ and $-a_{max} \leq \Delta\alpha_k \leq +a_{max}$, that encode the correction factors applied to the distance *d*, and angle *a*, measurements respectively. The initial population of chromosomes was obtained by randomly initializing the values of $\Delta\delta_k$ and $\Delta\alpha_k$**,** which were derived from the robot’s odometry and were assumed to lie within a range of ±2% for both the distances and angles. This data was combined with the recorded robot laser data to build an occupancy grid [51]. For each cell *i* in the grid, the number of laser readings, occ_i, to find if a cell was occupied and the number of readings, emp_i, which indicate if a cell was empty were determined to calculate the following heuristics:

$$MC_1 = \Sigma_i \min(occ_i, emp_i) \tag{5.1}$$

where MC_1 was Map Consistency, and

$$MC_2 = (x_{max} - x_{min}) \times (y_{max} - y_{min}) \tag{5.2}$$

where MC_2 was Map Compactness measured in the number of grid cells and x_{max} and x_{min} were the maximum and minimum x-coordinates of the bounding box respectively. These heuristics were then combined to create the fitness function, *F*, for each map:

$$F = MC_1 + wMC_2 \tag{5.3}$$

here, *w* was the weight that determined the relative importance factor of the two heuristics. The population was then sorted according to its fitness and each string, *l*, was assigned an offspring count, e_l, based on its rank.

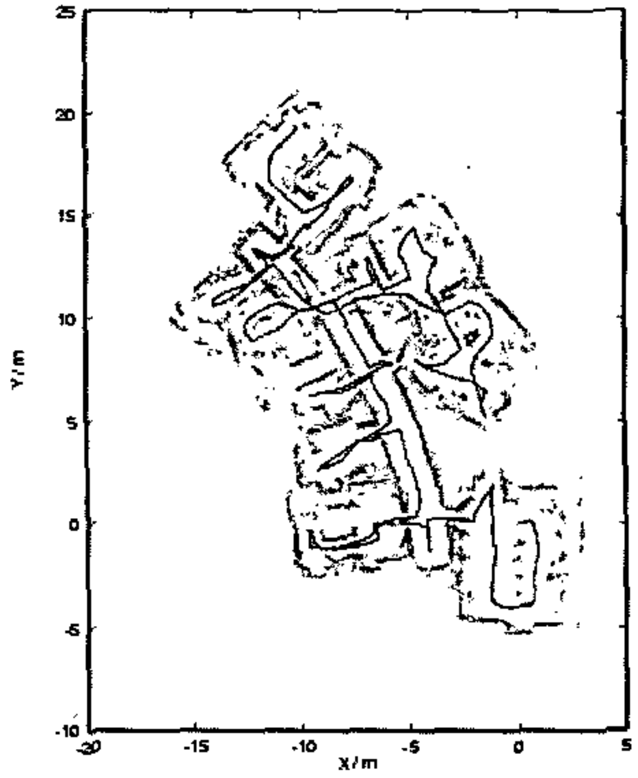


Figure 23 Raw sensor data with the odometry trace and robot laser findings [48].

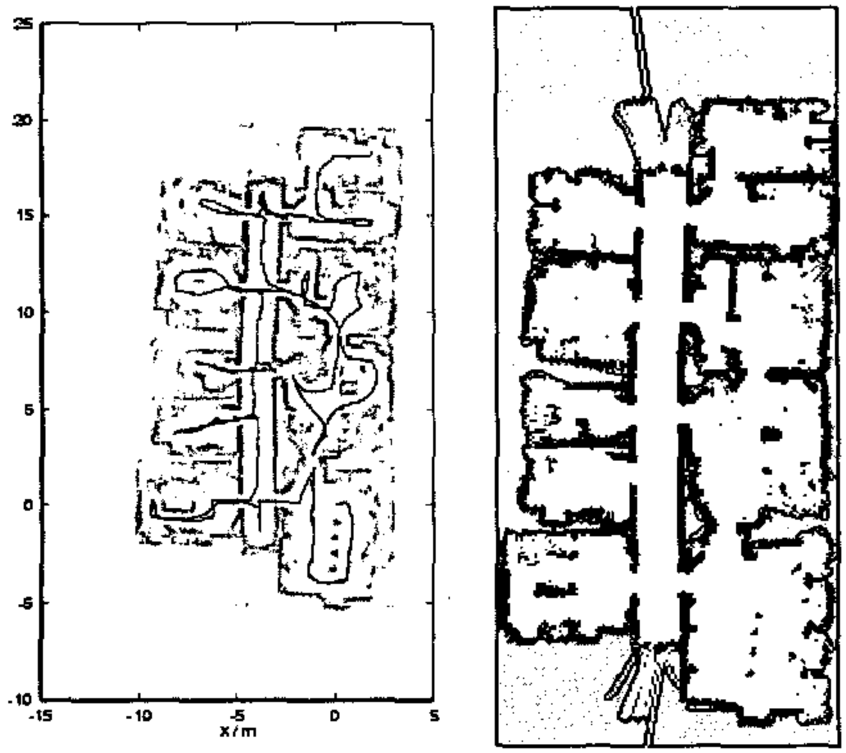


Figure 24 Corrected sensor data and the corresponding grid map [48].

Offspring were allocated to every individual according to the integer part of e_l, and the strings needed for the rest of the population were obtained by randomly generating new offspring for each string with the probability of the fractional part of e_l. Pairs of selected strings were then recombined by multipoint crossover, so that the encoded values in the two mating strings were completely mixed up in the resulting offspring. Mutation replaced very low probability single values within the strings with randomly generated values. Figure 24 shows the corrected sensor data obtained from the fitness solution (left) and the corresponding grid map (right).

5.4 MULTI-OBJECTIVE GENETIC ALGORITHM

MOO is an area of multiple-criteria decision making concerned with mathematical optimisation problems involving more than one objective to be optimised simultaneously [2]. It might be difficult to optimise a single objective by evolution for certain problems because it may present many local optima or suffer from the bootstrap problem.

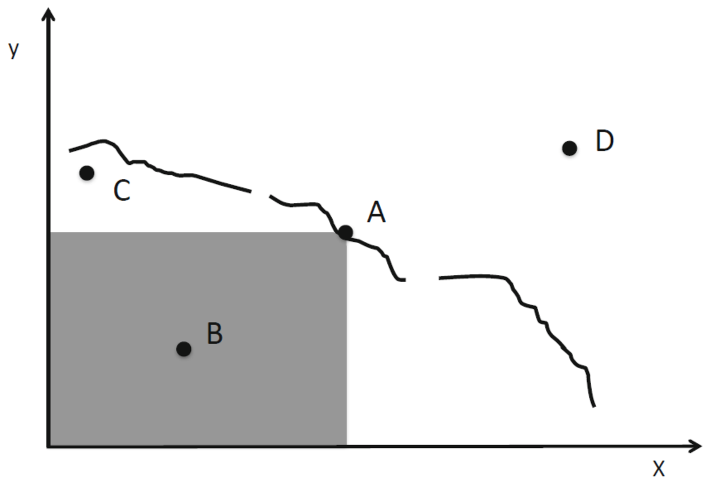


Figure 25 Illustration of Pareto front [52].

Hence, it may be preferable to use a multi-objective formulation and find the most favourable solution a-posteriori by a trade-off of the objectives from a pareto set. In [52], Pareto dominance is defined as:

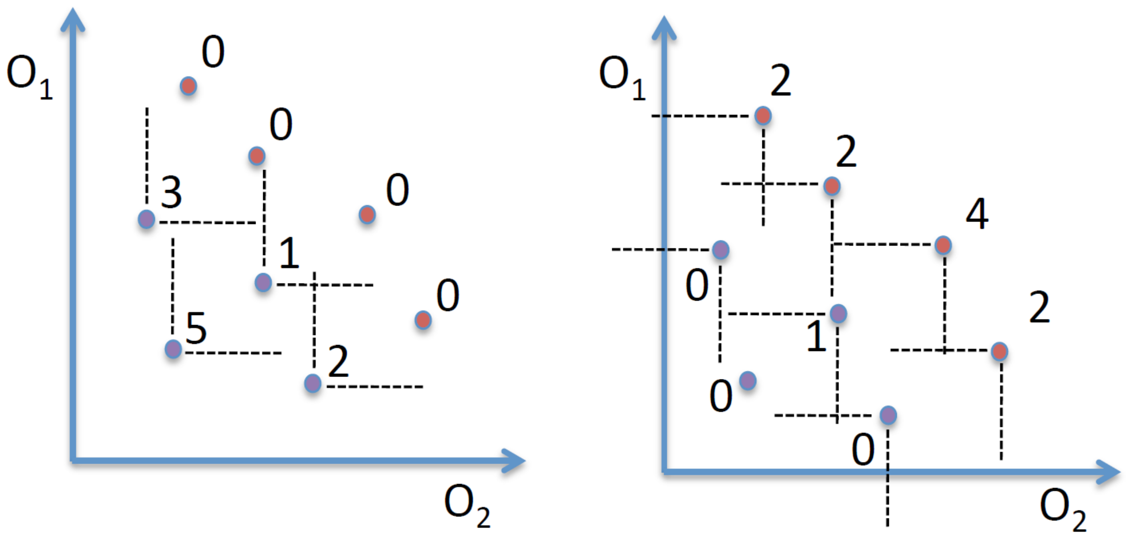
A solution x^* is said to dominate another solution x, if both conditions 1 and 2 are true:

1. the solution x^* is not worse than x with respect to all objectives.

2. *the solution x^* is strictly better than x with respect to at least one objective.*

In Figure 25, the objectives optimised in the problem are represented by the x- and y-axes, the larger the x and y values the better the solution for that objective. Here, point A dominates B and all other points in the grey area, whereas A and C do not dominate each other. The line represents the Pareto set, the set of nondominated solutions, like point A. As explored in [2] and [47], using Multi Objective Optimisation:

i. promotes the evolution of a more varied set of behaviours by exploring multiple trade-offs of the objectives to optimise,
ii. avoids premature convergence to local optima possibly introduced by multi-component fitness functions, and
iii. solves the bootstrap problem by exploiting objectives to guide evolution in the early phases.



(a) Dominance ranking (b) Dominance count

Figure 26 Examples of Pareto based dominance measures [53].

In this research three different methods of implementing a MOGA have been explored, two Pareto-based dominance measures, namely, dominance rank and dominance count, and one by introducing a periodic switch in the objectives.

- Dominance rank (DR) is the count of the number of other solutions that dominate each candidate solution. Figure 26 (a) shows how the lower ranked solutions occupy the pareto front. This mechanism tends to result in a broad front of solutions being maintained.

- Dominance count (DC) is the count of the number of solutions that each candidate solution dominates. Figure 26 (b) illustrates solutions with higher count values occupying the pareto front. This method tends to result in a concentration of solutions about some focal point.
- Switching fitness (SF) where a switch is implemented to shift between the objectives. Therefore, rather than attempting to optimise multiple objectives simultaneously, there is only one objective optimised at any given time [54].

5.5 DEFINING THE OBJECTIVES

To find which set of parameters produces the most accurate map of an environment the quality of the map needs to be examined. [55] lays out three metrics to evaluate a SLAM algorithm using its map, namely

- The ratio between the number of occupied cells and the number of free cells,

$$\eta = \frac{number\ of\ occupied\ cells\ in\ a\ map}{total\ number\ of\ cells\ in\ a\ map}\ . \qquad 5.4$$

  This measure helps determine whether features, such as walls on a map, are blurred, like the one shown in Figure 27, or appear more than once due to a failed execution of the algorithm. Lesser value of η indicates that the map is of better the quality than a map with higher η.

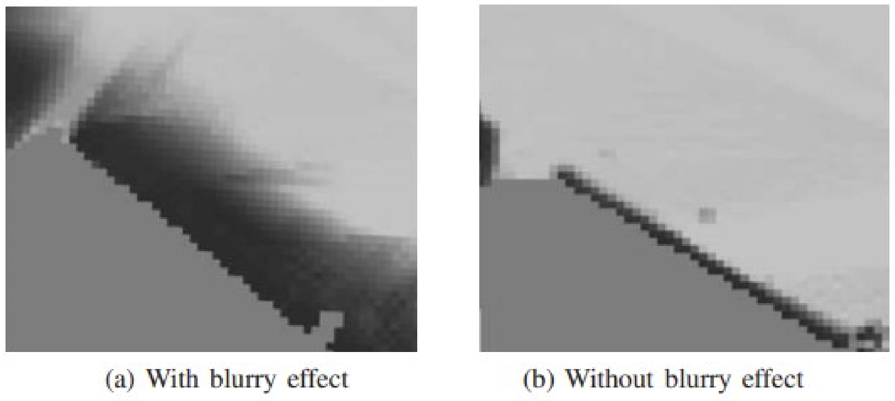


Figure 27 Map with (a) blurred wall and (b) accurate wall [55].

- The number of corners in a map, n_c. A lower quality map would contain a greater number of corners than one of higher quality. For example, consider the maps in Figure 28, the map on the right side of the figure has 39 corners when compared to the 107 corners in the map on the left.
- The number of enclosed areas, n_e, represents a similar idea as the number of corners. If a map is skewed or there was a failed attempt at loop closure, maps can have overlapping features. [56] suggests following borders to detect enclosed spaces, hence finding the number of contours in the map could be a useful technique to find n_e. When comparing the maps in Figure 28, the map on the right side has 27 contours compared to the 55 contours of the one on the left.

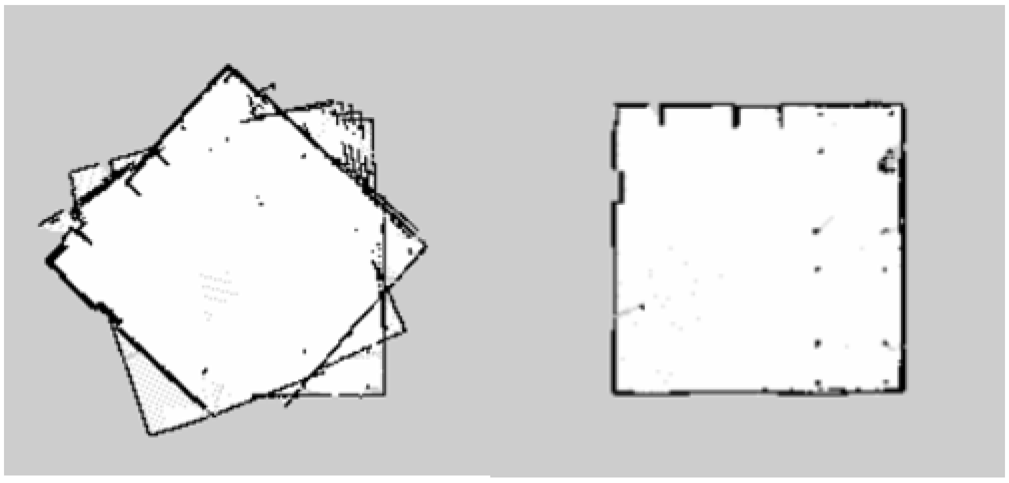


Figure 28 Comparing maps based on corners and enclosed spaces.

Therefore, it can be concluded that minimizing η, n_c and n_e can be considered conducive to producing more accurate maps.

CHAPTER 6   IMPLEMENTATION AND RESULTS

This chapter discusses the steps followed in conducting the different sets of experiments, the implementation of different MOGA techniques in each set of experiments and the results of these experiments. The results from the set of simulation experiments are then used to conduct experiments on the QBot2 robot. The experimental setup and results from the experiments on the QBot2 are discussed as well.

6.1 AUTONOMOUS EXPLORATION AND MAPPING

For the scope of this research the robot is tested in a custom environment named 'RoboticsLab', which is modelled after the Robotics lab located at C160 in the C-Building at Sexton campus, as shown in Figure 29. It is a 6m × 6m room with brick walls that are 15 cm thick and 2.5m high and natural sunlight as the light source. There are three long wooden tables, a café table, two wooden bookshelves, a wooden case, a cardboard box and an extra TurtleBot2 placed in the room. The simulated world is created using Gazebo 3D simulator

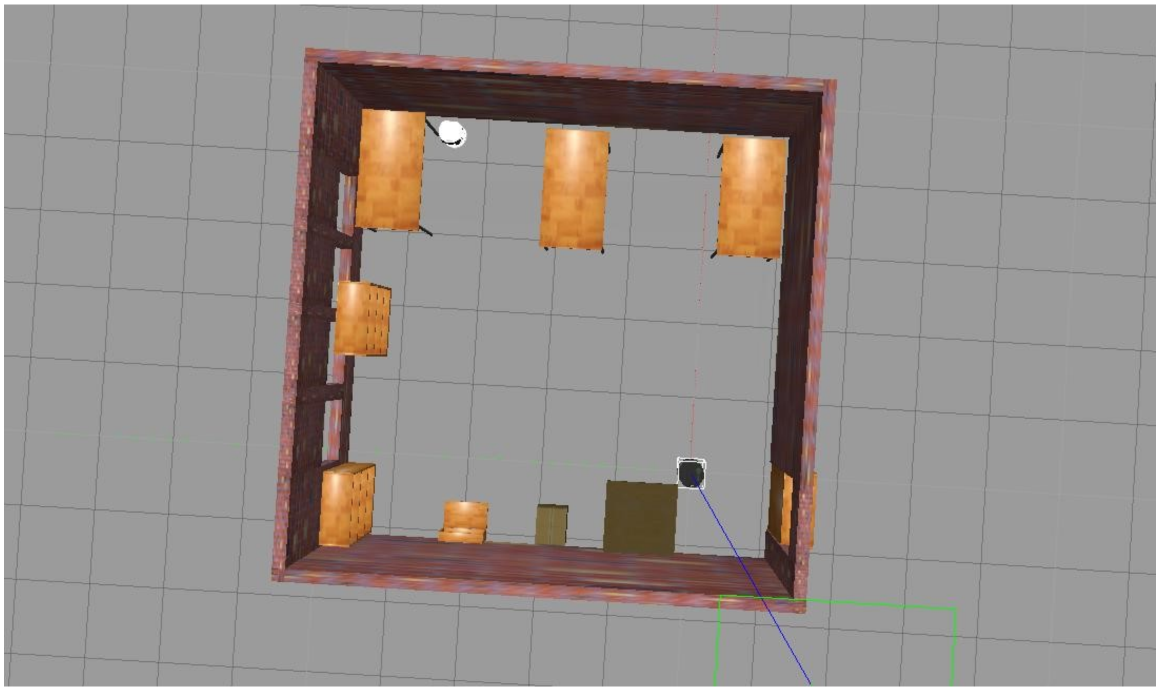


Figure 29   'RoboticsLab' environment simulated in Gazebo.

The TurtleBot2 is initially tested in the ‘RoboticsLab’ world using the RTAB-Map package in ROS with the parameters set to their default values, i.e., Mem/RehearsalSimilarity = 0.3, Kp/MaxDepth = 4.0 and Vis/MinInliers = 15. The *explore_lite* package is used for autonomous navigation of the TurtleBot2 and the RViz package to observe the robot movement and mapping. The commands executed to start the exploration and navigation process are listed in Table 10. The TurtleBot2 starts from the origin, depicted by the three axes in red, blue, and green, and explores the world from there. RViz produces 3D maps like the one shown on the left side of Figure 30 where the robot’s field of vision is shown by the coloured highlights in the map. Maps like the one shown on the right side of Figure 30 could also be exported using rtabmap-databaseViewer application GUI; here the blue lines depict the path followed by the robot from the origin to its destination and the red lines indicate loop closures.

Table 10 ROS Commands used in simulation experiments.

```
$ roslaunch turtlebot_gazebo turtlebot_world.launch
  world_file:=/”file_path”/RoboticsLab.world
$ roslaunch rtabmap_ros demo_turtlebot_mapping.launch simulation:=true
$ roslaunch rtabmap_ros demo_turtlebot_rviz.launch
$ roslaunch explore_lite explore.launch
```

For our experiments 2D maps are required for comparing the corners, contours and occupied grids of these maps and these experiments will be run recursively for multiple generations making the use of GUI impractical. Therefore, the rtabmap-database is accessed through command-line interface using the commands in Table 11 to generate maps, like the one in Figure 28, as .pgm files.

Table 11 Commands for map creation using RTAB-Map database.

```
$ rosrun rtabmap_ros rtabmap _database_path:=~/.ros/rtabmap.db
$ rosrun map_server map_saver map:=grid_map
$ rosservice call /publish_map 1 1 0
```

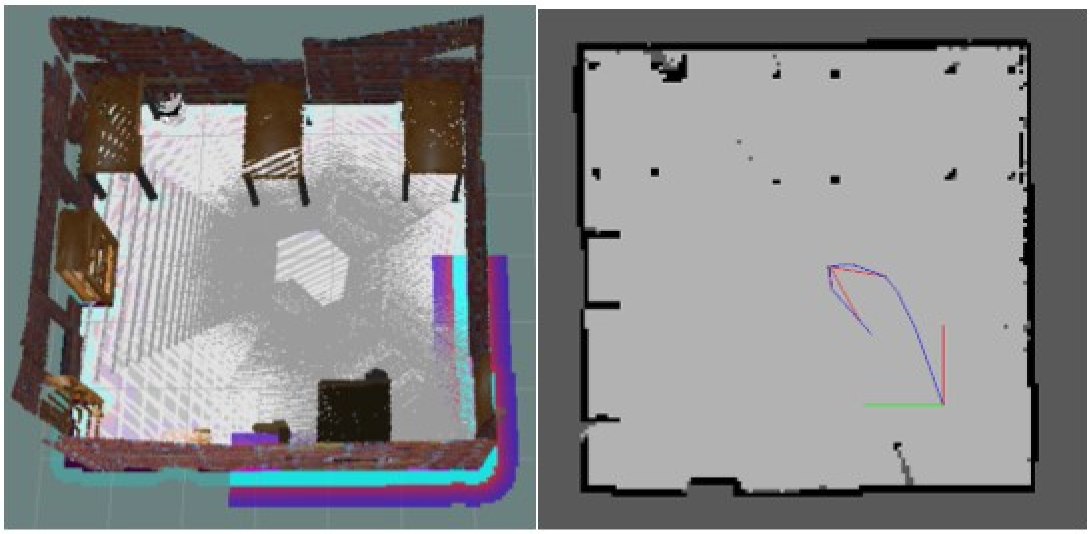


Figure 30 3D and 2D Maps of 'RoboticsLab' environment.

6.2 MOGA FOR RTAB-MAP PARAMETER OPTIMISATION

For RTAB-Map parameter optimisation, the ranges for the selected parameters are defined as follows:

- Mem/RehearsalSimilarity is a ratio so its values lie between 0 and 1.
- Kp/MaxDepth is the distance/depth at which the keypoints extracted are filtered and the range is set between 0 and 10m.
- Vis/MinInliers' value is a trade-off between the number of localizations and accuracy and the range is set between 10 and 20.

Figure 31 illustrates the flow of events during the implementation of the MOGA.

6.2.1 Selection and Replacement

The population size |P| for the MOGA is set as 50. An initial population of |P| members, is randomly generated, where each member is a set of parameter values within the defined range. All members of population P(t) generate one offspring. Both parents and offspring are ranked using the Pareto dominance techniques or the fitness switch introduced in the previous chapter, and the top |P| individuals from this pool are retained as the basis for population P(t + 1) and the rest of the individuals are discarded.

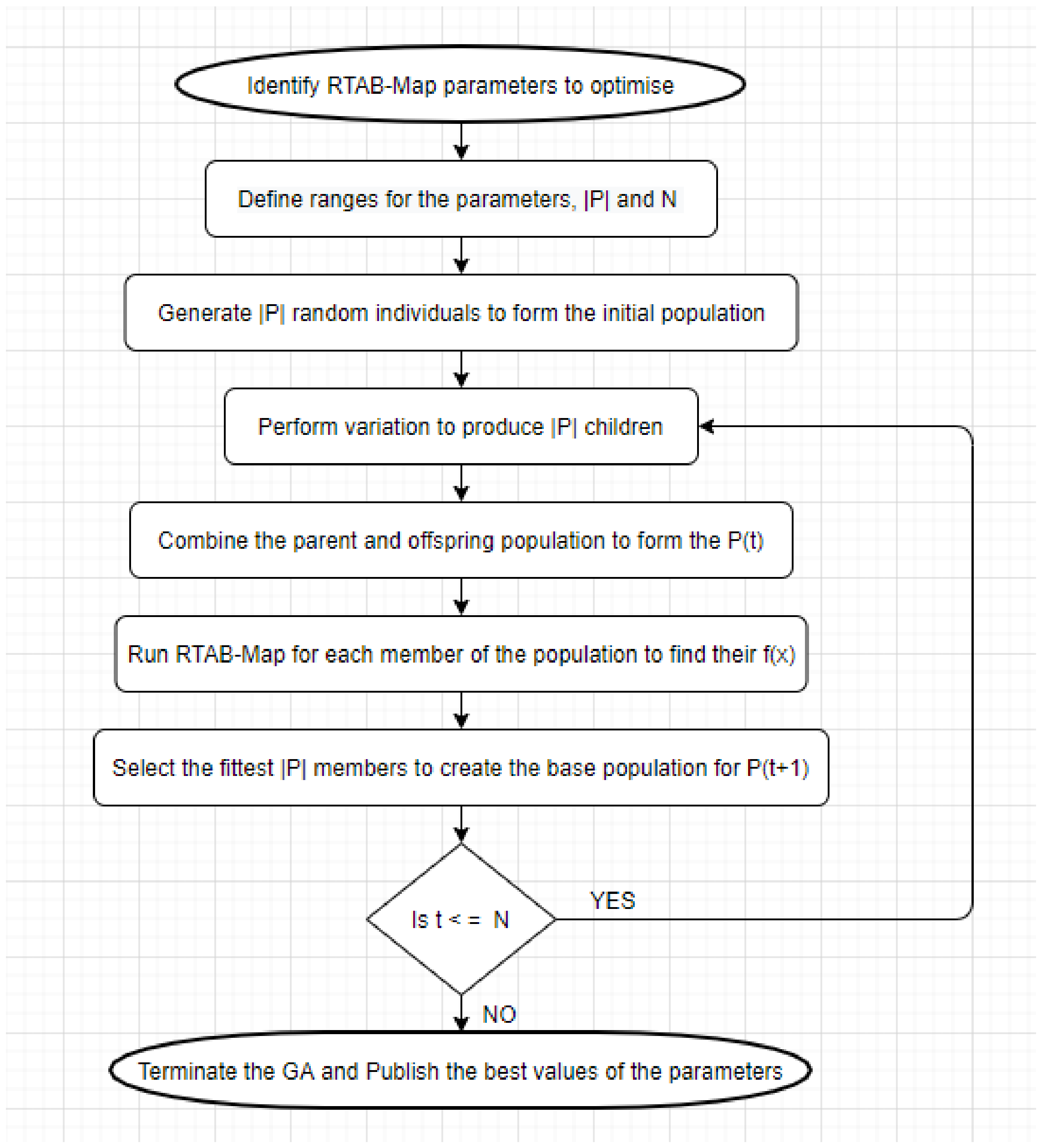

Identify RTAB-Map parameters to optimise
Define ranges for the parameters, |P| and N
Generate |P| random individuals to form the initial population
Perform variation to produce |P| children
Combine the parent and offspring population to form the P(t)
Run RTAB-Map for each member of the population to find their f(x)
Select the fittest |P| members to create the base population for P(t+1)
Is t < = N
YES
NO
Terminate the GA and Publish the best values of the parameters

Figure 31 Flowchart of Multi-Objective Genetic Algorithm.

6.2.2 Evaluation

To evaluate every individual of the population, *rtabmap_ros* commands are executed for each set of parameters and their maps are generated. These maps are converted from the '.pgm' format to greyscale images. To calculate the number of corners, n_c, these greyscale images are converted to float32 type and the OpenCV Harris corner detection function, cv2.cornerHarris(), is applied. To calculate the number of contours, n_e, the greyscale images are converted to binary images and the OpenCV function, cv2.findContours, is applied. The rtabmap_ros logs provide the number of occupied, empty and total grids in every local map that is added to the global map cache. To calculate the proportion of occupied grids of each map, η for every local map is calculated and their average is taken as the η of the final global map. The fitness function, also the quality of the map, for this GA can be defined as:

$$f(x) = min\{n_c(x), n_e(x), \eta(x)\}\,. \qquad 6.1$$

6.2.3 Variation

To implement this MOGA, uniform mutation is used as the variation operator, i.e., for every member of the parent population one of the three parameters will be changed at random to produce an offspring. This helps in maintaining diversity in the population. To prevent a solution associated with one peak in the search space being replaced by the offspring associated with a completely different peak and potentially resulting in the loss of an entire mode of the search space, crossover is not used and the variation operator is limited to mutation alone [53]. The number of generations, N, is used as the stop criterion for our MOGA setup.

6.3 SIMULATION EXPERIMENTS SETUP

Various python and shell scripts were created to control the automatic execution of the commands used for testing the MOGA in simulation as described below:

- Shell script to launch Gazebo, RTAB-Map and *explore_lite* using the ROS commands listed in Table 6.
- Shell script to create maps from the RTAB-Map database using the commands from Table 7.
- Python script to calculate n_c and n_e using the maps generated above.
- Python script to calculate η using data from RTAB-Map text logs.
- Python script for the appropriate MOGA mechanism.

These tests were executed for a total of 1,410 generations as detailed in Table 12 with a population pool of 100 members, 50 parents and 50 children, per generation that sums up to a total of 141,000 tests overall.

Table 12 Number of MOGA Generations for each set of experiments.

Experiment Setup	**Number of Generations**		
MOGA methods →	**Dominance Count**	**Dominance Rank**	**Switching Fitness**
Experiment Setup 1	60	60	60
Experiment Setup 2	60	60	60
Experiment Setup 3	350	350	350
Total	470	470	470
Grand Total	1410		

In this research the three MOGA methods, DC, DR and SF, are tested with RTAB-Map using the different techniques listed below:

- For the first experimental setup, the path taken by the robot to explore its world, i.e., the odometry, is kept constant for the entire test setup. The primary requirement for evolution is that the population exhibit variation in traits. This setup enabled evaluating the changes to the map quality when changing the parameter values for every member of the population. Therefore, setting the platform for this research.
- For the second experimental setup, we consider there are no motion errors and measurement error in the setup and the robot takes the least cost path to explore its environment. Therefore, the robot follows the same path to explore its environment for a particular member of the population even on multiple reruns. We observe which members of the population are promoted for their map quality.
- For the third experimental setup, the maps are evaluated considering the fact that due to motion and measurement errors, every time the robot explores the world it will take a different route, depending on the number of frontiers detected, even if

it is supplied the same set of parameters. Thus, producing a truly complex multi-objective problem to solve.

6.3.1 Experiment Setup 1

In this setup the robot odometry is kept constant throughout the evolution process. To achieve this,

- A default database containing the robot odometry and sensor information is generated using RTAB-Map with the default values of the parameters by executing commands from Table 6.
- 50 random members are generated as the initial population.
- 50 children are generated for the above members by applying uniform mutation.
- With the parents and children combined to form the population pool, *rtabmap-reprocess* application is implemented for every member using the following command, where the default database is supplied as the input to ensure constant odometry:

  ```
  $ rtabmap-reprocess --Mem/RehearsalSimilarity xxx --Kp/MaxDepth xxx --Vis/MinInliers xxx Input.db Output.db
  ```
- Using the databases generated in the previous step, maps are created by executing the commands from Table 7.
- As the robot does not explore the world for every member of the population, the *rtabmap_ros* logs are unavailable. Therefore, in these sets of experiments only n_c and n_e are considered as the objectives to be optimised. For the maps generated in the previous step, the values of n_c and n_e are calculated.

Appropriate MOGA method is then applied to assess the quality of these maps.

- When using DC, the n_c and n_e values of these maps are compared to find out the fitness of each member i.e., the number of members each member dominates. Based on these values, the members best suited to be the parents for next generation are selected. In case of DC higher the count, better the quality of member.

- When using DR, the values of the objectives are compared and members that are not dominated by any other member, i.e., those with least DR are selected to become parents for the next generation.
- With SF, the objective to be optimised is switched every five generations. For example, when considering n_c as the objective to be optimised, maps with least number of corners (n_c) will be considered the fittest, and the corresponding members will be selected to form the parent population of next generation. After five generations the process will be continued with n_e as the objective for the next five generations and so on. Hence in the SF approach, only one objective is optimised at any given time

To be true to the random nature of GA, a different set of initial population and a different default database is assigned to each of DC, DR and SF. The maximum number of generations, N, is set to 60 for each type of MOGA.

6.3.2 Experiment Setup 2

For these sets of experiments,

- 50 random members are generated to create the initial population for GA.
- Considering them as parents, 50 children are generated by applying uniform mutation
- With these 100 members as the population pool for the first generation *rtabmap_ros* SLAM is executed using the commands from Table 6.
- A map is created for each member of the population pool using commands from Table 7, i.e., 100 maps in total.
- The values of n_c, n_e and η are calculated for each of these maps.
- Applying the appropriate MOGA technique the 50 fittest members are identified.

In these experiments, to maintain the assumption that one member can only generate one type of map due to the absence of motion and measurement errors, from the next generation onwards *rtabmap_ros* SLAM is executed only for the children. This data is then combined with the data of the parent population for GA evaluation. This process is implemented for the DC, DR and SF techniques, each starting with a different initial population. For the SF technique, the fitness is switched between n_c, n_e and η every five generations. The cap is set at 60 generations for these sets of experiments as well.

6.3.3 Experiment Setup 3

Every time a robot explores an environment, it is bound to take a different path because of motion and measurement errors leading to changes in the frontier detection process. This setup is more practical than the two previous sets of experiments as it considers the fact that *rtabmap_ros* SLAM might produce a different map even if it is supplied the same set of parameters.

- 50 random members are generated as the initial population.
- 50 children are generated for the above members by applying uniform mutation.
- These 100 individuals form the population pool for the first generation of GA and *rtabmap_ros* SLAM is executed for every member using the set of commands from Table 6
- A map is created for each member of the population pool using commands from Table 7, i.e., 100 maps in total.
- n_c, n_e and η are calculated for each member of the population pool.
- Applying the appropriate MOGA technique, the 50 fittest members are selected to form the parent population for the next generation.

These steps are then repeated till the maximum number of generations, i.e., 350 is reached. It should be noted that unlike setup 2, in this setup the rtabmap_ros commands are executed for the entire population pool of 100 in every generation. This process is implemented for DC, DR and SF. For the SF technique, the fitness is switched between n_c, n_e and η every five generations.

6.4 RESULTS FROM SIMULATION EXPERIMENTS

The following sections discuss the results and observations from the simulation experiments conducted. The results for each experiment are presented in the below format.

- A table listing the best and worst values of the RTAB-MAP parameters along with the values calculated for the 3 objectives.
- A graph representing the evolution of the fittest candidate in terms of the three objectives namely corners, contours and Eta.

Each sub-section below contains the results of experiments conducted for the Dominance Count, Dominance Rank and Switching Fitness MOGA mechanisms.

6.4.1 Experiment Setup 1

Table 13 Experiment Setup 1- Results for Dominance Count.

RTABMAP Parameters	**Default**	**Best**	**Worst**
Mem/RehearsalSimilarity	0.3	0.77	0.57
Kp/MaxDepth	4	3.77	4.46
Vis/MinInliers	15	13	10
n_c	45	42	86
n_e	19	18	35

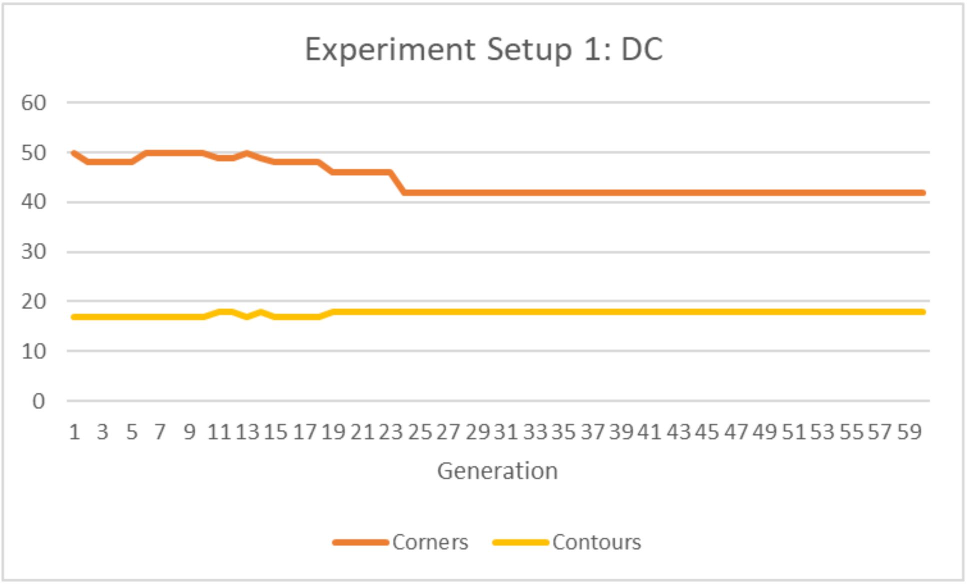


Figure 32 Experiment Setup 1- Evolution of fittest candidate- Dominance Count.

For the first set of experiments, each MOGA method was started with a different default *rtabmap.db* file, which means each database had different odometry and sensor data. Tables 13, 14 and 15 present the results for map evaluation of the default, best and worst members of the population for the DC, DR and SF MOGA methods, respectively.

Table 14 Experiment Setup 1- Results for Dominance Rank.

RTABMAP Parameters	**Default**	**Best**	**Worst**
Mem/RehearsalSimilarity	0.3	0.41	0.77
Kp/MaxDepth	4	8.11	4.23
Vis/MinInliers	15	14	14
n_c	49	42	49
n_e	18	21	18

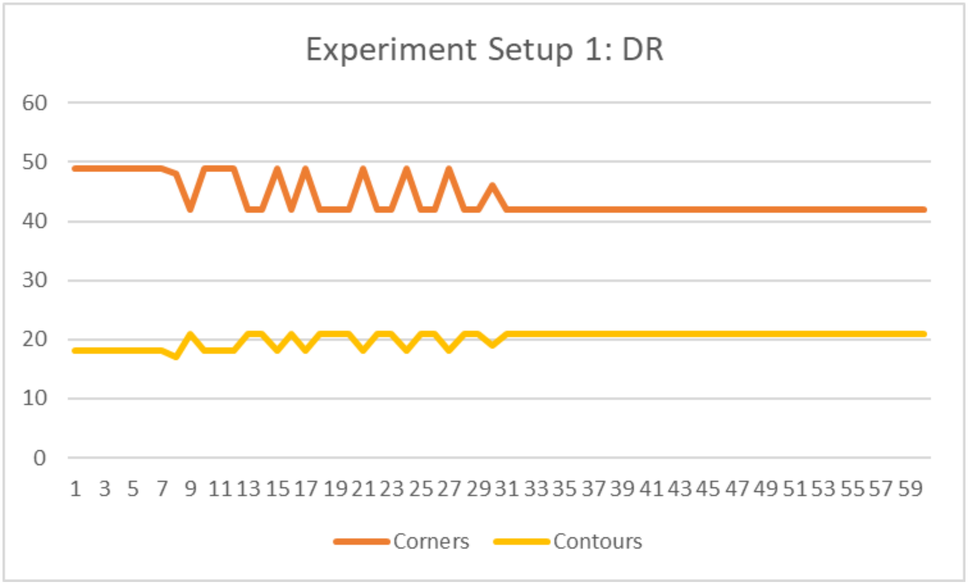


Figure 33 Experiment Setup 1- Evolution of fittest candidate- Dominance Rank.

Figures 32, 33 and 34 show the evolution of the fittest candidate for the DC, DR and SF techniques, respectively, over generations. It can be seen that all three techniques converge to produce a fittest candidate, albeit more than one. It is observed that over generations each technique produces multiple candidates with the same fitness value.

Where DC converges in generation 24, DR in generation 31, SF converges to produce its best result in the 41st generation. Because this setup is a two-objective problem, it is an easier problem to optimise than the other two setups. It must also be noted that the best and worst solutions in DR are both ranked 0, where they are a trade-off between n_c and n_e values.

Table 15 Experiment Setup 1- Results for Switching Fitness.

RTABMAP Parameters	Default	Best	Worst
Mem/RehearsalSimilarity	0.3	0.5	0.44
Kp/MaxDepth	4	2.52	2.96
Vis/MinInliers	15	10	20
$\boldsymbol{n_c}$	100	45	83
$\boldsymbol{n_e}$	50	20	55

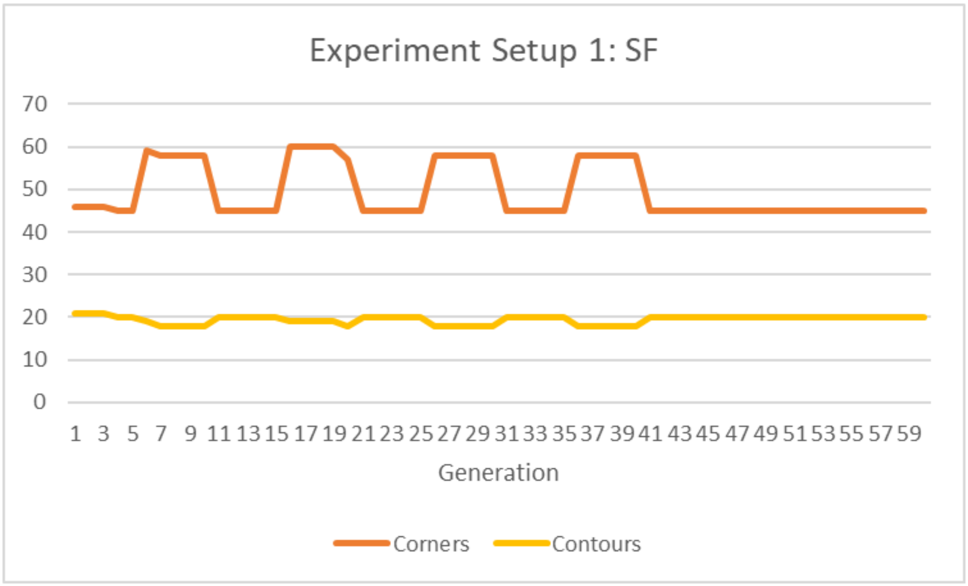


Figure 34 Experiment Setup 1- Evolution of fittest candidate- Switching Fitness.

Starting with a good map, the changes to parameter values in DR were not able to produce a variety of maps, even though mutation maintains diversity in the population.

6.4.2 Experiment Setup 2

Table 16 Experiment Setup 2- Results for Dominance Count.

RTABMAP Parameters	Best	Worst
Mem/RehearsalSimilarity	0.72	0.72
Kp/MaxDepth	1.98	4.17
Vis/MinInliers	18	13
n_c	39	132
n_e	19	48
η	0.24	0.27

In this set of experiments, we assumed that each candidate will produce the same map even on multiple explorations by the robot. Hence, we only created maps for the children and compared their qualities with the existing maps of their parents and promoted the fitter half of the population every generation. As the GA progresses the members producing lower quality maps are eliminated from the population pool. Additionally uniform mutation ensures diversity in the population.

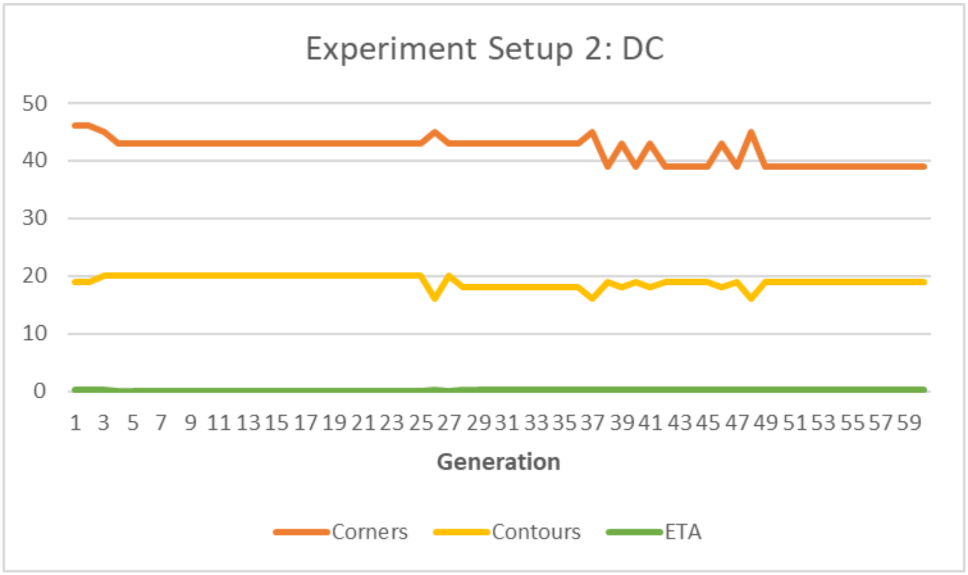


Figure 35 Experiment Setup 2- Evolution of fittest candidate- Dominance Count.

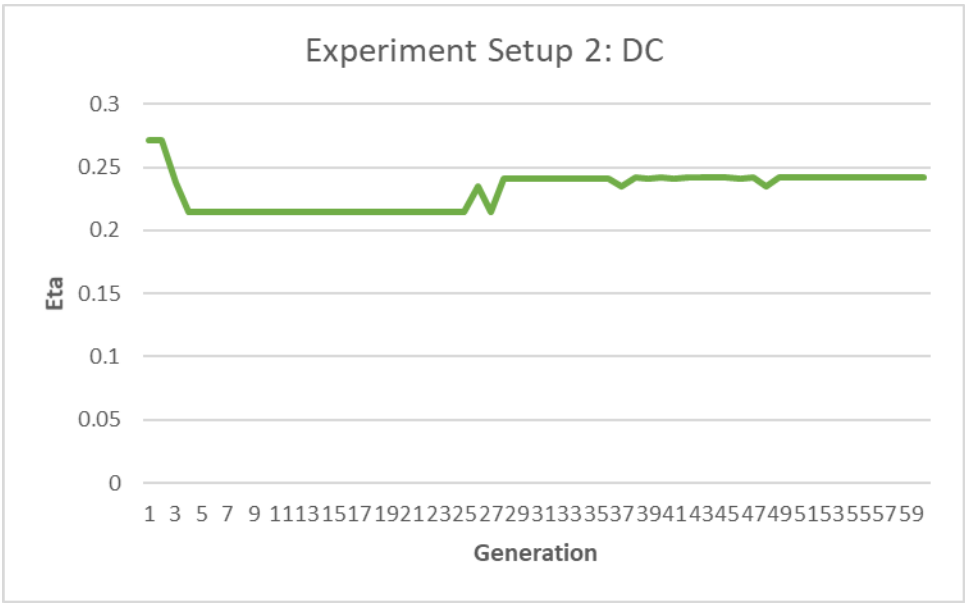


Figure 36 Experiment Setup 2- Evolution of fittest candidate (Eta)- DC.

Tables 16, 17 and 18 present the results for the best and worst candidates for the DC, DR and SF MOGA mechanisms, respectively. From Figures 35, 37 and 40 the evolution of the fittest candidates for the DC, DR and SF mechanisms, respectively, can be observed over generations. For DC, the quality of optimal candidate converges in generation 49 and for DR in generation 39. Because this problem is truly multi-objective in nature, there is no convergence when using the pseudo MOGA technique SF.

Table 17 Experiment Setup 2- Results for Dominance Rank.

RTABMAP Parameters	**Best**	**Worst**
Mem/RehearsalSimilarity	0.28	0.11
Kp/MaxDepth	9.08	7.32
Vis/MinInliers	14	13
$\boldsymbol{n_c}$	35	150
$\boldsymbol{n_e}$	18	126
$\boldsymbol{\eta}$	0.24	0.76

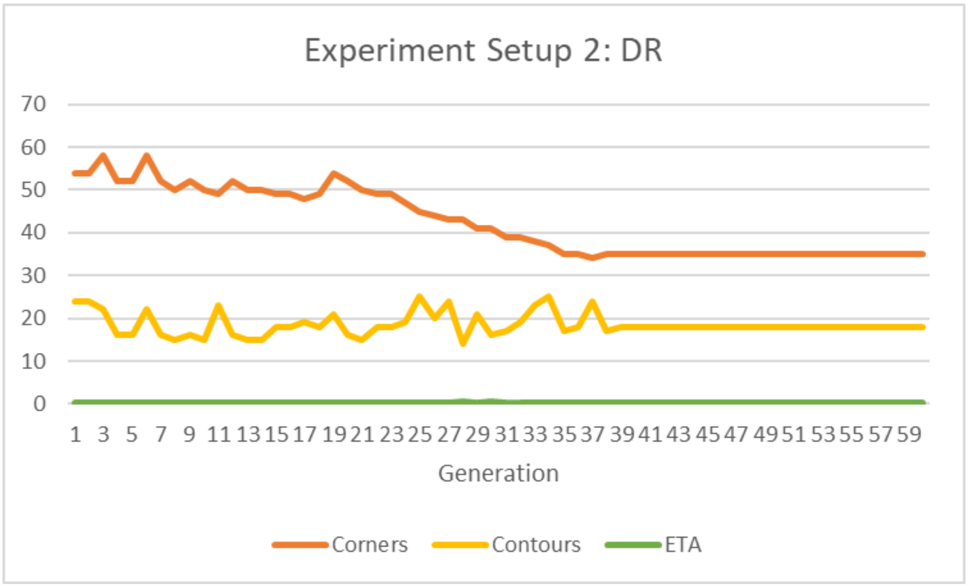


Figure 37 Experiment Setup 2- Evolution of fittest candidate- Dominance Rank.

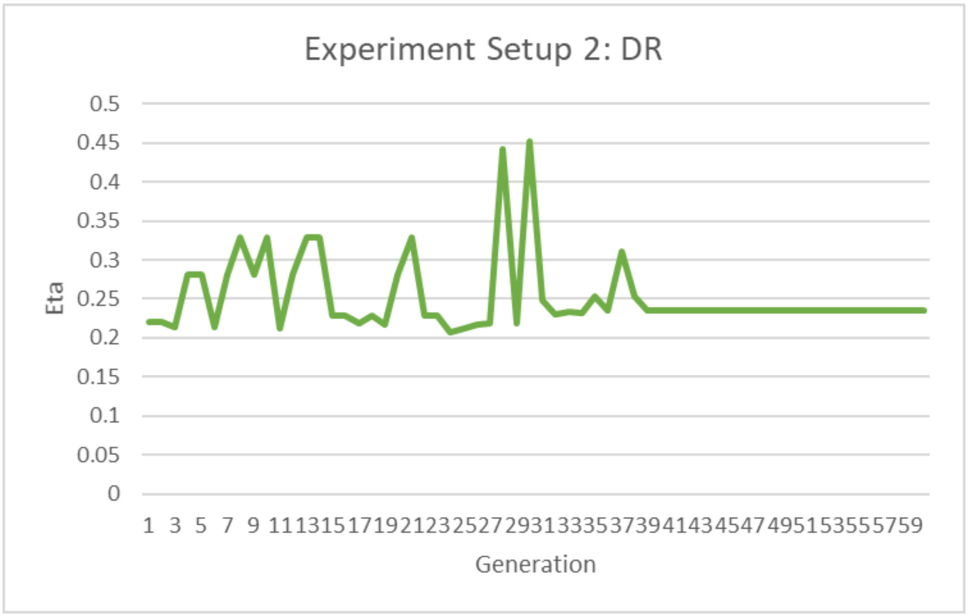


Figure 38 Experiment Setup 2- Evolution of fittest candidate (Eta)- DR.

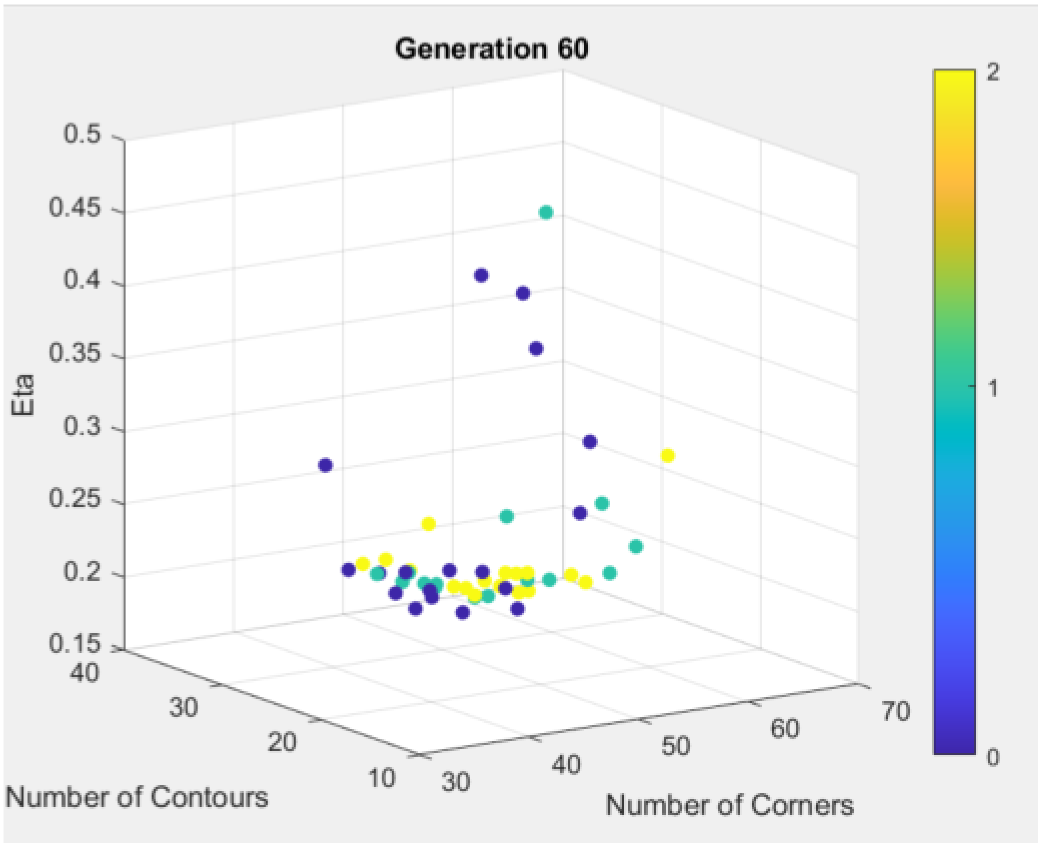


Figure 39 Experiment Setup 2- MOGA results for 60th Generation.

From Figure 40, it should be noted that the graph is stable for 5 generations, i.e., every five generations SF is able to successfully optimise one of the objectives to produce a fittest candidate.

Table 18 Experiment Setup 2- Results for Switching Fitness.

RTABMAP Parameters	**Best**	**Worst**
Mem/RehearsalSimilarity	0.17	0.02
Kp/MaxDepth	9.66	5.34
Vis/MinInliers	18	13
$\boldsymbol{n_c}$	49	93
$\boldsymbol{n_e}$	19	47
$\boldsymbol{\eta}$	0.21	2.57

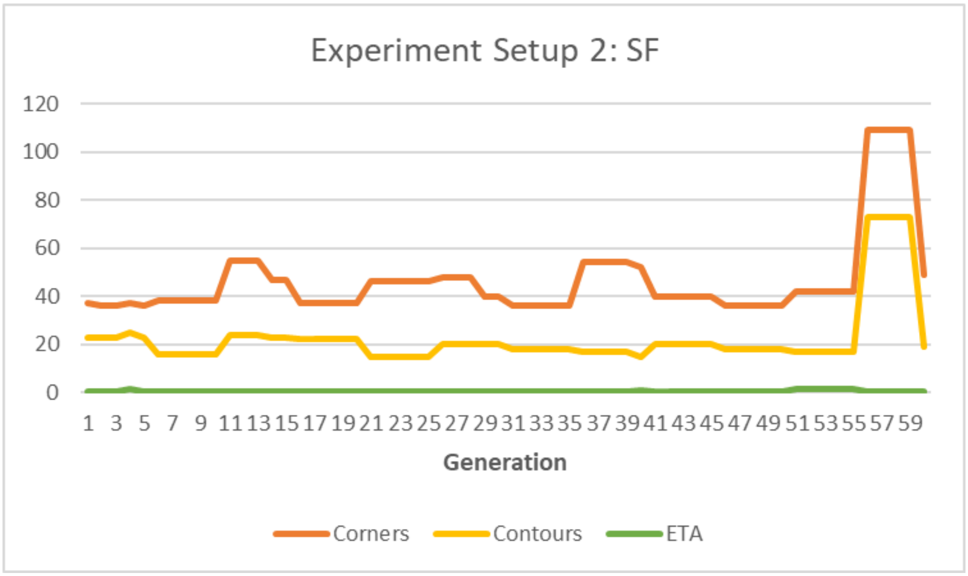


Figure 40 Experiment Setup 2- Evolution of fittest candidate- Switching Fitness.

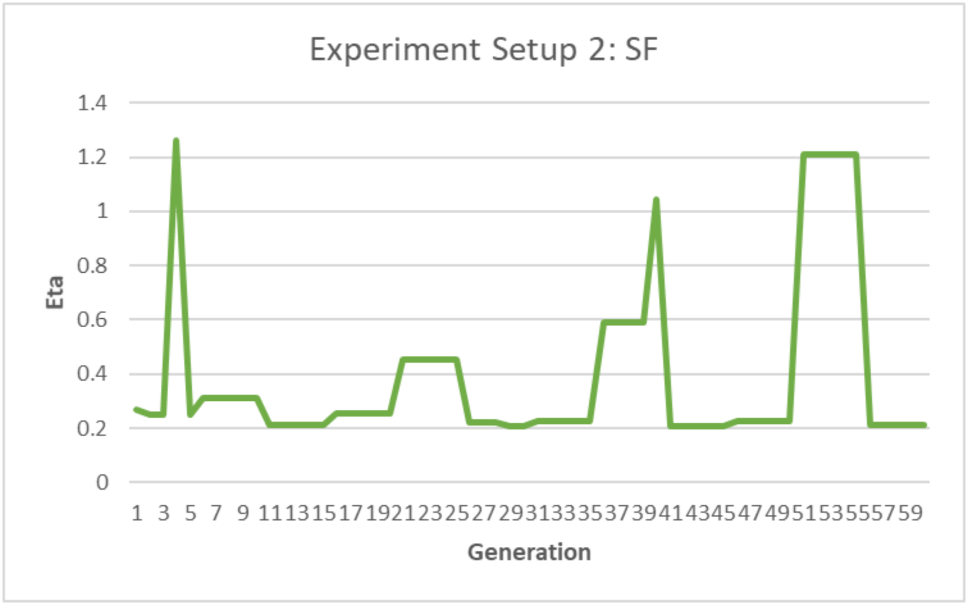


Figure 41 Experiment Setup 2- Evolution of fittest candidate (Eta)- SF.

When using the DC measure for Pareto dominance, individuals are rewarded to find a spot in the Pareto front where they increase the count of individuals they dominate. Therefore, it can be observed that DC results in a single optimal solution. The DR measure rewards individuals to find a spot where no other individual dominates them.

Therefore, DR offers multiple optimal solutions, depicted in purple in Figure 39, that offer a trade-off between the different objectives. For plotting the graph depicting the evolution of the fittest candidate, the first individual appearing in the Dominance Rank list of every generation is used. Figures 36, 38 and 41 offer a closer look at how η converges to its optimal value.

6.4.3 Experiment Setup 3

Table 19 Experiment Setup 3- Results for Dominance Count.

RTABMAP Parameters	**Best**	**Worst**
Mem/RehearsalSimilarity	0.55	0.67
Kp/MaxDepth	5.2	9.9
Vis/MinInliers	17	15
$\boldsymbol{n_c}$	39	146
$\boldsymbol{n_e}$	19	87
$\boldsymbol{\eta}$	0.27	1.03

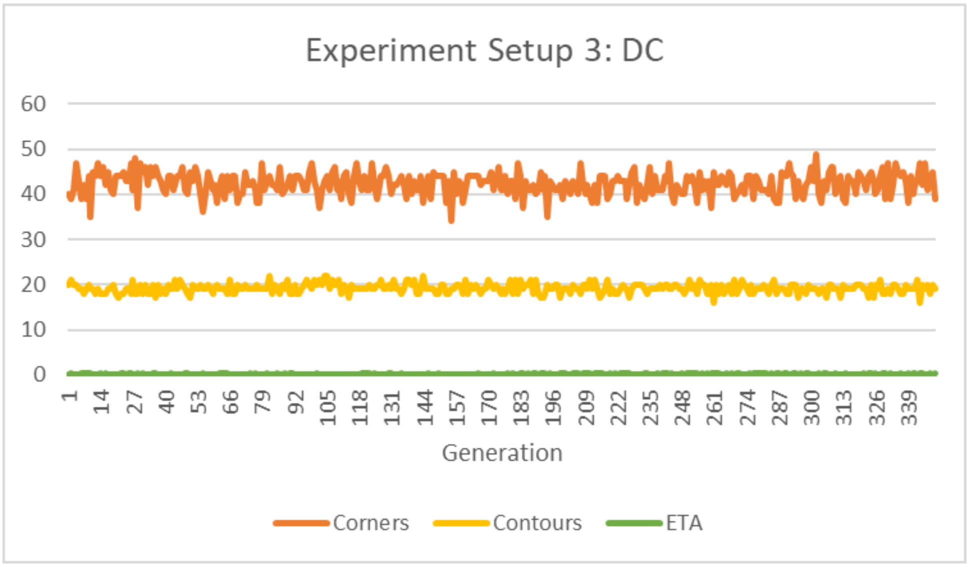


Figure 42 Experiment Setup 3- Evolution of fittest candidate- Dominance Count.

In this set of experiments, we observe that for the same set of parameters, a different map is generated because the robot's odometry changes. Therefore, compared to the other two setups, this setup puts forth a more complex three-objective problem to solve.

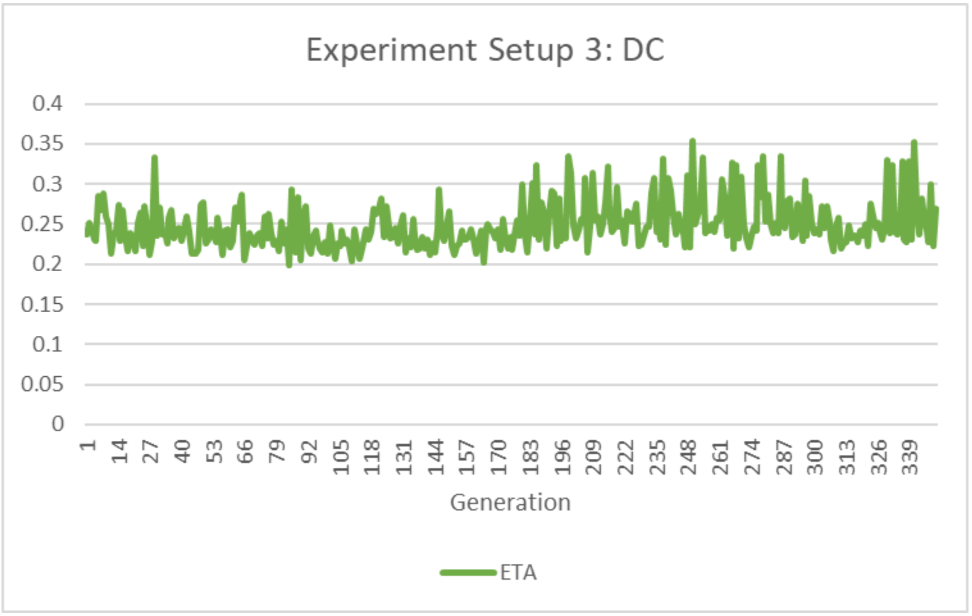


Figure 43 Experiment Setup- Evolution of fittest candidate (Eta)- DC.

Table 20 Experimental Setup 3- Results for Dominance Rank.

RTABMAP Parameters	**Best**	**Worst**
Mem/RehearsalSimilarity	0.4	0.81
Kp/MaxDepth	9.67	4.24
Vis/MinInliers	16	15
$\boldsymbol{n_c}$	38	110
$\boldsymbol{n_e}$	20	71
$\boldsymbol{\eta}$	0.25	0.36

Tables 19, 20 and 21 present the results for the best and worst candidates for the DC, DR and SF MOGA techniques respectively. In this setup, due to the continuously changing robot odometry, the lower quality candidates are not entirely eliminated from the population pool, even though their genetic information might not be selected to be passed on to the next generation.

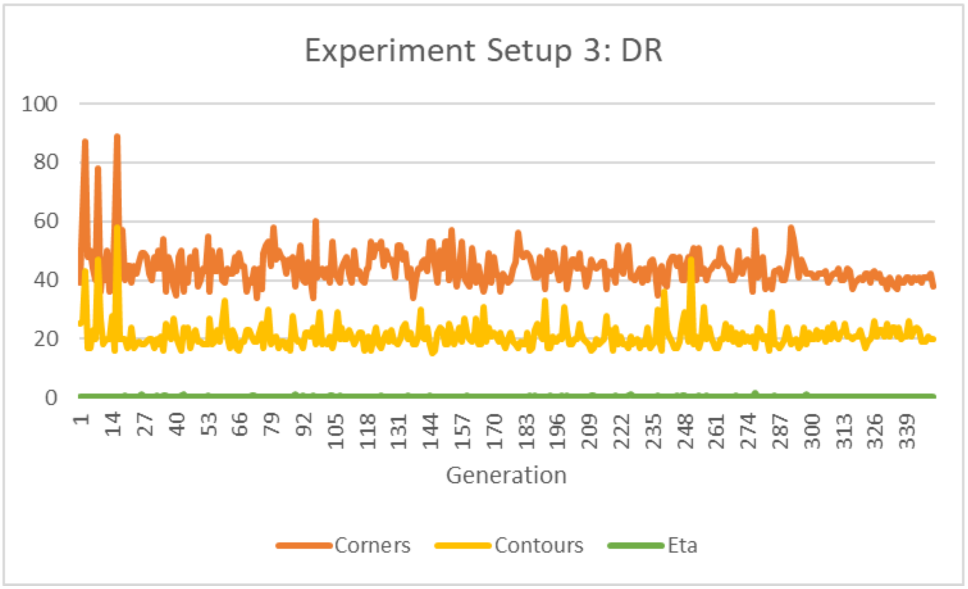


Figure 44 Experiment Setup 3- Evolution of fittest candidate- Dominance Rank.

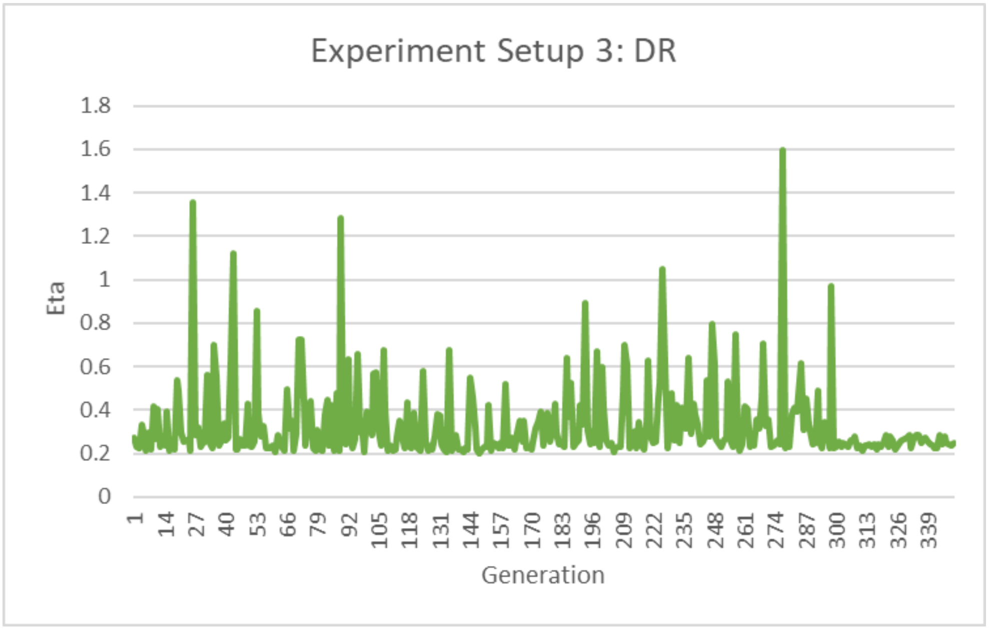


Figure 45 Experiment Setup 3- Evolution of fittest candidate (Eta)- DR.

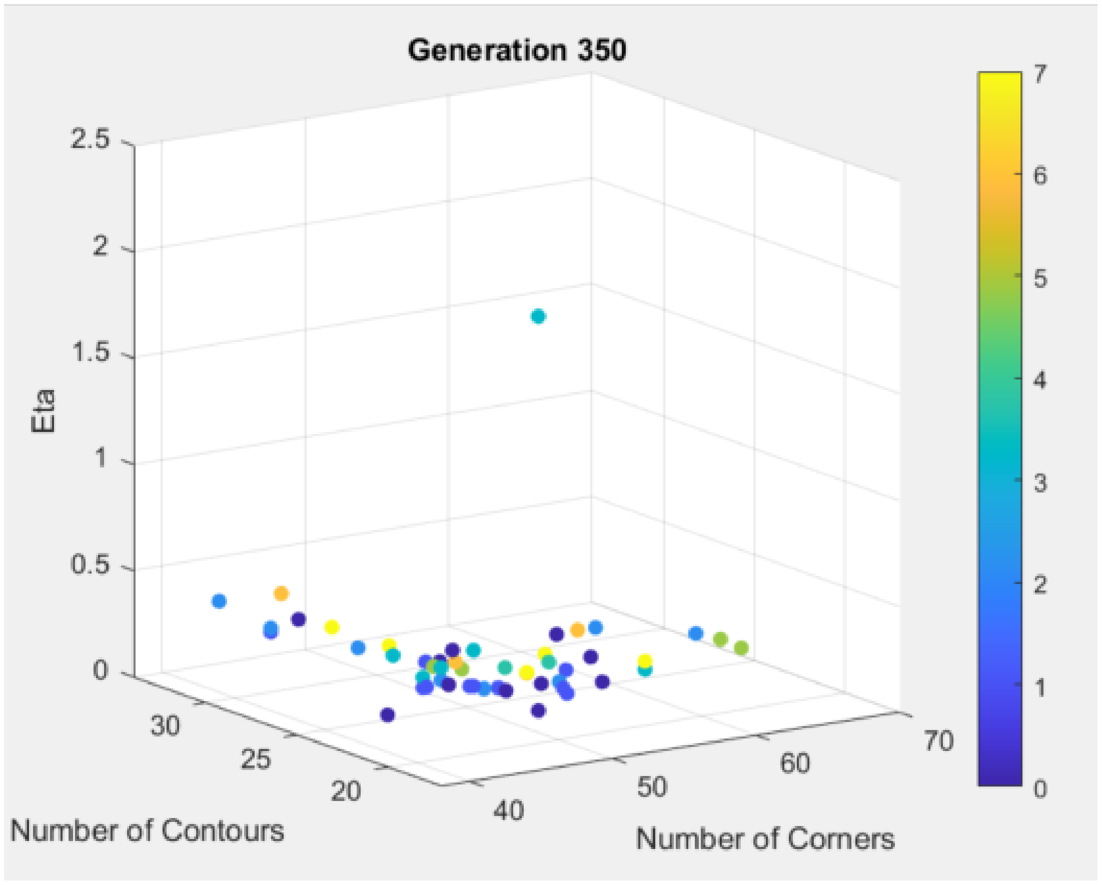


Figure 46 Experiment Setup 3- MOGA Results for 350th Generation.

Table 21 Experiment Setup 3- Results for Switching Fitness.

RTABMAP Parameters	Best	Worst
Mem/RehearsalSimilarity	0.42	0.12
Kp/MaxDepth	0.58	3.79
Vis/MinInliers	11	17
$\boldsymbol{n_c}$	52	190
$\boldsymbol{n_e}$	19	149
$\boldsymbol{\eta}$	0.22	2.09

Figures 42, 44 and 47 show the evolution of the fittest candidate for DC, DR and SF, respectively, over generations. Figures 43, 45 and 48 contain the graphs for η optimisation to provide a closer look at how it converges for the fittest members across generations. For the DC technique, the GA presents one optimal candidate whose quality does not change as drastically as the one from DR, however, because the robot odometry keeps varying on every run, convergence is elusive till generation 350.

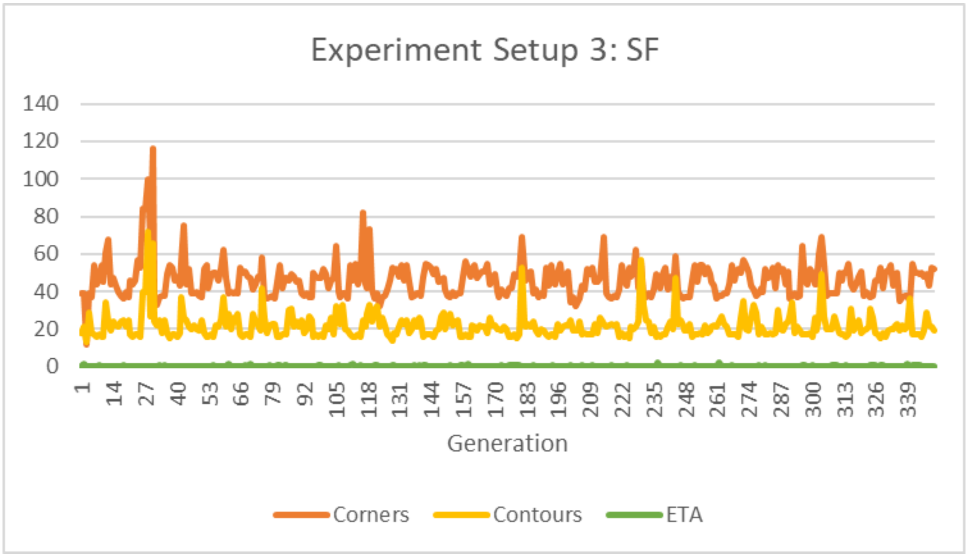


Figure 47 Experiment Setup 3- Evolution of fittest candidate- Switching Fitness.

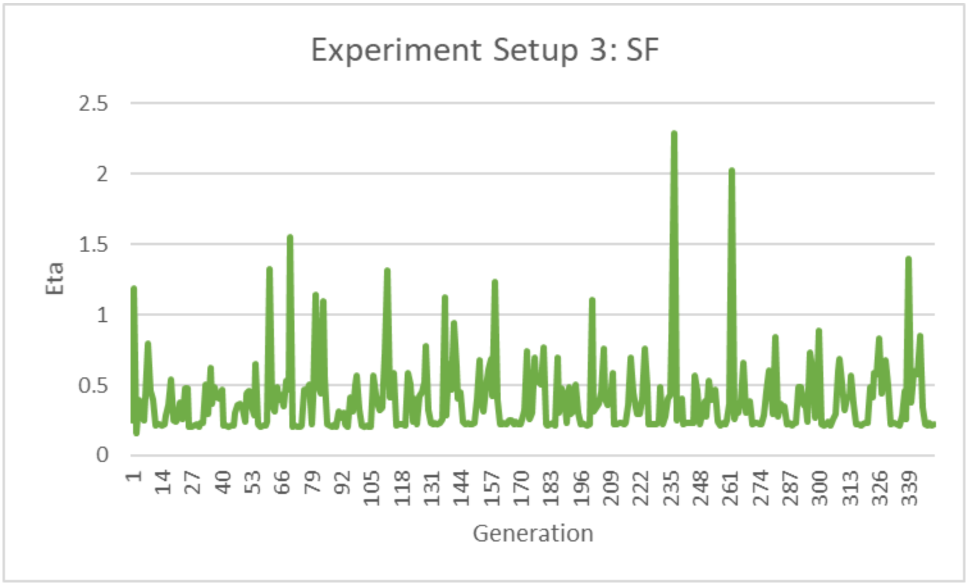


Figure 48 Experiment Setup 3- Evolution of fittest candidate (Eta)- SF.

For the DR technique, the graph stabilises fairly after the first few generations, which then converges further from generation 300. We observe that over generations, DR presents multiple optimal solutions which provide a trade-off between the three objectives. In Figure 46, the purple dots represent the pareto front for generation 350, i.e., solutions with rank 0. For the pseudo MOGA technique SF, this setup seems too complex to handle. When compared to Figure 40, the graph in Figure 47 has more instabilities even during the five generations each objective is optimised for. When comparing the best candidate results, it can be noticed that DC and DR provide more balanced solutions as opposed to the SF solution. This in turn emphasises the importance of using MOGA, rather than single objective GA, when optimising complex problems like SLAM.

6.5 QBOT2 EXPERIMENT SETUP

University closure due to COVID-19 has made the C160, Robotics lab at Sexton campus in Dalhousie University, inaccessible for testing the robot. Therefore, the QBot2 robot was tested in an environment setup that closely resembles the environment created for the simulation experiments, i.e., a 3.96m × 3.96m room with two wooden tables, a wooden bookshelf, a wooden chest of drawers, a metal pedestal, four dining chairs and cylindrical jar placed around for the robot to identify as shown in Figure 50. Because the QBot2 sits on a Yujin Kobuki base like the TurtleBot2 and has a Kinect camera, the *rtabmap* command for TurtleBot2 works with the QBot2 as well. Using the commands in Table 22 the QBot2 was able to autonomously explore its world and the commands in Table 7 were used to create maps from the databases generated.

Table 22 ROS Commands RTAB-Map SLAM for QBot2 experiments.

```
$ roslaunch turtlebot_bringup minimal.launch
$ export TURTLEBOT_3D_SENSOR=kinect
$ roslaunch rtabmap_ros demo_turtlebot_mapping.launch args:="--delete_db_on_start"
$ roslaunch rtabmap_ros demo_turtlebot_rviz.launch
$ roslaunch explore_lite explore.launch
```

The QBot2 robot was tested with the default values of the parameters and the best and worst candidates for the three MOGA techniques from experimental setup 2 and experimental setup 3. Figure 49 shows the how the Kinect camera detects its world.

Experiment setup 1 is used to establish that variation in the values of the chosen parameters will change the quality of map generated by the robot. Hence, the candidates from this setup are not used for QBot2 testing.

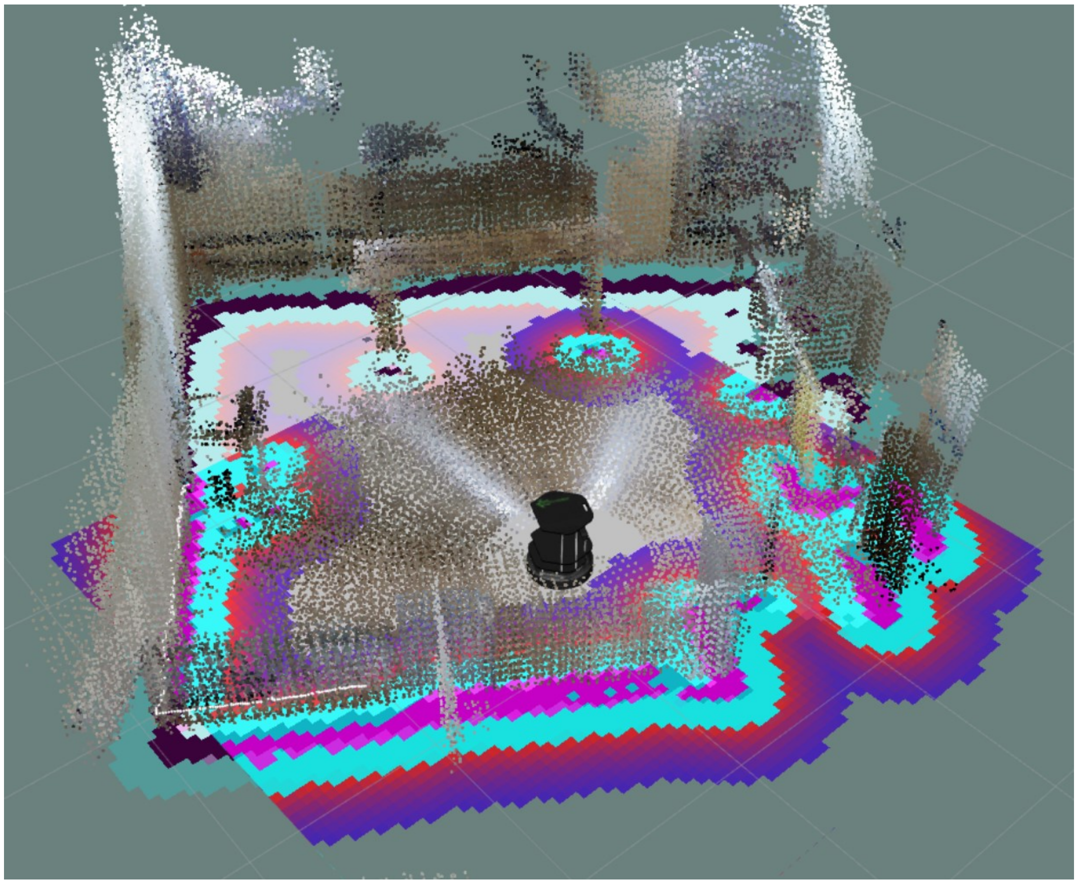
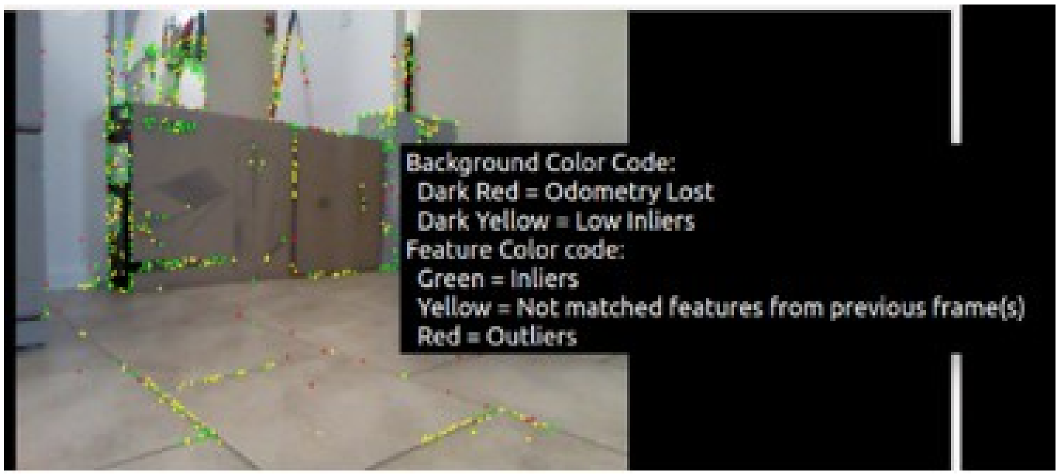


Figure 49 3D Map of the world from RVIZ.



Background Color Code:
Dark Red = Odometry Lost
Dark Yellow = Low Inliers
Feature Color code:
Green = Inliers
Yellow = Not matched features from previous frame(s)
Red = Outliers

Figure 50 Qbot2 Kinect Camera- Initial Field of View.

6.6 RESULTS FROM QBOT2 EXPERIMENTS

Table 23 presents the results from experiments conducted with the Qbot2 robot. The numbers in parentheses denote if the results are for the data from experimental setup 2 or experimental setup 3. In experiments conducted with the QBot2, the best candidates from experiment setup 2 are not able to produce results similar to the simulation experiments due to errors in motion and measurement. The best candidates for both DC and DR generated maps that are worse than the map generated by the default candidate. The best candidate for SF generated a better quality map than the default candidate's map, i.e., the number of corners and contours of the map generated by the best candidate for SF was lesser than the map generated by the default candidate.

Table 23 Results from QBot2 experiments.

RTABMAP Parameters →	**Mem/Rehearsal Similarity**	**Kp/MaxDepth**	**Vis/MinInliers**	n_c	n_e	η
Default	0.3	4.0	15	66	29	0.74
DC Best (2)	0.72	1.98	18	86	31	0.80
DC Worst (2)	0.72	4.17	13	71	25	1.08
DR Best (2)	0.28	9.08	14	78	30	0.81
DR Worst (2)	0.11	7.32	13	78	42	0.63
SF Best (2)	0.17	9.66	18	63	25	0.80
SF Worst (2)	0.02	5.34	13	92	36	1.29
DC Best (3)	0.55	5.22	17	66	30	0.65
DC Worst (3)	0.67	9.9	15	105	29	0.85
DR Best (3)	0.4	9.67	16	66	17	0.62
DR Worst (3)	0.81	4.24	15	77	26	0.54
SF Best (3)	0.42	0.58	11	61	29	0.62
SF Worst (3)	0.12	3.79	17	81	24	0.60

The best candidate from experiment setup 3 for the DC approach, generated a map that has the same number of corners and one more contour than the default candidate's map, however the map has a lower proportion of occupied grids than the default candidate's map. Hence, they offer two maps of similar quality with a trade-off between the number of contours and the proportion of occupied grids. For the DR approach, the best candidate generated a map with lesser number of contours and a lower proportion of occupied grids than the default candidate's map. Thus, making the best candidate's map no worse than the default candidate in all three objectives and better than the default candidate in two objectives. Fig. 51 and 52 show the maps generated by the default, best and worst candidates from experiment setup 3 for DC and DR respectively. For the SF approach, despite there being no convergence to an optimal solution, the best candidate was able to generate a better quality map with lesser number of corners and lower proportion of occupied grids than the map generated by the default candidate.Using multi-objective GA to optimise the values of RTAB-Map parameters aided in obtaining the optimal set of parameter values for the QBot2 robot.

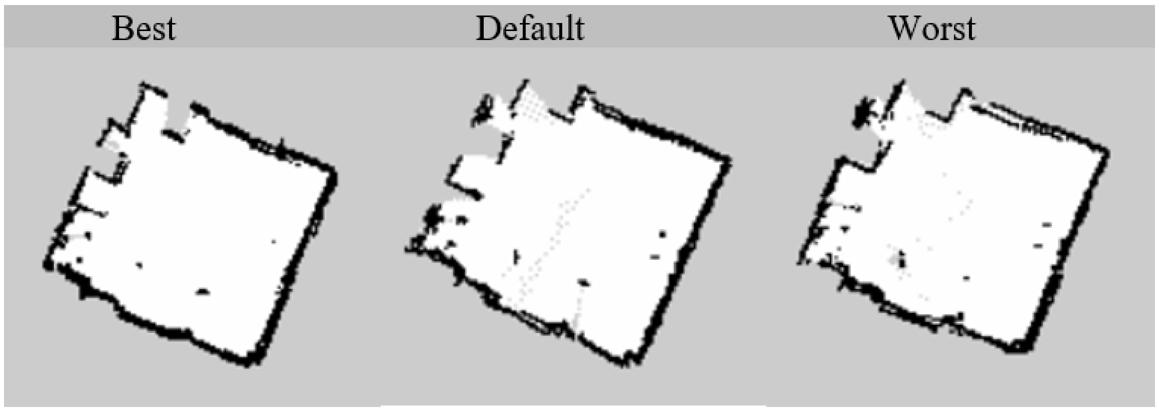


Figure 51 Maps for Best, Default and Worst candidates for DC.

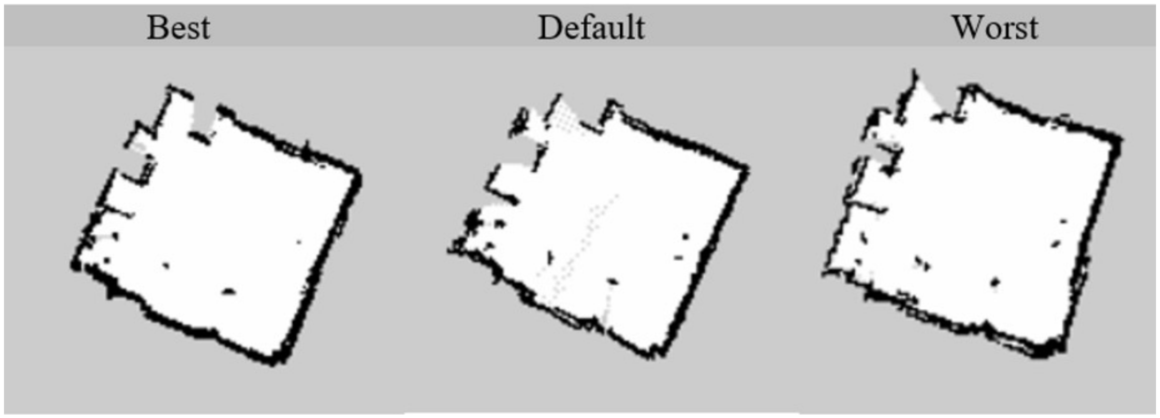


Figure 52 Maps for Best, Default and Worst candidates for DR.

CHAPTER 7 CONCLUSION

In this thesis, parameter optimisation for the RTAB-Map package has been explored using different multi-objective GA mechanisms- Dominance Count, Dominance Rank and Switching Fitness, in three sets of simulation experiments.

- In the first set of experiments where two objectives were optimised considering constant odometry for the entire setup, it is observed that all the three multi-objective GA methods are able to fairly converge to an optimal solution.
- In the second set of experiments where three objectives were optimised considering no motion and measurement errors, it is observed that the Dominance Count and Dominance Rank methods are able to converge to an optimal solution. However, the Switching Fitness, which is a pseudo multi-objective GA method, is unable to converge to a solution that optimises all three objectives equally well; although it is able to optimise each objective individually.
- In the third set of experiments three objectives were optimised with noise in motion and measurement to emulate real world, and it is observed that while the Dominance Count method provides one optimal solution, the Dominance Rank method provides multiple optimal solutions which offers a trade-off between the objectives. However, the Switching Fitness method is unable to handle this complex multi-objective problem effectively.

A sample set of best and worst candidates were identified from the above sets of simulation experiments and tests were conducted with a physical QBot2 robot using these candidates. When the best candidates from the Dominance Count and Dominance Rank methods of the third set of experiments were used, the QBot2 generated more accurate maps compared to the map generated when the default RTAB-Map parameters were used.

The results indicate that for a combination of a given environment, robot and SLAM package, implementing a Multi-Objective Genetic Algorithm to optimize the parameters used by that SLAM package would enable the robot to explore the environment more effectively and identify more frontiers resulting in more accurate maps.

While there are multiple packages available to implement the different SLAM approaches, parameter optimisation using a GA does not appear often in the available literature and applying multi-objective GA with SLAM is almost unheard of. For new researchers in SLAM, relying on historical data from literature might not be a practical approach to proceed with. Additionally, different robots come with different sensors. The default set of parameters offered by the available SLAM packages would not be uniformly efficient for all these sensors. In such scenarios, instead of relying on available data to arrive at the optimal set of parameters for any experimental setup, it makes sense to apply the proposed multi-objective GA to obtain the optimal values of parameters for the required set of sensors. This would enable the researcher in achieving a more efficient implementation of SLAM suitable to their specific requirements.

7.1 DIRECTION FOR FUTURE RESEARCH

In future, to obtain the optimal set of parameters for varied scenarios, this algorithm can be tested

- in different environments,
- with different robots, and
- with different ROS SLAM packages.

This algorithm can be built upon and made package-independent by testing it with other SLAM packages available in ROS.

With fuzzy logic, it can additionally be tested in different SLAM approaches for odometry and loop closure optimisation to ultimately replace the various Bayesian filters used, as suggested in the "Integrated fuzzy logic and genetic algorithmic approach for simultaneous localization and mapping of mobile robots" paper [49].

BIBLIOGRAPHY

[1] R. Siegwart, I.R. Nourbakhsh and D. Scaramuzza, Ed., *Introduction to Autonomous Mobile Robots.* 2nd ed. Cambridge, Mass.: MIT Press, 2011.

[2] V. Trianni and M. López-Ibáñez, “Advantages of Task-Specific Multi-Objective Optimisation in Evolutionary Robotics,” *PloS One,* vol. 10, no. 8, p. e0136406, Aug. 2015. [Online]. Available: doi: 10.1371/journal.pone.0136406 [Accessed: Jan. 1, 2019].

[3] R. Smith, M. Self and P. Cheeseman, “Estimating uncertain spatial relationships in robotics,” in *UAI ’86 Proceedings of the Second Conference on uncertainty in Artificial Intelligence*, Philadelphia, PA, AUAI Press Corvallis, Oregon, August 8 - 10, 1986, pp. 267-288. [Online]. Available: arXiv:1304.3111 [Accessed: Jan. 1, 2019].

[4] H. Durrant-Whyte and T. Bailey, “Simultaneous localization and mapping: Part I,” *IEEE Robotics & Automation Magazine*, vol. 13, no. 2, pp. 99-110, Jun. 2006. [Online]. Available: doi: 10.1109/MRA.2006.1638022 [Accessed: Jan. 2, 2019].

[5] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit, “Particle filter SLAM 2.0: An Improved Particle Filtering Algorithm for Simultaneous Localization and Mapping that Provably Converges,” in *IJCAI ’03 Proceedings of the 18th International Joint Conference on Artificial Intelligence*, Acapulco, Mexico, Morgan Kaufman Publishers Inc. San Francisco, Aug. 9 – 15, 2003, pp. 1151-1156. [Online]. Available: http://robots.stanford.edu/papers/Montemerlo03a.pdf [Accessed: Jan. 6, 2019].

[6] M. Montemerlo and S. Thrun, “Simultaneous localization and mapping with unknown data association using FastSLAM,” in *ICRA '03 Proceedings of the IEEE International Conference on Robotics and Automation,* vol. 2, Taipei, Taiwan, IEEE, September 14 - 19, 2003, pp. 1985-1991. [Online]. doi: 10.1109/ROBOT.2003.1241885 [Accessed: Jan. 7, 2019].

[7] M. Quigley, B. Gerkey and W. D. Smart, *Programming robots with ROS*, 1st ed. Sebastopol: O'Reilly & Associates Inc., 2015.

[8] J. Arents, R. Cacurs and M. Greitans, “Integration of Computervision and Artificial Intelligence Subsystems with Robot Operating System Based Motion Planning for Industrial Robots,” *Automatic Control and Computer Sciences*, vol. 52, no. 5, pp. 392-401, Sep. 2018. [Online]. Available: doi: 10.3103/S0146411618050024 [Accessed: Jan. 21, 2019].

[9] A. Hellmund, S. Wirges, O. Tas, C. Bandera and N. Salscheider, “Robot operating system: A modular software framework for automated driving,” in *ITSC ’16 Proceedings of the 19th International Conference on Intelligent Transportation Systems*, Rio de Janeiro, Brazil, IEEE, Nov. 1 – 4, 2016, pp. 1564-1570. [Online]. doi: 10.1109/ITSC.2016.7795766 [Accessed: Jan. 21, 2019].

[10] Open Source Robotics Foundation, *ROS Start Guide*, Ros.org, Feb. 2018. [Online]. Available: http://wiki.ros.org/ROS/StartGuide [Accessed: Jan. 24, 2019].

[11] S. Kohlbrecher, O. von Stryk, J. Meyer and U. Klingauf, "A flexible and scalable SLAM system with full 3D motion estimation," 2011 IEEE International Symposium on Safety, Security, and Rescue Robotics, Kyoto, 2011, pp. 155-160. [Online]. Available: 10.1109/SSRR.2011.6106777 [Accessed: Jul 05, 2020].

[12] S. Thrun, W. Burgard and D. Fox, *Probabilistic robotics*. Cambridge, Mass.: MIT Press, 2006.

[13] M. Montemerlo and S. Thrun, *FastSLAM a scalable method for the simultaneous localization and mapping problem in robotics*. Vol. 27. Berlin: Springer, 2007. [Online]. Available: https://link.springer.com/content/pdf/10.1007%2F978-3-540-46402-0.pdf [Accessed: Jan. 2, 2019].

[14] A. Doucet, N. De Freitas, K. Murphy and S. Russell, "Rao-Blackwellised Particle Filtering for Dynamic Bayesian Networks," in *UAI '00 Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence*, San Francisco, CA, Morgan Kaufman Publishers Inc., June 30 – July 3, 2000, pp. 176-183. [Online]. Available: https://arxiv.org/ftp/arxiv/papers/1301/1301.3853.pdf [Accessed: Jan. 3, 2019].

[15] G. Grisettiyz, C. Stachniss and W. Burgard, "Improving Grid-based SLAM with Rao-Blackwellized Particle Filters by Adaptive Proposals and Selective Resampling," Proceedings of the 2005 IEEE International Conference on Robotics and Automation, Barcelona, Spain, 2005, pp. 2432-2437. [Online]. Available: 10.1109/ROBOT.2005.1570477 [Accessed: Jul 05, 2020].

[16] G. Grisetti, C. Stachniss and W. Burgard, "Improved Techniques for Grid Mapping With Rao-Blackwellized Particle Filters," in IEEE Transactions on Robotics, vol. 23, no. 1, pp. 34-46, Feb. 2007. [Online]. Available: 10.1109/TRO.2006.889486 [Accessed: Jul 05, 2020].

[17] K. Konolige, G. Grisetti, R. Kümmerle, W. Burgard, B. Limketkai and R. Vincent, "Efficient Sparse Pose Adjustment for 2D mapping," 2010 IEEE/RSJ International Conference on Intelligent Robots and Systems, Taipei, 2010, pp. 22-29. [Online]. Available: 10.1109/IROS.2010.5649043 [Accessed: Jul 05, 2020].

[18] E. B. Olson, "Real-time correlative scan matching," 2009 IEEE International Conference on Robotics and Automation, Kobe, 2009, pp. 4387-4393. [Online]. Available: 10.1109/ROBOT.2009.5152375 [Accessed: Jul 05, 2020].

[19] M. Labbé and F. Michaud, "Memory management for real-time appearance-based loop closure detection," in *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems*, San Francisco: IEEE, Sep 2011, pp. 1271–1276. [Online]. Available: 10.1109/IROS.2011.6094602 [Accessed: Jul 31, 2019].

[20] M. Labbé and F. Michaud, “Appearance-Based Loop Closure Detection for Online Large-Scale and Long-Term Operation,” *IEEE Transactions on Robotics*, vol. 29, no. 3, Jun 2013, pp. 734-745. [Online]. Available: IEEE Xplore, doi: 10.1109/TRO.2013.2242375 [Accessed: Jul 31, 2019].

[21] M. Labbé and F. Michaud, “RTAB-Map as an Open-Source Lidar and Visual SLAM Library for Large-Scale and Long-Term Online Operation,” *Journal of Field Robotics*, vol. 36, no. 2, Oct. 2018, pp. 416–446. [Online]. Available: Wiley Online Library, doi: https://doi.org/10.1002/rob.21831 [Accessed: Jul 31, 2019].

[22] *QBot2 for QUARC Set Up and Configuration Manual*, Quanser Inc., Markham Ontario, 2017. [Online]. Available: https://www.quanser.com/courseware-resources/?fwp_resource_types=manuals&fwp_resource_related_products=1722 [Accessed: Jan. 16, 2019].

[23] G. Dudek and M. Jenkin, *Computational Principles of Mobile Robotics*, 2nd ed. New York: Cambridge University Press, 2011.

[24] G. Campion, G. Bastin and B. Dandrea-Novel, “Structural properties and classification of kinematic and dynamic models of wheeled mobile robots,” *IEEE Transactions on Robotics and Automation*, vol. 12, no. 1, pp. 47-62, Feb. 1996. [Online]. Available: doi: 10.1109/70.481750 [Accessed: Jan. 15, 2019].

[25] *Everything about STMicroelectronics’ 3-axis digital MEMS gyroscopes*, Technical Article TA0343, Rev 1, STMicroelectronics, 2011. [Online]. Available: https://www.elecrow.com/download/TA0343.pdf [Accessed: Jan. 16, 2019].

[26] *Kinect sensor manual and warranty*, Microsoft Corporation, 2010. [Online]. Available: http://download.microsoft.com/download/f/6/6/f6636beb-a352-48ee-86a3-abd9c0d4492a/kinectmanual.pdf [Accessed: Jan. 16, 2019].

[27] M. Kassir and M. Palhang, “Novel qualitative visual odometry for a ground: Vehicle based on funnel lane concept,” in *MVIP ’17 10th Iranian Conference on Machine Vision and Image Processing*, Isfahan, Iran, IEEE, Nov. 22 – 23, 2017, pp. 182-187. [Online]. doi: 10.1109/IranianMVIP.2017.8342345 [Accessed: Jan. 17, 2019].

[28] D. Nister, O. Naroditsky and J. Bergen, “Visual odometry,” in *CVPR ‘04 Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 1, Washington, DC, IEEE, Jun. 27 – Jul. 2, 2004. [Online]. doi: 10.1109/CVPR.2004.1315094 [Accessed: Jan. 17, 2019].

[29] D. Scaramuzza and F. Fraundorfer, "Visual Odometry [Tutorial]," *IEEE Robotics & Automation Magazine*, vol. 18, no. 4, pp. 80-92, Dec. 2011. [Online]. Available: doi: 10.1109/MRA.2011.943233 [Accessed: Jan. 18, 2019].

[30] D. G. Lowe, “Object recognition from local scale-invariant features,” in *Proceedings of the Seventh IEEE International Conference on Computer Vision*, Kerkyra, Greece, IEEE, Sep. 20 – 27, 1999. [Online]. doi: 10.1109/ICCV.1999.790410 [Accessed: Jan. 18, 2019].

[31] D. G. Lowe, “Distinctive Image Features from Scale-Invariant Keypoints,” *International Journal of Computer Vision*, vol. 60, no. 2, pp. 91-110, Nov. 2004. [Online]. Available: doi: 10.1023/B:VISI.0000029664.99615.94 [Accessed: Jan. 18, 2019].

[32] H. Bay, T. Tuytelaars and L. Van Gool, “SURF: Speeded up robust features,” in *ECCV ’06 Proceedings of the 9th European Conference on Computer Vision*, vol. 3951, Graz, Austria, Springer Verlag, May 7 – 13, 2006, pp. 404-417. [Online]. doi: 10.1007/11744023_32 [Accessed: Jan. 19, 2019].

[33] Tully Foote, Michael Ferguson and Melonee Wise, *Turtlebot Tutorials*, Ros.org, 2016. [Online]. Available: http://wiki.ros.org/turtlebot/Tutorials/indigo [Accessed: Jul 30, 2019].

[34] *Oracle VM VirtualBox User Manual*, version 6.0.10, Oracle Corporation, Dec. 2019. [Online]. Available: https://download.virtualbox.org/virtualbox/6.0.10/UserManual.pdf [Accessed: July 20, 2019].

[35] Canonical Ltd., *Ubuntu 16.04.5 LTS (Xenial Xerus)*, Ubuntu releases, 2018. [Online]. Available: http://releases.ubuntu.com/16.04/ [Accessed: May 11, 2018].

[36] L. Joseph, *Robot operating system for absolute beginners: Robotics programming made easy*. Berkeley, CA: Apress, 2018.

[37] T. Foote, “Tf: The transform library,” in *TePRA ’13 Proceedings of the IEEE International Conference on Technologies for Practical Robot Applications*, Woburn, MA, USA, IEEE, Apr. 22 – 23, 2013. [Online]. doi: 10.1109/TePRA.2013.6556373 [Accessed: Jan. 27, 2019].

[38] Open Source Robotics Foundation, *Gazebo Tutorials*, Gazebo, 2014. [Online]. Available: http://gazebosim.org/tutorials [Accessed: Jan 3, 2019].

[39] Open Source Robotics Foundation, *Kobuki Tutorials*, Ros.org, Mar. 2017. [Online]. Available: http://wiki.ros.org/kobuki/Tutorials [Accessed: May 20, 2018].

[40] Mathieu Labbe, *rtabmap_ros*, Ros.org, 2019. [Online]. Available: http://wiki.ros.org/rtabmap_ros [Accessed: Aug 1, 2019].

[41] Dave Hershberger, David Gossow and Josh Faust, *rviz*, Ros.org, 2016. [Online]. Available: http://wiki.ros.org/rviz [Accessed: Aug 1, 2019].

[42] B. Yamauchi, “A Frontier-Based Approach for Autonomous Exploration,” in *Proc IEEE International Symposium on Computational Intelligence in Robotics and Automation*, Monterey: IEEE, Jul 1997. [Online]. Available: 10.1109/CIRA.1997.613851 [Accessed: Aug 1, 2019].

[43] Jiri Horner, *explore_lite*, Ros.org, 2017. [Online] Available: http://wiki.ros.org/explore_lite [Accessed: Aug 1, 2019].

[44] Open Source Robotics Foundation, *freenect_stack*, Ros.org, Dec. 2012. [Online]. Available: https://wiki.ros.org/freenect_stack [Accessed: May 14, 2018].

[45] M. Brameier and W. Banzhaf, “A comparison of linear genetic programming and neural networks in medical data mining,” *IEEE Transactions on Evolutionary Computation*, vol. 5, no. 1, pp. 17-26, Feb. 2001. [Online]. Available: doi: 10.1109/4235.910462 [Accessed: Jan. 15, 2018].

[46] S. Luke, *Essentials of Metaheuristics*, Second edition, Lulu, 2013. [Online]. Available: http://cs.gmu.edu/←-sean/book/metaheuristics/ [Accessed: Jan. 16, 2018].

[47] C.A. Coello Coello, “Evolutionary multi-objective optimization: A historical view of the field,” *IEEE Computational Intelligence Magazine*, vol. 1, no. 1, pp. 28-36, Feb. 2016. [Online]. Available: doi: 10.1109/MCI.2006.1597059 [Accessed: Jan. 17, 2018].

[48] T. Duckett, “A genetic algorithm for simultaneous localization and mapping,” in *ICRA '03 Proceedings of the IEEE International Conference on Robotics and Automation*, vol. 1, Taipei, Taiwan, IEEE, September 14 - 19, 2003, pp. 434-439. [Online]. doi: 10.1109/ROBOT.2003.1241633 [Accessed: Jan. 1, 2019].

[49] M. Begum, G.K.I. Mann and R.G. Gosine, “Integrated fuzzy logic and genetic algorithmic approach for simultaneous localization and mapping of mobile robots,” *Applied Soft Computing Journal,* vol. 8, no. 1, pp. 150-165, Jan. 2008. [Online]. Available: doi: 10.1016/j.asoc.2006.11.010 [Accessed: Jan. 1, 2019].

[50] A. Bakdi, A. Hentout, H. Boutami, A. Maoudj, O. Hachour and B. Bouzouia, “Optimal path planning and execution for mobile robots using genetic algorithm and adaptive fuzzy-logic control,” *Robotics and Autonomous Systems*, vol. 89, pp. 95–109, Mar. 2017. [Online]. Available: doi: 10.1016/j.robot.2016.12.008 [Accessed: Jan. 18, 2018].

[51] H. Moravec and A. Elfes, “High resolution maps from wide angle sonar,” in *ICRA’85 Proceedings of the IEEE International Conference on Robotics and Automation*, St. Louis, MO, USA, IEEE, Mar. 25 – 28, 1985, pp 116-121. [Online]. Available: 10.1109/ROBOT.1985.1087316 [Accessed: Jan. 27, 2019].

[52] A.E. Eiben,and J.E. Smith, *Introduction to Evolutionary Computing*, 2nd edition, Berlin, Heidelberg: Springer, 2015. [E-book]. Available: https://link.springer.com/content/pdf/10.1007%2F978-3-662-44874-8.pdf [Accessed: Jan. 20, 2018].

[53] M. Heywood. Introducing multi-objective GP. (2017, Winter). CSCI6506. Halifax, Canada: Dalhousie University.

[54] S. Doncieux and J.-B. Mouret. Behavioral diversity with multiple behavioral distances. *In IEEE Congress on Evolutionary Computation, pages 1427–1434, 2013*

[55] A. Filatov, A. Filatov, K. Krinkin, B. Chen and D. Molodan, "2d slam quality evaluation methods," Proceedings of the 2017 21st Conference of Open Innovations Association, Helsinki, Finland, Nov. 2017, pp. 120–126. [Online]. Available: arXiv:1708.02354 [Accessed: Aug 20, 2020].

[56] S. Suzuki et al., "Topological structural analysis of digitized binary images by border following," Computer vision, graphics, and image processing, vol. 30, no. 1, pp. 32–46, 1985. [Online]. Available: https://doi.org/10.1016/0734-189X(85)90016-7 [Accessed: Aug 20, 2020].