TRIBHUVAN UNIVERSITY
INSTITUTE OF ENGINEERING
PULCHOWK CAMPUS

SIMULTANEOUS LOCALIZATION AND MAPPING

(SLAM)

FINAL REPORT

By:
Aasman Bashyal (074BEX402)
Asim Maharjan (074BEX408)
Basanta Rijal (074BEX409)
Saju Khakurel (074BEX438)

DEPARTMENT OF ELECTRONICS AND COMPUTER ENGINEERING
LALITPUR, NEPAL

February, 2021

ACKNOWLEDGEMENT

We would like to express our deepest gratitude towards **Prof. Dr. Ram Krishna Maharjan, Dr. Dibakar Raj Pant** and **Mr. Anand Kumar Sah** for bestowing us upon this project. We are very much thankful towards them for giving us an opportunity to work on this project and helping us in expanding our knowledge.We are really fortunate to have the kind association as well as supervision and encouragement of our professors during the project. We are obliged to work under their guidance and even our most profound gratitude aren't enough.

We would like to pay our deep sense of gratitude to the Department of Electronics and Communication for incorporating minor project as part of our syllabus for realization of our knowledge in the real world application and giving us a great opportunity like this to carry out the project that will help us shape our career.

We would like to thank Robotics Club, Pulchowk Campus for providing the space for conduct the project and resources.

At last we would like to thank our parents for their everlasting love and support.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

EKF Extended Kalman Filter.

LIDAR Light Detection and Ranging.

MATLAB Matrix Laboratory.

ROS Robot Operating System.

SLAM Simultaneous Localization and Mapping.

ABSTRACT

Simultaneous Location and Mapping (SLAM) is a method used to numerically solve the problem of extracting map and location in an unknown environment for Mobile robot navigation. It has been widely popular due to its wide range of applications in any sort of robot motion in an unexplored environment. Various approaches have been developed for tackling this problem like probabilistic approach, feature based, graph based and so on. We are using grid-map based approach derived from Tiny Slam that is enhanced with a layer of particle filter over it. The approach resulted in a improved map with less orientation errors and can encompass even greater robot motion speed. The robot motion however doesn't encompass any control measures resulting in errors in map updates due to inertial jerk.

1. INTRODUCTION

1.1. Background

Automation is the process of using physical machines, computer software and other technologies to perform tasks that are usually done by humans. Autonomous motion for robots nowadays has become a cardinal factor in robotics since the field of robotics has been growing tremendously and enhancements in what the robot can do is the main concern in fields of robotics today. Automation in robots is the technology that allows anyone today to configure computer software, or a robot to emulate and integrate the actions of a human interacting within digital systems to execute a business process. An automated robot has to be able to sense the nearby environments and be able to react accordingly. The problem of mobile robot automation includes various sub-problems of positioning, planning and motion implementation. Positioning is a crucial part in the overall navigation process and means identifying one's location in the surrounding. However such an approach requires an initial knowledge of the environment which is not always practically feasible. As a sophisticated solution SLAM is implemented in which the knowledge of the environment and robots position both are simultaneously updated from the robots measurement itself.

Taking SLAM approach to the positioning problem, the path-planning and motion implementation follow based on the information extracted from the positioning and thus the robot can move without collision in different unknown environments. It is the computational problem of constructing or updating a map of an unknown environment while simultaneously keeping track of an agent's location within it. SLAM has been the subject of technical research for many years. But with vast improvements in computer processing speed and the availability of low-cost sensors such as cameras and laser range finders, SLAM is now used for practical applications in a growing number of fields.

One of characteristic of SLAM is that it is a continuous and discrete problem. Robot poses and object or landmark locations are continuous aspects of the SLAM problem. While sensing the environment continuously, a discrete relation between detected objects and newly detected ones needs to be made. This relation is known by correspondence and helps the robot to detect if it has been in the same location. With SLAM, a mobile robot is establishing a discrete relation between newly and previously detected objects.

SLAM methods can be classified according to the type of map that generated. The map generated can be a landmark based map or a volumetric map (also known as a grid map). Some of the steps involved in Landmark based SLAM algorithm follows are Landmark Extraction, Data association, State estimation, State update and Landmark update. Landmarks are features which can easily be re-observed and distinguished from the environment. These features are used by the robot to find out it's actual position and orientation in the given environment also termed as self localize.

The mapping algorithm that is generally used is the occupancy grid mapping which we have used in our project. The algorithm can map any arbitrary environment by dividing it into a finite number of grid cells. Grid maps use arrays (typically square or hexagonal) of discretized cells to represent a topological world, and make inferences about which cells are occupied. Typically the cells are assumed to be statistically independent in order to simplify computation. Under such assumption, $P(m_t|x_t, m_{t-1}, o_t)$ are set to 1 if the new map's cells are consistent with the observation o_t at location x_t and 0 if inconsistent.

SLAM is useful in many other applications such as navigating a fleet of mobile robots to

arrange shelves in a warehouse, parking a self-driving car in an empty spot, or delivering a package by navigating a drone in an unknown environment. Many platforms like MATLAB, Simulink, ROS provide SLAM algorithms, functions, and analysis tools to develop various applications. We can also implement simultaneous localization and mapping along with other tasks such as sensor fusion, object tracking, path planning and path following.

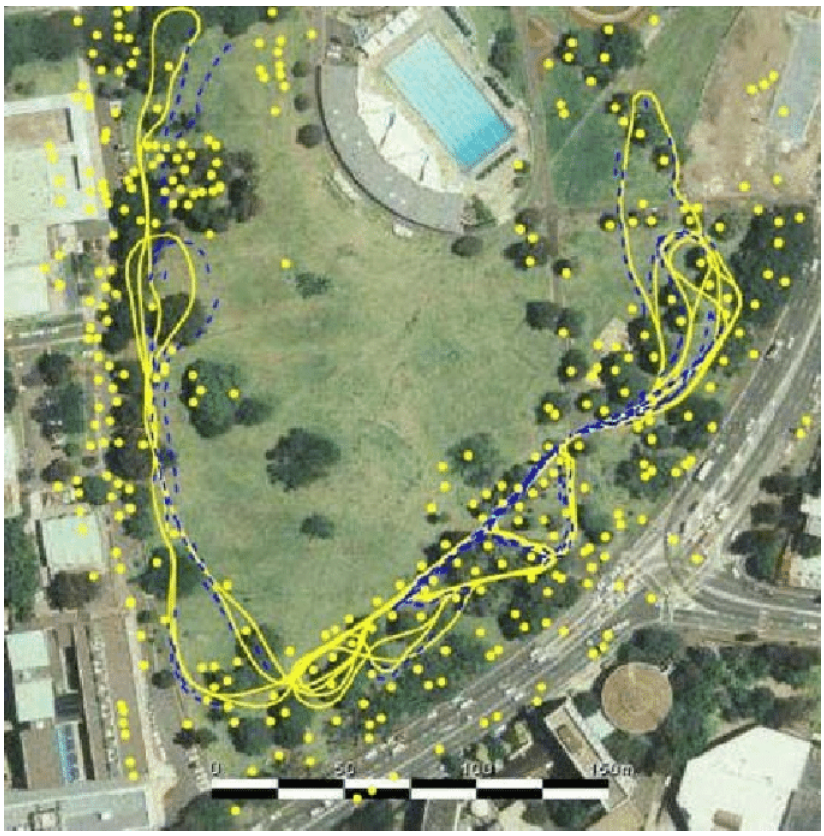

Figure 1: Map generated using Landmark Based SLAM

Figure 2: Map generated using grid map based SLAM

1.2. Objectives

1. To estimate the map and the object pose in that map for a different environments.
2. To understand and illustrate the concepts behind SLAM
3. To simulate the designed system and visualize the results.

1.3. Problem Statement

SLAM has been a hot topic under mobile robot motion finding its applications under autonomous vehicles, service robots, delivery drones and so on. SLAM proposes following challenges:

1. Localisation problem
2. Mapping problem
3. Simultaneous map and position approximation problem.
4. Test environment setup

Localisation poses the issue of finding the x,y position and orientation in a given map for a measurement. Mapping is for constructing a map and obstacles for a robot position far a given measurement. SLAM introduces iterative approximation of position and map at a time T taking the position and map at the time T-1. The uncertainty in the measurement is handled using normal probabilistic models of position and map estimate.

2. LITERATURE REVIEW

Mobile Robots are capable of moving in the surrounding and thus have found various application in various fields today among which Unmanned Ground Vehicles, Unmanned Aerial Vehicles, Autonomous underwater vehicles and Polar robots are the major categories. Going through various developments, the mobile robots have come to the phase of Autonomous Mobile Robots. The general problem of mobile robot navigation was summarized into three questions: “Where am I?,” “Where am I going?,” and “How should I get there?.” in [1].

The question of ”Where am I?” addresses the localization problem of autonomous robot control.Localization involves the task of identifying the robot position with respect to its environment.In a typical robot localization scenario, the robot uses the sensor information and already available environment map. This approach takes in noisy sensor data that give the measured pose of the robot in the environment and applies various filtering approaches to find the estimate of the actual pose of the robot and solve the localization problem [2]. Bayesian filtering can be employed as a powerful technique to address this problem [3]. However, the constraint of already knowing the map weighs heavy on the this approach to the problem.The other approach overcomes this constraint in the sense it builds the map and localizes itself in it simultaneously and hence, called SLAM. The classical age of SLAM(1986-2004) witnessed the probabilistic approaches to SLAM among which were EKF and maximum likelihood estimation. The later period, algorithmic-analysis age (2004-2015), was much involved with the study of SLAM and its fundamental properties like observability, convergence and consistency itself [4]. Presently much sophisticated approaches to this problem exist such as Visual-optometry [5], Multi robot SLAM [6], Visual place recognition [7] and so on. A simpler yet powerful approach to this problem is the GraphSLAM demonstrated in [8].

3. METHODOLOGY

3.1. System Block Diagram

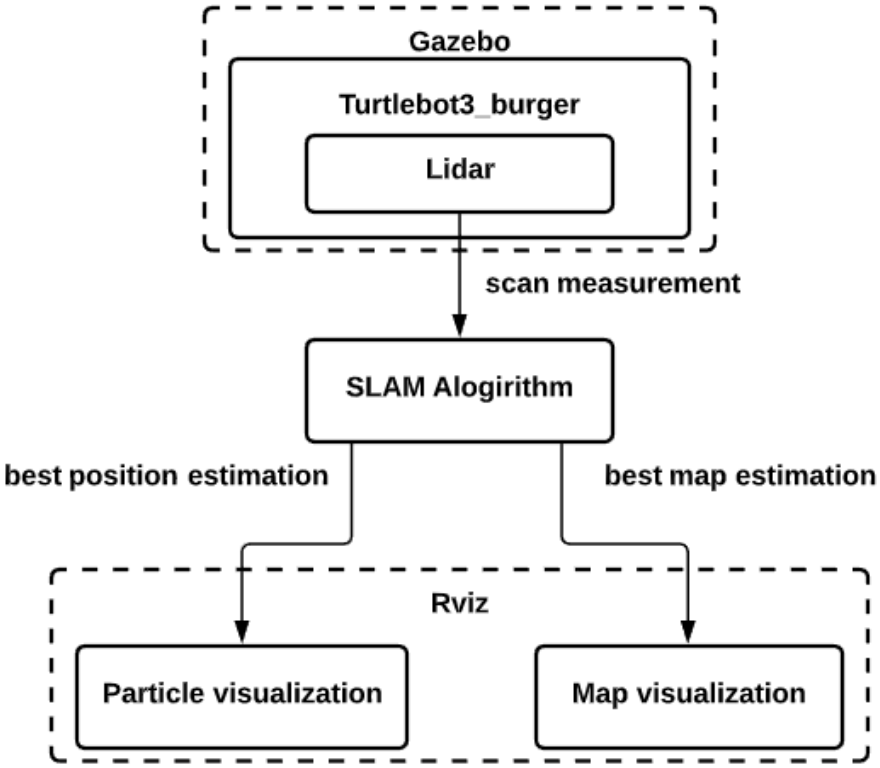

Figure 3: System Diagram

3.2. Laser Sensor

LIDAR is a method of calculating the distance of the target on which the high frequency laser pulses strike, based on the time required to return to the source from the target. A LIDAR system in total comprises of Laser Transmitter, Optical receiver, Signal detection and Data Acquisition and Control.LIDAR sends rapid pulses of laser light to the surface and calculates the time it requires to return to the source. This is used by the robot to make a 2d representation of the target. The lidar we have used is LDS-01 which is one of the affordable 360 lidar which is capable of collecting data from 360 degree surrounding which can be used for SLAM. It has USB interface so it is easily compatible with computers.

Table 1: LDS-01 Parameters

Angular Range	360°
Distance Range	120-3500 mm
Angular Resolution	1°
Scan Rate	10 Hz

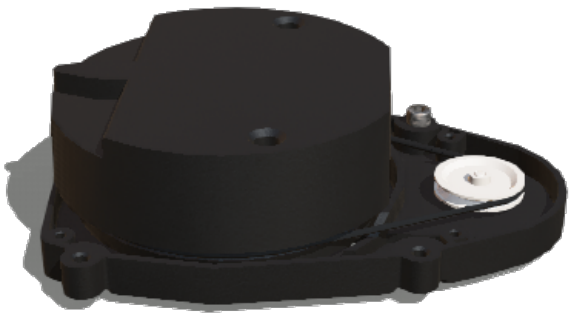

Figure 4: LDS-01

3.2.1. Laser Scan Data Processing

The laser range finder LDS-01 sends laser scan data to the turtlebot. The laser scan data is in the form of a sequence of numbers which denote the distance to the obstacle from the laser sensor at a given angle. The LDS-01 has an angular range of $360°$ with a resolution of 1 degrees. Therefore there are a total of 360 data points from the laser. If $\mathbf{z}$ is the list of distances measured by the laser range sensor, then,

$$\mathbf{z} = [z_0, z_1, ..., z_{359}]$$

where, z_0, z_1, .. etc are the individual range readings. The first distance measurement z_0 is taken at an angle of $-180°$, z_1 at an angle of $-179°$ and so on. Then, the Cartesian co-ordinates of the sensor reading is given by:

$$x_k = z_k \cos\theta_k$$
$$y_k = z_k \sin\theta_k$$

where, θ_k is the angle at which the measurement z_k was taken. These co-ordinates are with respect to the robot's frame of reference. As shown in figure 5, the vectors $\mathbf{i}$ and $\mathbf{j}$ and the point P_0 make up the global (map) frame of reference, while $(\mathbf{v_1}, \mathbf{v_2}, P_1)$ make up the robot's frame of reference. The transformation from the robots frame of reference to the global frame of reference can be carried out by the following matrix:

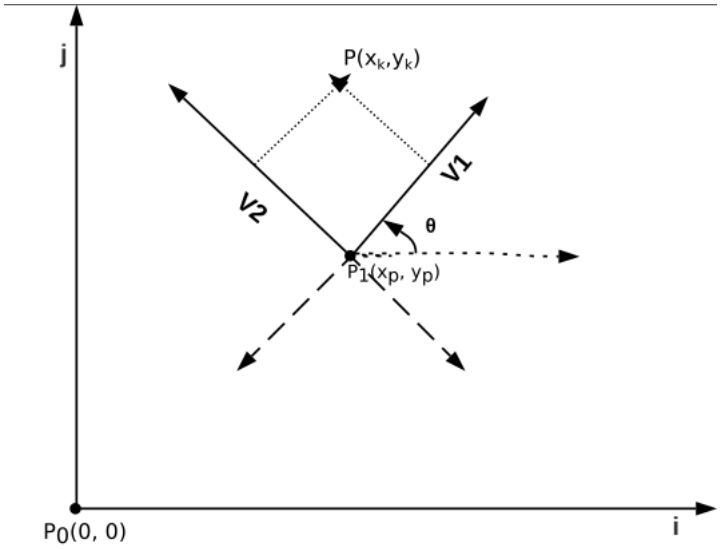

Figure 5: Map frame and Robot Frame

$$T = \begin{bmatrix} x_k & y_k & 1 \end{bmatrix} \begin{bmatrix} \cos(\theta) & \sin(\theta) & 0 \\ -\sin(\theta) & \cos(\theta) & 0 \\ x_p & y_p & 1 \end{bmatrix} \begin{bmatrix} i \\ j \\ P_0 \end{bmatrix} \tag{1}$$

which gives,

$$x = x_k \cos(\theta) - y_k \sin(\theta) + x_p$$
$$y = x_k \sin(\theta) + y_k \cos(\theta) + y_p$$

Here, (x_p, y_p, θ) is the pose of robot in the global frame of reference as shown in figure 5 and $x, y)$ is the coordinate of the point (x_k, y_k) in the global reference frame.

3.3. Robot Description

In order to build the map the LIDAR sensor must be moved in the given environment. For that purpose we have used a differential wheeled robot with two wheels connected with motors and a free turning wheel. For the sensing environment.We have used Laser Sensor 360 laser distance sensor LDS-01. For simulation purposes in gazebo we have used the ROS Turtlebot 3 package which has inbuilt LDS-01 sensor and well documented too. For the differential wheeled robot to move with given velocity we use kinematics which is explained below:

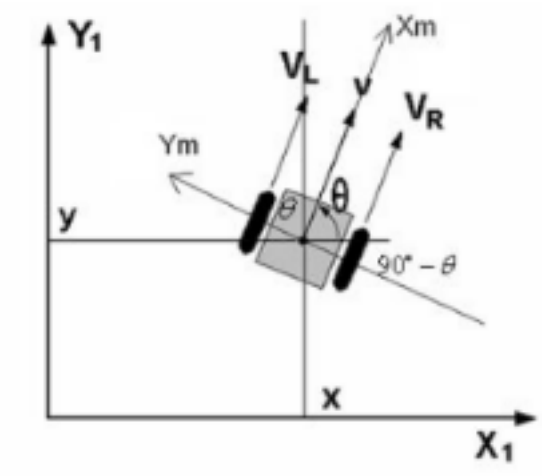

Figure 6: Kinematics for differential wheel drive

The kinematics for the two wheel differential drive is:

$$\begin{bmatrix} x_{new} \\ y_{new} \\ \theta \end{bmatrix} = \Delta t \begin{bmatrix} \cos(\theta) & 0 \\ \sin(\theta) & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ w \end{bmatrix} \tag{2}$$

where,

$$\begin{aligned} v &= (V_r + V_l)/2 \\ w &= (V_r - V_l)/L \\ V_r &= r * w_r \\ V_l &= r * w_l \\ w_r &= \text{Velocity of right wheel} \\ r &= \text{radius of the wheel} \\ w_l &= \text{Velocity of left wheel} \\ v &= \text{velocity of the robot} \end{aligned}$$

Finally,new position x_{new} and y_{new} is:

$$\begin{aligned} x_{new} &= v\cos(\theta)\Delta t \\ y_{new} &= v\sin(\theta)\Delta t \\ \theta &= w\Delta t \end{aligned}$$

3.4. SLAM

Among the many SLAM algorithms that use this approach, we have chosen the tiny SLAM algorithm for study in our project. The Tiny SLAM [9] is a SLAM method for generating grid maps. This particular SLAM method is well known for its simplicity and its quality given

the simplicity. The SLAM algorithm is very comprehensive using only simplistic models for sensors, robot motions and its mapping technique. Additionally, the source code for Tiny SLAM was easily available and was well documented in the Breezy SLAM repository. Due to these features, we selected Tiny SLAM as our SLAM algorithm of interest during this project. Besides this, various improvements have also been proposed to improve the existing tiny SLAM algorithm [10]. In general the tiny SLAM algorithm can be broken down into following simple steps:

1. Calculate the best possible position of the robot using the available laser range data
2. Update the map according to the laser range data and the best estimated position that was calculated in step 1.

In Breezy SLAM, the robot position is represented by a single position which represents the most likely pose of the robot given the previous poses and the map posterior is represented by a single occupancy grid map which is the map generated from the most likely positions of the robot using the corresponding laser scans. Also, in our project, we have decided to make certain modifications to the tiny SLAM algorithm. Currently, we have added a particle filter to the existing algorithm and visualized the results in ROS.

3.4.1. Particle Filter

Particle Filter uses a set of particles to represent the posterior distribution of a model in a process and filter out the particles as the number of observations increase, leaving out the particles with the highest probability of survival. In our project we used the particle filter for calculating the possible location of the robot in the given map, observing the laser scan data and filtering out the particles by calculating its importance weight.

At the beginning a list of particles is defined. Each particle has the necessary attributes like 2D position, orientation (θ) weight and their own individual map. The position of particles are then updated using the Hill Climbing algorithm (described in 4), which uses random normal distribution and returns the best estimated position of the particles respective to scan data. After that the weight of each particle was calculated using the sensor model as described in 3. The calculated weight of the particles are then normalized and re-sampled.

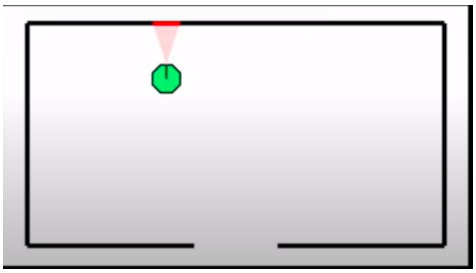

Figure 1: True position of the Robot

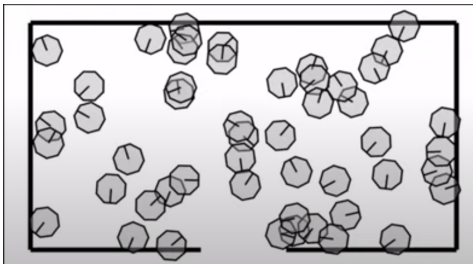

Figure 2: Particles representing the position of the robot

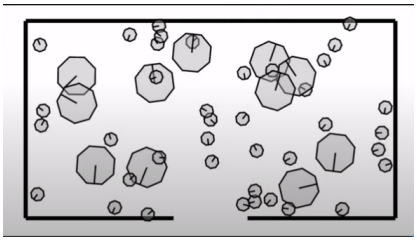

Figure 3: The particles with their weight represented, larger size represents larger weight.

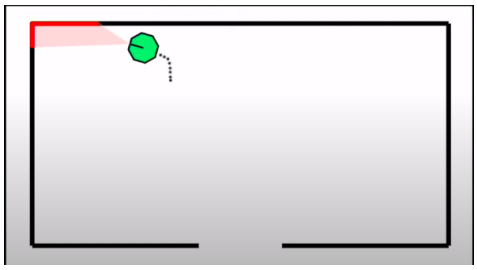

Figure 4: Movement of the robot

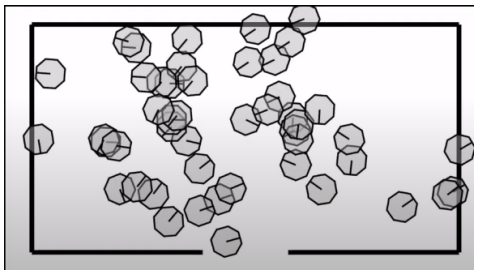

Figure 5: Movement of the robot causing a change in the position of the particles

Figure 7: Basic Working of Particle Filter

The algorithm for the implementation of particle filter is given in algorithm 1.

Algorithm 1 Particle Filter

Input: $\mathbf{X_t}$ list of particles, $\mathbf{W_t}$ weight of particles, $\mathbf{M}$ Number of particles
Output: $\bar{\mathbf{X}}_\mathbf{t}$ list of new generation of particles

procedure PARTICLE_FILTER
    $\bar{X}_t = X_t = \phi$
    for n = 1 to M **do**
        $X_t^{[n]}$ = HILL_CLIMBING$(\mathbf{x}, \mathbf{m}, \mathbf{z})$
        $W_t^{[n]}$ = SENSOR_MODEL$(\mathbf{z}, \mathbf{x}, \mathbf{m})$
    end for
    RESAMPLER $(X_t^{[n]}, W_t^{[n]})$
end procedure

The re-sampling step 5 of the particle filter given in algorithm 1 is done using low variance re-sampling as described in [3]. The algorithm is presented in 2.

Algorithm 2 Low Variance Re-sampling

Input: $\mathbf{X_t}$ list of particles, $\mathbf{W_t}$ weight of particles $\mathbf{x}$ pose and orientation, $\mathbf{m}$ map, $\mathbf{z}$ Reading from laser scan

Output: $\bar{\mathbf{X}}_\mathbf{t}$ list of new generation of particles

procedure RESAMPLER(X_t, W_t)
    $\bar{X}_t = \phi$
    M = len(X_t)
    $r = rand(0, M^{-1})$
    $c = W_t^{[0]}$
    $i = 0$
    for m = 0 to M-1 **do**
        $u = r + m/M$
        while $u > c$ **do**
            $i = (i + 1) \mod M$
            $c = c + W_t^{[i]}$
        end while
        Add $X_t[i]$ to $\bar{X}_t$
    end for
    return $\bar{X}_t$
end procedure

3.4.2. Occupancy Grid Map

Tiny SLAM represents the map as an occupancy grid map. The occupancy grid map represents the continuous real-world map as a field of random variables, arranged in an evenly spaced grid where each random variable determines the binary probability that the corresponding grid is occupied or not occupied. Mathematically, in occupancy grid maps, the map $\mathbf{m}$ is represented as a collection of grid cells with occupancy probabilities denoted by random variable m_i. Here, each grid cell has a probability of being occupied or not occupied. In Tiny SLAM, the grid cells are given by the probability of being not occupied, i.e,

$$P(m_i) = P(m_i \text{is not occupied })$$

Occupancy Grid map makes the following assumptions about the world:

1. Each cell m_i is independent of all the other cells.
2. The world is static i.e, the map (or the environment) does not change with time
3. Each cell is represented by a binary state, i.e, it is either occupied or not occupied.

Due to these assumptions, the calculation of the posterior of the maps $p(\mathbf{m}|x, z)$ is now reduced to the calculation of the posterior over each grid cell i.e., $p(m_i|x, z)$.

Initially, all the grid cells are initialized with a probability of 0.5 which states that the grid cell has equal chances of being occupied or of being free. Once the sensor data is read, these probabilities are then updated using the map update described further below.

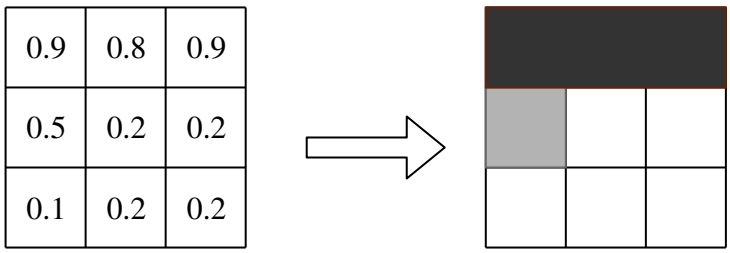

Figure 8: Occupancy Grid Map

As shown in figure 8, the resulting map of the grid with probability values on the left is given in the right.

3.4.3. Sensor Model

The sensor model or the observation model, denoted by $p(z_t|x_t, m)$, is the likelihood of getting a certain observation given our position and the current state of the map. In simpler terms, it tells us the probability of getting a certain sensor reading (e,g, measuring a certain distance from a laser range sensor) given our current pose and the map. The sensor model is used to calculate the importance weight of our particles as well as to perform the hill climbing search for the best possible robot position. Tiny SLAM uses a relatively simple sensor model, where each laser range data is assumed independent of other laser range data. The algorithm for the sensor model is as shown below:

Algorithm 3 Calculation of the likelihood of sensor model

Input: $\mathbf{z_t}$ Reading from laser scan, $\mathbf{m_t}$ map
Output: $\mathbf{x_t}$ position and orientation

procedure SENSOR_MODEL($\mathbf{z_t}$, $\mathbf{x_t}$, $\mathbf{m_t}$)
    $sum = 0$
    $n = 0$
    for each reading z_t^k in $\mathbf{z_t}$ **do**
        if z_t^k is finite **then**
            Calculate the map coordinates (x,y) of reading z_t^k using the pose $\mathbf{x_t}$
            value = occupancy probability of the grid cell at position (x,y)
            $sum = sum + value$
            $n = n + 1$
        end if
    end for
    if $n > 0$ **then**
        return $(sum * 1024)/n$
    else
        return -1
    end if
end procedure

The calculation of map coordinates of a reading z_t^k is done by using the transformation described in section 3.2.1.

3.4.4. Best Estimate of Particle Position

Tiny SLAM uses the hill climbing approach in order to search for the best possible position that matches the current laser scan data. We use a random hill climbing approach which is described as in algorithm 4. Here, the algorithm iterates max number of times and returns the position for which the sensor model gives the least value. The robot poses are searched randomly by using a normal distribution. The parameters $\sigma_{xy}, \sigma_\theta$ can be used to control the variance of the search. Higher value of these parameters lead to a much larger search space.

Algorithm 4 Calculation of the the best estimated position using Hill Climbing

Input: $\mathbf{x_{t-1}}$ Previous position and orientation
$\mathbf{z_t}$ Current reading, $\mathbf{m_t}$ map
max Number of iterations
Output: Best estimated position of the robot

procedure HILL_CLIMBING($\mathbf{x_{t-1}}, \mathbf{z_t}, \mathbf{m_t}, max$)
  $iterations = 0$
  $best = \mathbf{sensor\_model}(\mathbf{z_t}, \mathbf{x_{t-1}}, \mathbf{m_t})$
  $current = 0$
  $pose = \mathbf{x_{t-1}}$
  while $iterations < max$ **do**
    $x^{'} = \mathbf{random\_normal}(pose.x, \sigma_{xy})$
    $y^{'} = \mathbf{random\_normal}(pose.y, \sigma_{xy})$
    $\theta^{'} = \mathbf{random\_normal}(pose.\theta, \sigma_\theta)$
    $current = \mathbf{sensor\_model}(\mathbf{z_t}, \mathbf{x'}, \mathbf{m_t})$
    if $current \neq -1$ and $current < best$ **then**
      $current = best$
      $pose = (x', y', \theta')$
    end if
    $iterations = iterations + 1$
  end while
  return $pose$
end procedure

Here, the $\mathbf{random\_normal}(\mu, \sigma)$ is a function which draws a sample from the normal distribution given by $\mathcal{N}(\mu, \sigma)$.

3.4.5. Map Update

Each particle has its individual map and the map is updated after each scan. At first each particle is assigned with it's individual map and a map handle is assigned to the particles instead of the map itself. A map is divided into a number of cells known as grid cells and the size of each grid cell is given by (size of map)/(number of cells). This ratio is also known as the resolution of the map and the map is updated on the basis of probability of occupancy

of each grid cell. More the probability of the cell occupancy, higher the chances of that cell being free. To perform the map update, first the updated particle position is taken by using the most recent scan. The line from the position of the particle to the (x, y) position of the scan gives the line through which the laser traverses. The, we use the Bresenham's Line Algorithm to generate all the grid cells that the line passes through. The Bresenham algorithm is an incremental scan conversion algorithm. It is an efficient algorithm to render a line with pixels. It is commonly used to draw line primitives in a bitmap image, as it uses only integer addition, subtraction and bit shifting, all of which are very cheap operations. It is therefore a very efficient method. The Bresenham's line algorithm is given in algorithm 5.

Algorithm 5 Bresenham's Line Algorithm

Input: (x_1, y_1) Starting coordinate of line, (x_2, y_2) Ending coordinate of line
Output: X list of (x,y) grid cells that contains the line,

procedure BLA((x_1, y_1), (x_2, y_2))
    $dx = |x_2 - x_1|$
    $dy = |y_2 - y_1|$
    $incx = (x_2 - x_1)/dx$
    $incy = (y_2 - y_1)/dy$
    $x = x_1, y = y_1$
    Add (x, y) to X
    if $dx > dy$ **then**
        $p = 2 * dy - dx$
        for $i = 1todx$ **do**
            if $p < 0$ **then**
                $p = p + 2 * dy$
                $x = x + incx$
            else
                $p = p + 2 * (dy - dx)$
                $x = x + incx$
                $y = y + incy$
            end if
            Add (x, y) to X
        end for
    else
        $p = 2 * dx - dy$
        for $i = 1tody$ **do**
            if $p < 0$ **then**
                $p = p + 2 * dx$
                $y = y + incy$
            else
                $p = p + 2 * (dx - dy)$
                $x = x + incx$
                $y = y + incy$
            end if
            Add (x, y) to X
        end for
        return X
    end if
end procedure

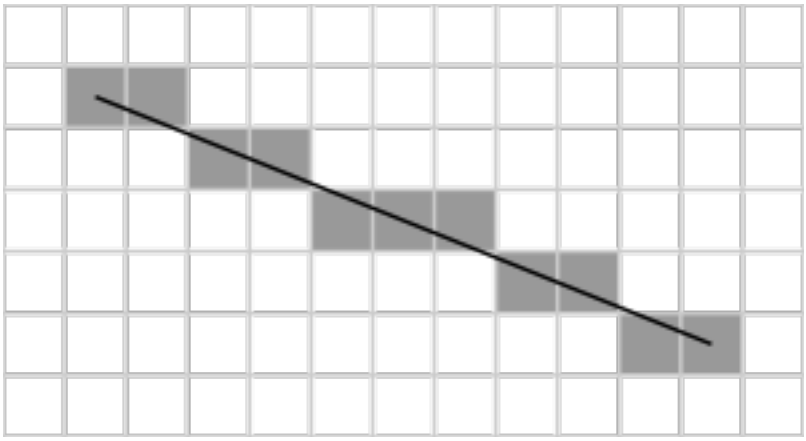

Figure 9: Bresenham's Line Algorithm converting line to corresponding pixels.

Then, for each grid cell, the probability of a grid cell being occupied is updated by using the formula:

$$P(m_i^{k+1}) = (1 - \alpha) * P(m_i{}^k) + \alpha * P(m_i|z)$$

where,

$$\begin{aligned} P(m_k^{k+1}) &= \text{New probability value of occupancy in the grid cell i} \\ P(m_k^k) &= \text{Current probability value of occupancy in the grid cell i} \\ \alpha &= \text{Belief in the new value of the sensor reading} \\ P(m_i|z) &= \text{Probability of grid cell i being occupied given reading z} \end{aligned}$$

Here the value of α allows us to set the priority for replacing the new probability with the older probability value of the grid cell.The value lies between 0 and 1. The expression $P(m_i|z)$ is known as the inverse sensor model. The value of the expression depends on the distance of the grid cell from the reading of the sensor.

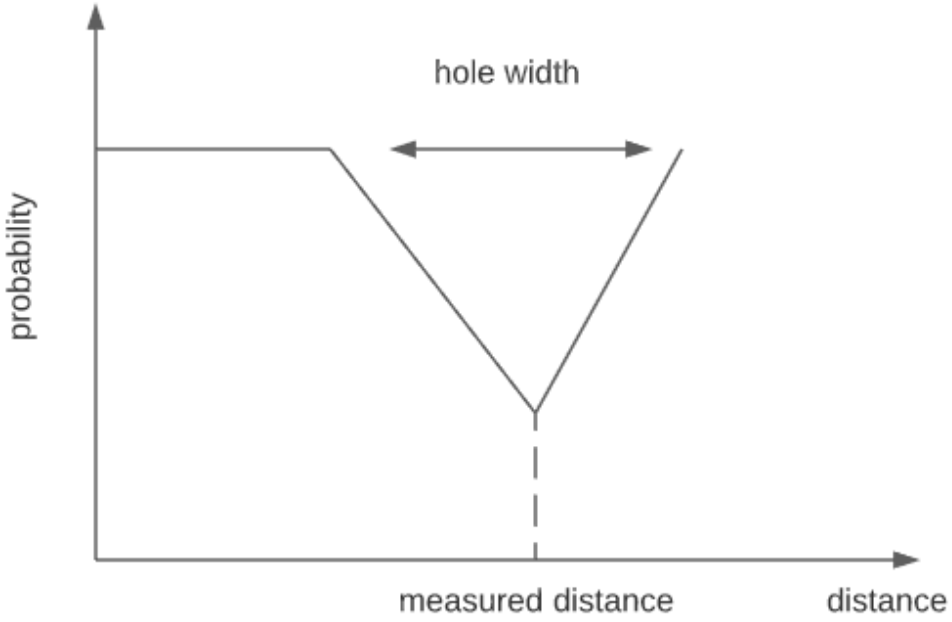

Figure 10: $P(\mathbf{m}|z)$, i.e, the probability of the grid cells that are intersected by the laser scan versus the distance from the scan measurement.

Here, the figure shows the probability of the grid cell not being occupied varying with the distance given by the laser scan.

3.5. Simulation

SLAM as we have devised in this project requires laser-scan data coming from the laser range finder sensor or LIDAR attached to the mobile robot. The laser-scan data corresponds to the environment of the mobile robot and is used to develop the occupancy grid map of the environment and localize the mobile robot in it. In practical applications, the laser data is received form LIDAR of the mobile robot and the SLAM processing is done resulting in the occupancy grid map. The obtained occupancy grid map gives the location of obstacles in the environment and that is used by path-planning algorithms to find the appropriate path from the robot's current position to the specified goal position. However, as for our project we require the laser-data and generate the occupancy grid map. So we selected Robot Operating System(ROS) as our platform to simulate the mobile robot in an environment and gather the laser scan data and also to visualize the output i.e. occupancy grid map and the particle position in it. For the world simulation we use a 3D simulation package called Gazebo and readily available turtlebot 3 packages as our environment. The real time visualization of output is done using the package Rviz.

3.5.1. ROS Framework

The essential components that constitute the working of ROS are ROS master, Nodes, Topics and Messages. ROS nodes are the processing components of ROS that perform the required computations. At a time there can be multiple nodes running in a ROS environment, each performing different types of computations. The nodes are interwoven in that the output of one computation can be the input of another. Hence, the nodes work as a system and the passing of the information is done using messages and topics. Topics are named buses for the message exchange between nodes. Thus actually a communicating node does not know the other node it is communicating with, instead subscribes to a topic that it requires. The message is a simple data structure consisting of the data that needs to be exchanged. A master is a special node in the framework that handles naming and registration of other nodes. These components of the ROS system are bundled together and distributed as packages and ROS provides lots of standard packages like Gazebo and Rviz however the user can also create own package to run a custom node.

3.5.2. ROS Packages Used

3.5.2.1. ROS Turtlebot Packages

Turtlebot 3 is a popular generic readymade mobile platform base that can be expanded to connect various sensors and actuators making it perfect for prototyping and testing applications. Turtlebot line of mobile bases enjoy a very large community support and simulation packages made available by the manufacturer because of their massive popularity. The available simulation packages are used in this project to simulate a mobile robot in a 3D environment. The wide range of available environments eased our testing.

For the simulation of our code in ROS we used few available nodes, configuration files, software models, libraries etc from turtlebot3 packages. For this project we used turtlebot3, turtlebot3_msgs and turtlebot3_simulations packages. First of all we used the turtlebot3 package for launching robot model "burger" and an environment space having some obstacles to navigate our robot. Then we used a turtlebot3_teleop_keyboard package that helped us to move our robot in the launched environment that created a cmd_velocity node. This node publishes the linear and angular velocity of the robot.

3.5.2.2. Gazebo

Gazebo is a 3D simulator tool for testing algorithms, designing robots, performing regression testing and many more. Here we used this tool to launch our robot model in a predefined environment and simulate the robot's movement by launching a teleop_keyboard launch file in ROS. Gazebo also publishes the laser scan data which was then subscribed by our subscriber node whose call back function passes the data to other functions for updating the maps.

3.5.2.3. RVIZ

Rviz is a ROS graphical interface for visualizing a lot of information, using plugins for available topics. We used rviz in our project to visualize the particles and updated maps published by the publisher. Here we selected two topics that are being published by the publisher node we created and visualized them using necessary plugins. One is particles for

generating the point clouds of particles and the other is MAP for viewing our environment seen through laser scan.

3.5.2.4. Minor

Minor is a custom package created to run the nodes that subscribe the laser scan data from the Gazebo and then process it using the devised SLAM algorithm and finally publish the particles and occupancy grid map for Rviz.

3.5.3. ROS Nodes Running

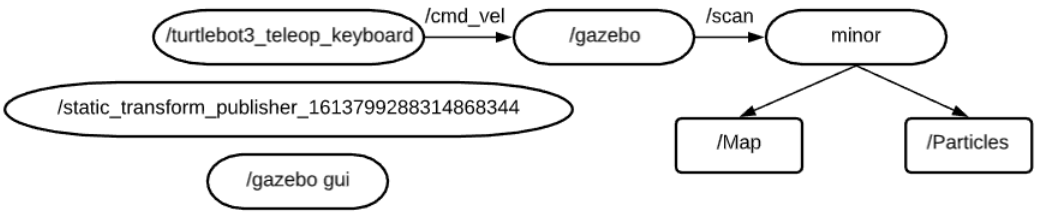

Figure 11: All the nodes currently in ROS

3.5.3.1. Turltebot3_teleop_keyboard

The turtlebot_teleop_key provides a generic keyboard teleop node. This node takes input from the keyboard for the movement of the robot in 4 directions w, a, x and d and s for stop then publishes the cmd_vel topic which is then subscribed by gazebo node. The msgs published by this node are geometry_msgs/Vector3 linear and geometry_msgs/Vector3 angular which are the output command velocity to the robot.

3.5.3.2. Static_transform_publisher_1613799288314868344

Transform(tf) is a package that lets the user keep track of multiple coordinate frames over time. It maintains the relationship between coordinate frames in a tree structure buffered in time, and lets the user transform points, vectors, etc between any two coordinate frames at any desired point in time. The static_transform_publisher publishes a static coordinate transform to tf using an x/y/z offset in meters and yaw/pitch/roll in radians. Here this publisher is used for transforming the map coordinate frames to map1 coordinates.

3.5.3.3. Gazebo_gui

The Gazebo GUI overlay can be thought of as a transparent 2D layer that sits on top of the render window. It's simply a simulating interface where ROS serves as the interface for the robot.

3.5.3.4. Gazebo

This node publishes a scan topic that gives laser scan data from the robot which is then subscribed by the minor node. Gazebo also subscribes to cmd_vel published by the teleop_keyboard and thus controls the motion of the robot in the simulation accordingly.

3.5.3.5. Minor

This node subscribes to the laser scan data under the scan topic and updates the robot location and map accordingly based on our SLAM algorithm. The updated map and position is published under map and particles topics respectively.

3.5.3.6. Rviz_1613836581962690003

This node subscribes to the map and particles topics published by minor node and displays them in the Rviz gui immediately and hence gives the real time visualization of our output.

4. EXPERIMENTAL RESULTS

4.1. Tiny SLAM with the provided data set

At the very beginning, we ran the tiny slam algorithm by using the data set provided in the BreezySLAM repository. For this, we modified BreezySLAM's program code to generate maps that were created at every step from each scan data. This was done in order to get a good grasp of the internal workings of BreezySLAM as well as to be familiar with the tool chain (e.g. compilers, debuggers, build environments etc.).

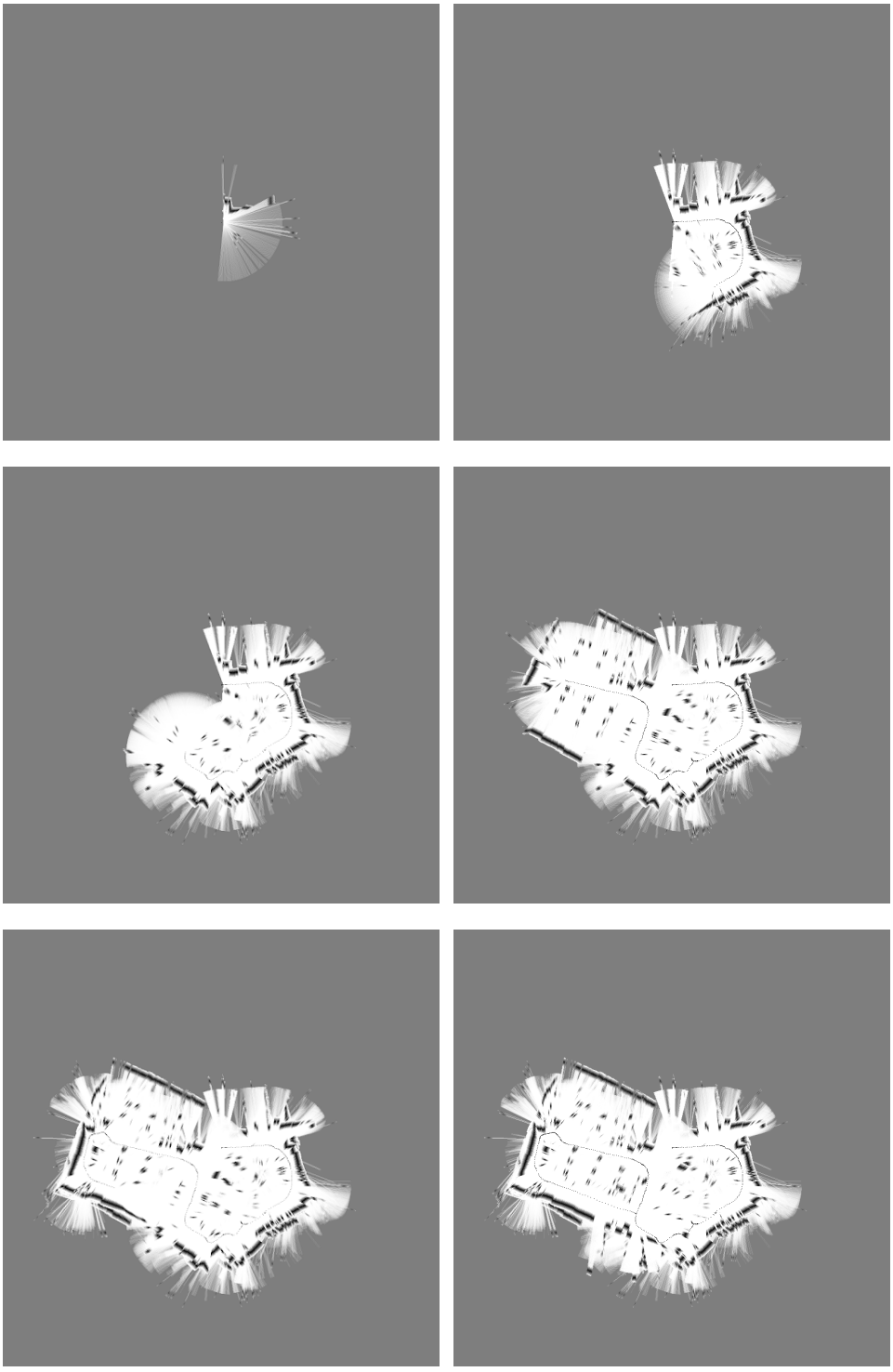

Figure 12: Steps of map generation from default data set

Once we were familiar with the BreezySLAM, we utilized BreezySLAM using the data set that was generated by the TurtleBot3 simulation in ROS. We also verified the various parameters of the laser range finder that the TurtleBot3 was using i.e, of LDS-01.

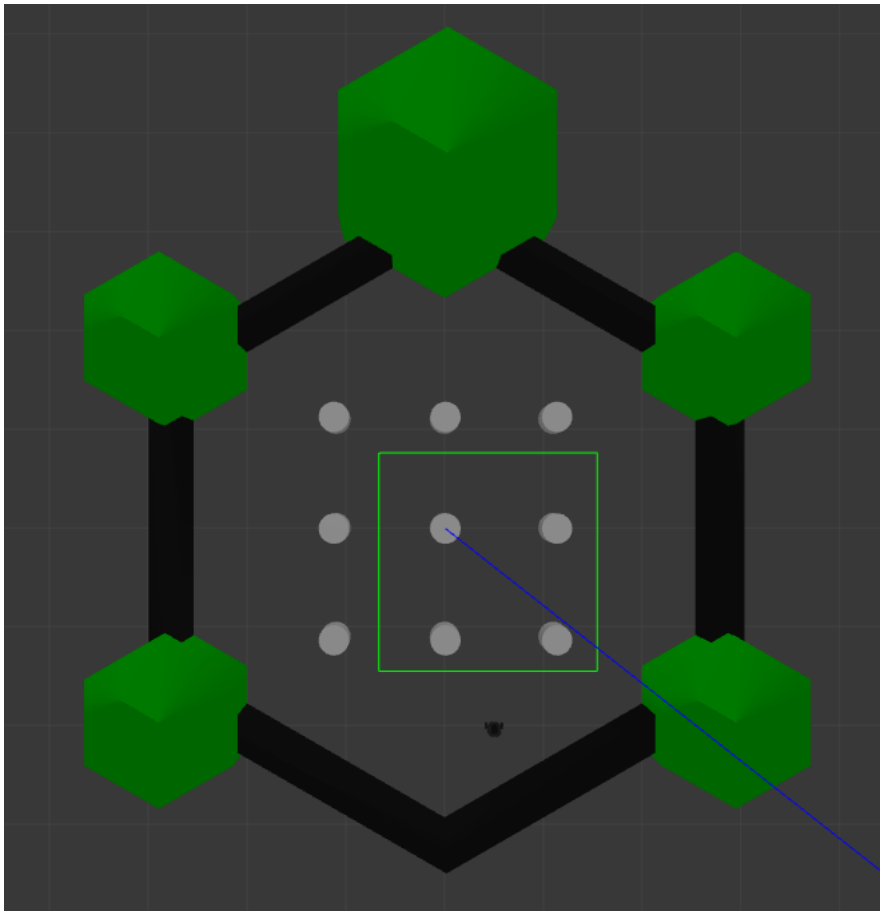

Figure 13: 3D simulation environment

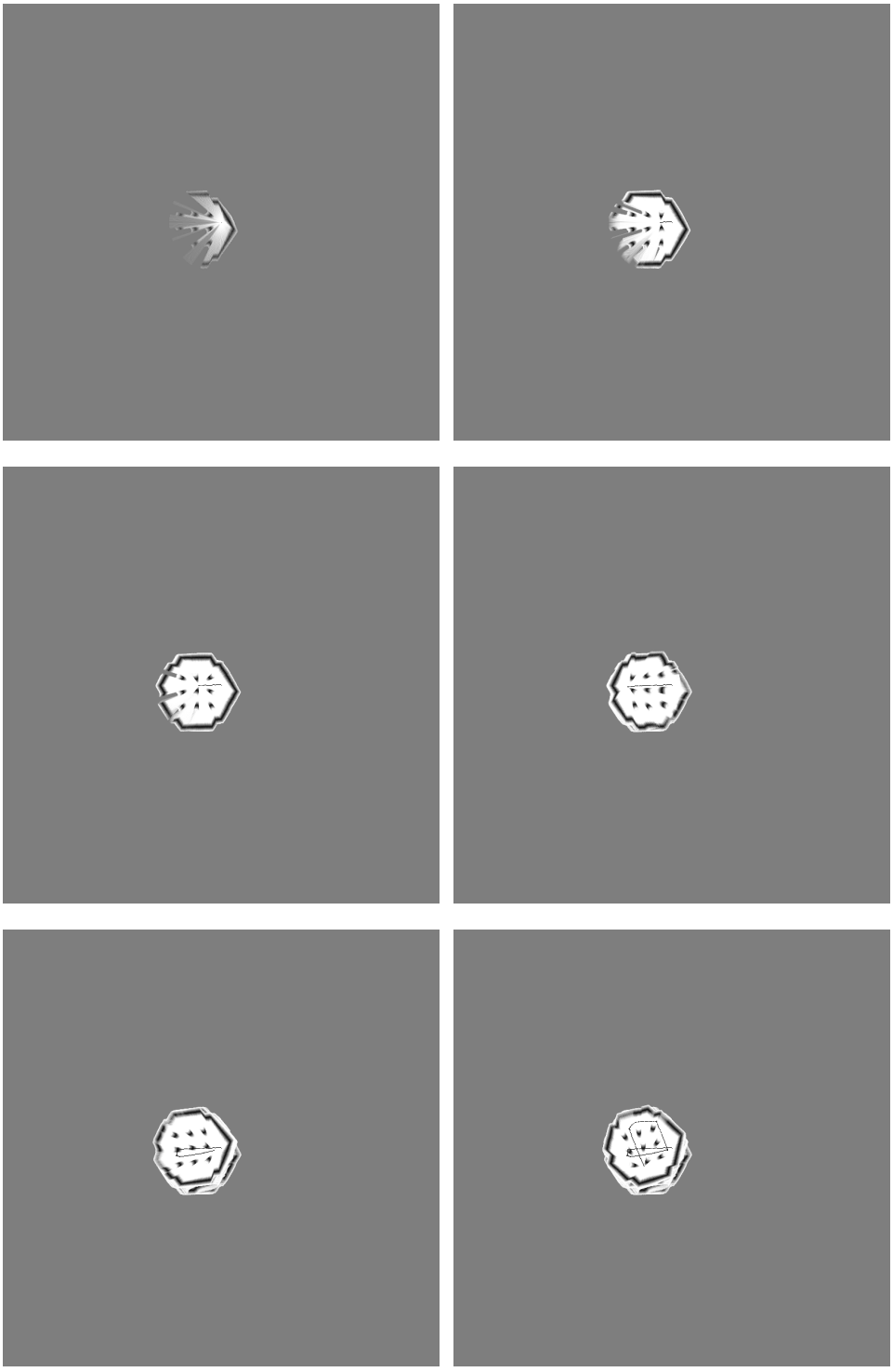

Figure 14: Steps of map generation from laser data of Gazebo environment

4.2. Improved Tiny SLAM (Low variance for position search)

Once the particle filter wrapper for Tiny SLAM was ready, we re conducted the above experiment using the same TurtleBot3 data. The variance parameters for running the Tiny SLAM were given low values of position variance of 45 and orientation variance of 2. Because of

these low variances the observed map was found to be quite inaccurate.

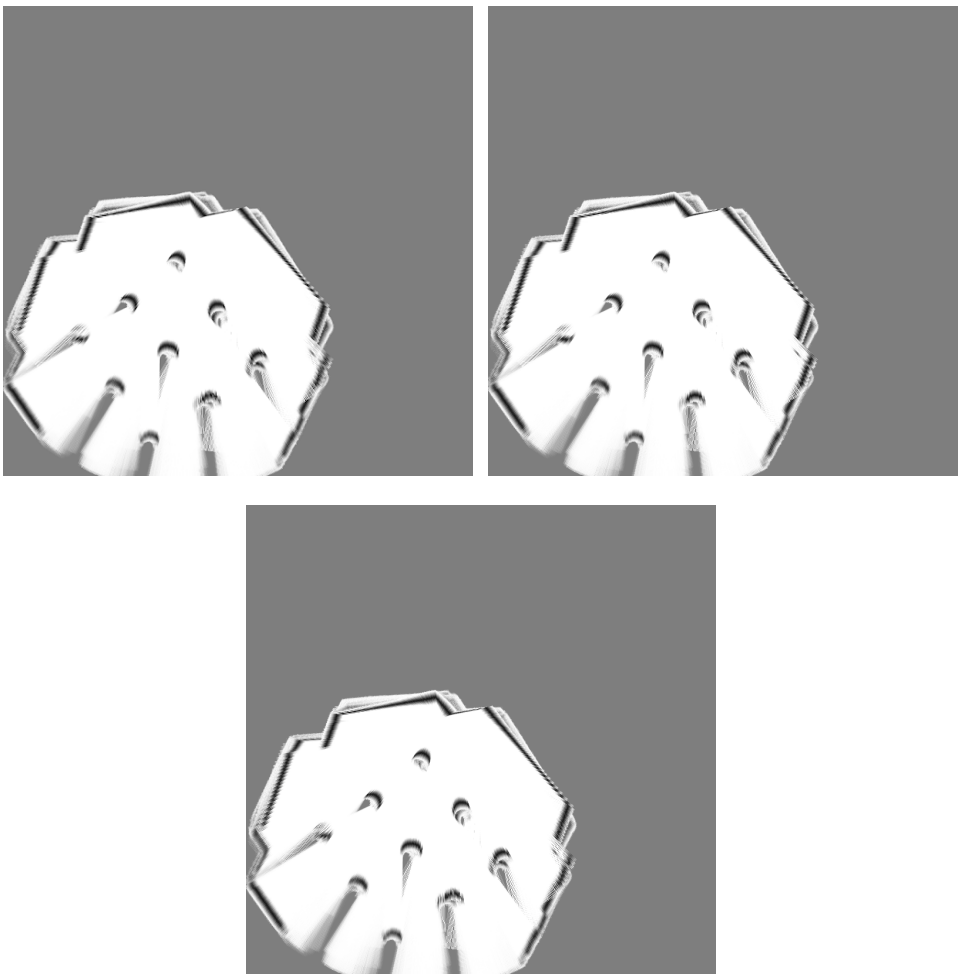

Figure 15: Maps generated by Improved Low Variance Tiny SLAM

4.3. Improved Tiny SLAM (High variance for position search)

We realized that when the variance parameter was increased, the quality of map increased. The low value of variance of the robot position that was given when searching for the best estimated position of the particle corresponded to the inaccurate maps. This value of the orientation variance was increased to 20 with the result of increased accuracy. The resulting maps were much better in terms of accuracy.

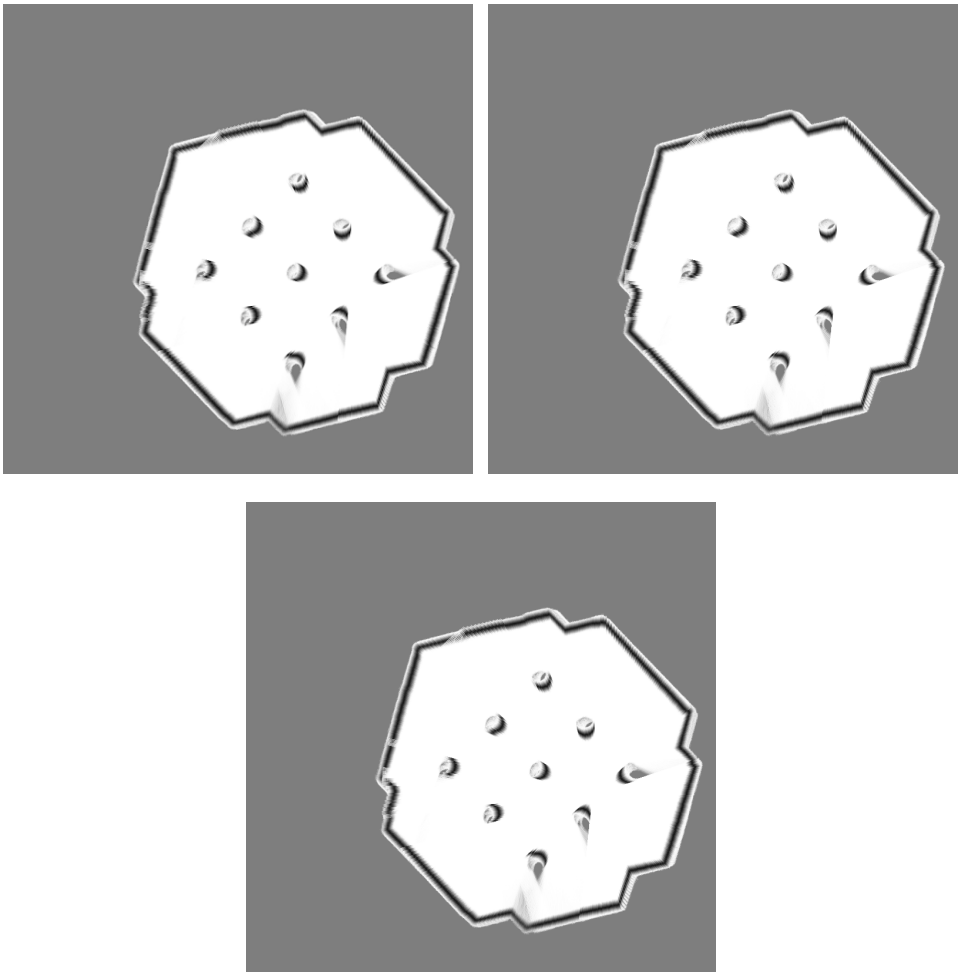

Figure 16: Maps generated by Improved High Variance Tiny SLAM

4.4. Real-Time Improved Tiny SLAM

After successfully finding the appropriate parameters, a platform for visualization of the position estimate and map estimate in ROS was completed. Then the whole system was run with a real-time visualization of the updated map, where we were able to directly read data from the Gazebo environment and use that data for performing SLAM and visualize the immediate output in Rviz environment. The resulting observations are as shown below:

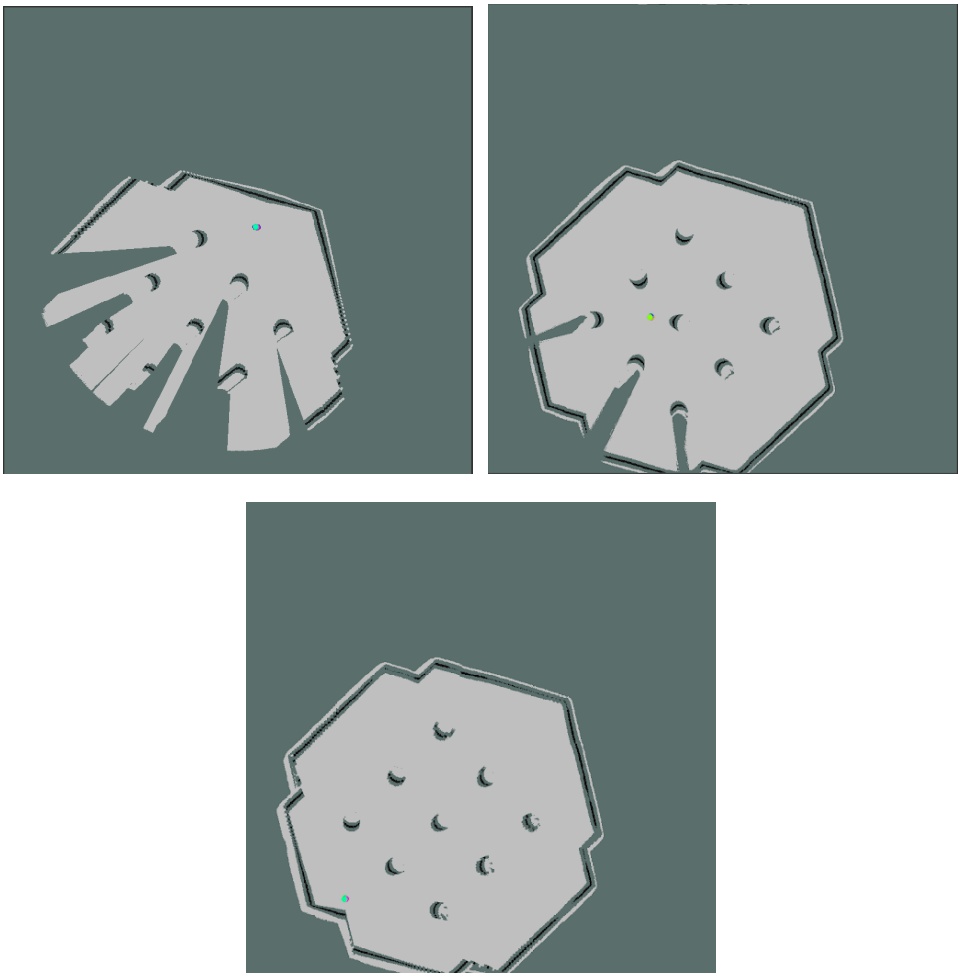

Figure 17: Maps generated by Real-time Improved Tiny SLAM

5. CONCLUSION

In this report, we presented an improved form of Tiny SLAM by adding a wrapper of particle filter above it to increase the quality of map produced. The increase in quality however came with a cost of increased computational complexity. We experimented with various numbers of particles to find the sweet spot between the map quality and increased computation and found that around 10 particles match the criteria. This improved Tiny SLAM was compiled as a ROS package for the ease of implementation and visualization of output. A method for experimentation and visualization of the output was also devised.

The quality of the produced map also depends on the variance parameters of the random position selector for the Random Hill Climbing Algorithm for the best position approximation. These variance parameters, namely: xy variance (σ_{xy}) and theta variance (σ_θ), were tweaked and fine tuned. Initially σ_{xy} and σ_θ was kept low with value of 15 and 2 respectively. This resulted in maps being disoriented with each iteration and looking like layered after rotating by a certain angle. As the variance was increased, the map quality also increased finally settling to a value of $\sigma_\theta = 20$. The role of variance is that it gives the width of probable random values for the next best position approximate. In case of low variances, this width is reduced and if the robot motion and change in orientation is greater than 3 times the variance then it is highly unlikely that the next random value chosen will be near the new actual position and orientation, even after a 1000 iterations. Thus increasing the variance increases the map quality as the probability of the next best position approximate being correct increases. Thus, the application of the particle filter over Tiny SLAM and setting its variance parameters $\sigma_{xy} = 45$ and $\sigma_\theta = 2$. increases the performance of the Tiny SLAM.

6. LIMITATION

There are certain limitations to this work. First of all, the robot motion in the experimental setup does not incorporate any sorts of control measures making it prone to the effect of inertia, especially when the speed of robot motion is high. As this introduces unwanted robot motion and unwanted sensor data and that also at a high speed, this affects the updated map introducing error in the map orientation. Also some of the grid cells beyond an occupied cell are allocated as unoccupied instead of unknown occupancy. The updated map shows a positional error whenever the robot hits hard on any obstacle due to the jerk introduced.

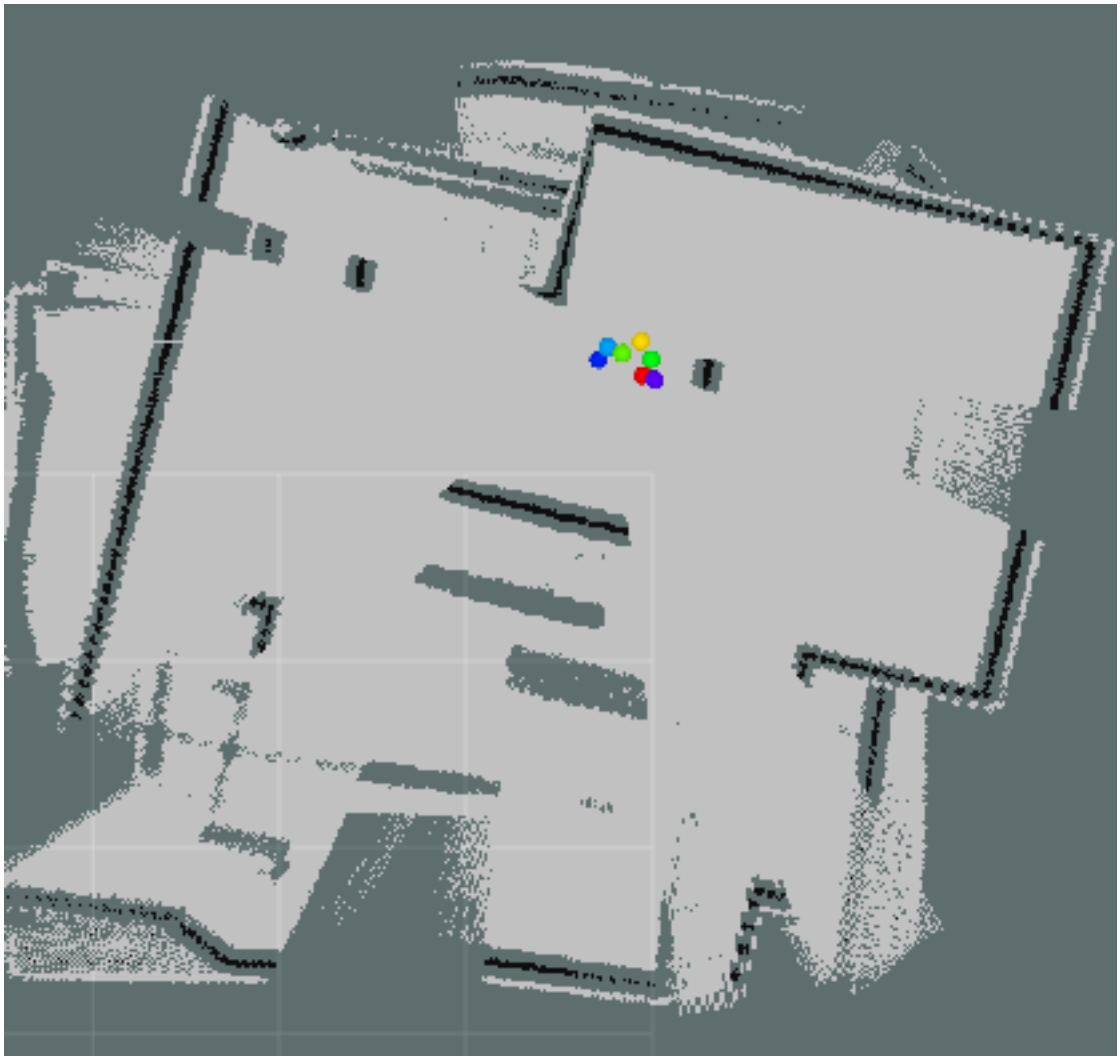

Figure 18: Distorted map due to sudden move

7. FUTURE ENHANCEMENTS

This project though complete in itself, has not been implemented in a physical hardware. As for the further development, this project can be implemented to a mobile robot in a physical environment addressing its issues. Next, the enhancement can be done by replacing the RMHC algorithm for best position approximation with a better optimization algorithm like particle filter or EKF. To increase the accuracy of laser scans, a better LIDAR like Hokuyo LIDARs, RP LIDAR etc.

REFERENCES

[1] J. J. Leonard and H. F. Durrant-Whyte, "Simultaneous map building and localization for an autonomous mobile robot.," in *IROS*, vol. 3, pp. 1442–1447, 1991.

[2] S. Huang and G. Dissanayake, "Robot localization: An introduction," *Wiley Encyclopedia of Electrical and Electronics Engineering*, pp. 1–10, 1999.

[3] D. F. Sebastian Thrun, Wolfram Burgard, *Probabilistic Robotics*. Intelligent Robotics and Autonomous Agents, 2005.

[4] C. Cadena, L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I. Reid, and J. J. Leonard, "Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age," *IEEE Transactions on Robotics*, vol. 32, p. 1309–1332, Dec 2016.

[5] F. Fraundorfer and D. Scaramuzza, "Visual odometry : Part ii: Matching, robustness, optimization, and applications," *IEEE Robotics Automation Magazine*, vol. 19, no. 2, pp. 78–90, 2012.

[6] S. Saeedi, M. Trentini, M. Seto, and H. Li, "Multiple-robot simultaneous localization and mapping: A review," *Journal of Field Robotics*, vol. 33, no. 1, pp. 3–46, 2016.

[7] S. Lowry, N. Sünderhauf, P. Newman, J. J. Leonard, D. Cox, P. Corke, and M. J. Milford, "Visual place recognition: A survey," *IEEE Transactions on Robotics*, vol. 32, no. 1, pp. 1–19, 2016.

[8] S. Thrun and M. Montemerlo, "The graph slam algorithm with applications to large-scale mapping of urban structures," *The International Journal of Robotics Research*, vol. 25, no. 5-6, pp. 403–429, 2006.

[9] B. Steux and O. E. Hamzaoui, "tinyslam: A slam algorithm in less than 200 lines c-language program," in *2010 11th International Conference on Control Automation Robotics Vision*, pp. 1975–1979, 2010.

[10] A. Huletski, D. Kartashov, and K. Krinkin, "Tinyslam improvements for indoor navigation," pp. 493–498, 09 2016.