SLAM-BASED MAPPING FOR OBJECT RECOGNITION

LOH WAN YING

A project report submitted in partial fulfilment of the requirements for the award of the degree of Bachelor of Engineering (Hons) Electronic Engineering

Faculty of Engineering and Green Technology
Universiti Tunku Abdul Rahman

May 2018

DECLARATION

I hereby declare that this project report is based on my original work except for citations and quotations which have been duly acknowledged. I also declare that it has not been previously and concurrently submitted for any other degree or award at UTAR or other institutions.

Signature	:	
Name	:	LOH WAN YING
ID No.	:	13AGB01857
Date	:	04 May 2018

APPROVAL FOR SUBMISSION

I certify that this project report entitled **“SLAM-BASED MAPPING FOR OBJECT RECOGNITION”** was prepared by **LOH WAN YING** has met the required standard for submission in partial fulfilment of the requirements for the award of Bachelor of Engineering (Hons) Electronic Engineering at Universiti Tunku Abdul Rahman.

Approved by,

Signature	:	
Supervisor	:	Dr. YAP VOOI VOON
Date	:	

Signature	:	
Co-Supervisor	:	Dr. HUMAIRA NISAR
Date	:	

Specially dedicated to

my beloved supervisor, co-supervisor, mother and father

ACKNOWLEDGEMENTS

I would like to thank everyone who had contributed to the successful completion of this project. First and foremost, I would like to express my gratitude to my research supervisor, Dr. Yap Vooi Voon for his invaluable advice, guidance and his enormous patience throughout the development of the research.

In addition, I would also like to express my gratitude to my research co-supervisor, Dr. Humaira Nisar for her guidance and care throughout the FYP. She has been motivating and encouraging by constantly sharing insightful comments and thoughts from time to time.

Furthermore, I would also like to express my gratitude to my loving parent and friends who had helped and given me encouragement.

SLAM-BASED MAPPING FOR OBJECT RECOGNITION

ABSTRACT

The aim of this project is to map an unknown environment, autonomously navigate to the 2D navigation goal set by user and recognize object placed in object database by using a custom made differential-drive mobile robot that works under the Robot Operating System (ROS) framework. The concept of deploying the robot in search and rescue mission, is being implemented so that the efficiency of search and rescue mission can be improved at a lower cost. The custom made robot is able to navigate in an unknown environment and feedback sensory data from Kinect Xbox 360 and odometry data to PC. Therefore, it is important for the robot to feedback a reliable and accurate odometry data efficiently so that the robot is able to localize itself in the unknown environment. The project architecture includes a personal laptop, a Kinect Xbox 360 sensor, the custom made robot and Arduino Mega 2560. The personal laptop acts as the command center where the Simultaneous Localization and Mapping (SLAM) algorithm are run by receiving odometry data from Arduino on the custom made robot. A USB connection is established between the Arduino, custom made robot and PC. After a map of the unknown environment is built, the Adaptive Monte Carlo Localization (AMCL) is used to localize the robot and Dijkstra’s algorithm is deployed to compute the shortest path to the destination goal. The SIFT (Scale-Invariant Feature Transform) is used to extract features from the current frame and match with the object database to identify and recognize the object whenever the robot come across the object. The location of object can also be obtained in respect to the location of Kinect sensor by using 3x3 Homography matrix. Implementation of project has been carried out successfully and the custom made robot is able to map and recognize object accurately.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS / ABBREVIATIONS

AMCL	Adaptive Monte Carlo Localization
BoW	Bag of Words
BRIEF	Binary Robust Independent Elementary Features
CCW	Counterclockwise
CW	Clockwise
DoG	Difference of Gaussians
EKF	Extended Kalman Filter
FAST	Feature from Accelerated Segment Test
GUI	Graphical User Interface
MCL	Monte Carlo Localization
ORB	Oriented FAST and Rotated BRIEF
PC	Personal computer
PCL	Point Cloud Library
PDF	Probability Density Function
PVC	Polyvinyl Chloride
RANSAC	Random Sample Consensus
RMSE	Root Mean Square Error
ROS	Robot Operating System
RPM	Revolution per minute
RTAB-Map	Real-Time Appearance-Based Mapping
RVIZ	ROS Visualization
SIFT	Scale Invariant Feature Transform
SIR	Sampling Importance Resampling
SLAM	Simultaneous Localization and Mapping

SURF	Speeded Up Robust Features
TF	Transformation
URDF	Unified Robot Description Format
USAR	Urban Search and Rescue
2D	Two dimensional
2WD	2 wheel drive
3D	Three dimensional
$A*$	A star algorithm
U	History of control inputs
X	History of vehicle locations
Z	Set of all landmark observations
m	Set of all landmarks
w_t	Importance weight of Rao-Blackwellized particles
x_k	Current state of robot
x_{k-1}	Previous state of robot
c_x	Principal point
$f(v)$	Sum of heuristic distance and length of path
fps	Frame per second
f_x	Focal length
$g(v)$	Length of path chosen
$h(x)$	Heuristic approximation
k_1	Radial distortion
L_{xx}	Convolution of input image with Gaussian differential operators
odom	Odometry
$\mathrm{P}(z\|x)$	Importance weight of the robot
p_1	Tangential distortion
qx	Quaternion

LIST OF APPENDICES

CHAPTER 1

INTRODUCTION

1.1 Project Overview

During disasters, it is critical to rescue the survivors who get trapped in the natural disaster such as earthquake, fire or flood, accident in manufacturing plant or manmade such as wars and terrorist attacks within the shortest period of time. According to the field of Urban Search and Rescue (USAR), there is a higher probability to rescue a victim within the golden 48 hours of rescue operation. Robots can be deployed in search rescue operating in recent years. Robots are also achieving a remarkable milestone in various fields which are manufacturing, education, medicine, military and industry.

A robot is a programmable mechanical device that is capable of performing tasks respectively. Therefore, robots can aid human in search and rescue activity. Robots are deployed to assist human tasks in order to reduce fatigue, improve efficiency, precision and quality of product as a robot can work for 24/7 (Khatib et al., 1999). Therefore, rescue robotics is one of the motivation in creating a truly autonomous system. Autonomous robot needs to have the capabilities of navigating around the dynamic environments, able to avoid obstacles, able to handle unpredictable situations and perform tasks and interact with its environment without the interference from human (Birk and Carpin, 2006).

Robots are widely used in the field of medicine such as telepresence, surgical assistants, medical transportation robots and even robotic prescription dispensing systems. Robots are also used to transport medicine and necessary aids among the patient and the medical team to reduce the burden of medical team. There is also application of robot which helps in increasing precision of surgery and even automated medicine dispensing systems as the biggest accuracy of robots are speed and accuracy.

From the above it is observed that robots are also expected to play an important role in search and rescue field. Rescue robots are equipped with hardware and able to communicate to computer to act as a life-saving tool. The objective of rescue robot is to cover a large area as fast as possible to provide information about the environment to the human rescue team. Robot can also be able to generate map of the environment and detect the victim to be saved. Rescue robots serve the purposes to enter those environment that is too small or too dangerous for human rescue team (Lafih and Meer, 2015).

1.2 Problem Statements

During the occurrence of disaster such as accident in manufacturing plant, it is difficult to rescue the human beings under the debris or rubble. However, in search and rescue activity, time is very critical as a large unknown area need to be covered by human rescue team within a short period of time. Detection of human in appropriate time is very important in such situations. In order to rescue victim within the first 48 hours of the rescue operation as fast as possible, a robot can be deployed to send the detected victims' location to a laptop PC as robot can operate 24/7. Then, these locations were marked and saved so that human rescue team can get to the location more accurately and thus saves time of searching the victim.

Furthermore, the robot designed is also meant to aid the search and rescue activity in an unknown environment. Therefore, the ability of robot to navigate and build a map in an unknown environment is very important. Rescue robots should also be able to sense the environments by using sensors to avoid obstacle. Performance of rescue robots should be stable enough to aid the search and rescue activity. The accuracy and stability of sensory data greatly affects the performance and accuracy of navigation of robot and also directly affect the path planning process. Location identification process also greatly depends on navigation process.

In order to identify the location of an object, the object must first be recognized by extracting features from the object. Once there is enough features extracted, the object can be detected and recognized, the object's position and orientation with respect to the pose of camera can be displayed in terms of image pixels.

1.3 Aims and Objectives

The objectives of the thesis are shown as following:

1) To implement SLAM on custom-made robot working under ROS framework in simulation and practical implementation.
2) To generate a 2D map about the unknown environment using SLAM-GMapping algorithm and 3D map using RGBD-SLAM algorithm.
3) To incorporate navigation stack into the robot for autonomous navigation and 2D navigation goal.
4) To identify, recognize and localize object without repeating and missing.

CHAPTER 2

LITERATURE REVIEW

2.1 Mobile Robot Navigation

Mobile robot navigation is a process of first acquiring the location and orientation of self-localization and then plan a path or route to allow the robot to reach its targeted destination by taking into account both sensor data and environment data. According to Leonard and Durrant-Whyte, the issue in handling navigation can be concluded into three questions which are "Where Am I?", "Where am I going?" and "How should I get there?" (Leonard and Durrant-Whyte, 1991). Robot navigation basically consists of three prime elements which are mapping, localization and path planning. Mapping is a process of generating a map based on sensory odometer data and environmental exploration. Localization is a process of identifying robot's own location based on the map build. Path planning is the process of navigating to the targeted location by using the shortest path distance between the robot's own location and the desired location (Pala et al., 2013). Robot need to run computations and calculations based on the sensory data collected to obtain the shortest path to the targeted location based on mapping and localization results as well.

2.1.1 Localization and Mapping

Navigation is a fundamental ability of a robot to localize itself based on robot orientation and location. The consistency and accuracy of sensor data play an important role to determine the best path to get to the targeted location (Tang, 2008). The past location of the robot and the future location it desired based on map making also determines the path calculated.

2.1.1.1 Dead Reckoning

Dead reckoning which is derived from “deduced reckoning” of sailing days is a simple mathematical procedure (Borenstein et al., 1997). Dead reckoning computes the present location by taking consideration of past locations and information from velocity and angular motion over a known period of time (Shufeldt, Dunlap and Bauer, 1999). Dead reckoning is the fundamental element of navigation skills (Borenstein et al., 1997).

Dead reckoning plays an important role in mobile robot navigation and will simplify the navigation process if accuracy of this method can be enhanced (Xu, Tan and Chen, 2002). Dead reckoning obtains orientation (direction), position linear and angular velocity of robot by computing basic trigonometry calculations. Dead reckoning is widely used in Autonomous Mobile Robot (AMR) due to the nature of simplicity and easy to debug (Park, Chung and Lee, 1998). Figure 2.1 shows the location of robot is computed by using dead reckoning.

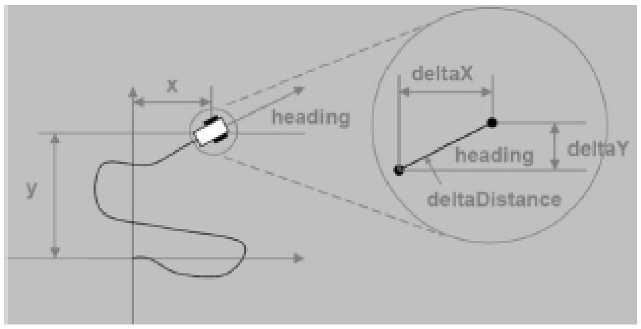


Figure 2.1: Dead Reckoning of the Robot (Zhenjun, Nisar and Malik, 2014)

Inertial sensors provide the robot with velocity and yaw angles which is measured by sensors integrated to the wheels of robot. Dead reckoning estimates the present location by using information relative to robot's starting points. Dead reckoning does not rely on external signals. Therefore, dead reckoning is known to have the features of cheap, simple and fast estimating time in current position of the robot (Cho et al., 2011). Although dead reckoning is the backbone of robot localization system but the accuracy is not obviously improved to obtain an accurate and reliable location estimation over a long period of time (Lee et al., 2008). Besides easy implementation and fast speed, dead reckoning results in accumulated errors unless error correction algorithm is integrated to eliminate any accumulated errors (Varveropoulos, 2005).

Odometry data alone is insufficient to provide a consistent and accurate position localized by the robot in using dead reckoning method (Von Der Hardt, Wolf and Husson, 1996). Accumulation of small errors due to robot's slippage in linear or rotational acceleration might lead to serious error in the process of self-localization of robot (Kanayama et al., 1990). Since dead reckoning is based on odometry data from the wheels, wheel slippage or mechanical rubber deformation and terrain roughness may lead to inability of the robot to keep track of its belief of its position over a long distance in a considerable amount of time (Tsai, 1998). The distance travelled is proportional to the accumulated error in estimating position of robot. As the distance travelled increases, the percentage error in position of robot also increases. Since the area of environment of experiments is expected to be large, dead reckoning is clearly not a suitable implementation to be used.

2.1.1.2 Simultaneous Localization and Mapping (SLAM)

SLAM is derived from a question raised from the robotics community which is whether it is possible to perform self-localization when placing a mobile robot in an unknown environment by building a reliable map of the environment. Therefore, a SLAM problem has been known as a "holy grail" for making the dream of fully autonomous mobile robot to come true (Bailey and Durrant-Whyte, 2006). An autonomous robot is known as a mobile robot with the capability of react to responses and perform designated specific task by itself without intervention from human or user. Autonomous robot can also be known as an artificial intelligence robot which is capable of “thinking” and “acting” based on the results of computations and decision making (Hadjia et al., 2015).

SLAM technique is meant to solve the problem of employing a mobile robot to construct a map of an unknown environment and thus enabling it to navigate the environment based on the map constructed (Riisgaard and Blas, 2004). Figure 2.8 shows the map constructed by mobile robot through repeating observations of environment and landmark in an unknown environment. Mobile robot does not need to have a prior knowledge about the location of itself and environment. Both the process of building map and computing robot’s location should be done in real-time implementation (Bailey and Durrant-Whyte, 2006). Optimization of performance of mobile robot in terms of landmark extraction and estimation, robot previous and current estimation of position, efficient path planning and reduction of localization error are the objectives of introducing SLAM algorithm (Leonard and Durrant-Whyte, 1991). SLAM algorithm is built up with multiple sections which are extracting and updating of landmark, associating of data, estimating and updating of state (Riisgaard and Blas, 2004).

An autonomous vehicle or mobile robot is equipped with a set of sensors such as accelerometer and gyrometer which are capable of measuring the rotation of wheels and camera to act as a vision sensor which is capable of extracting landmarks from the environment relative to the vehicle. The landmarks may be static or dynamic (Dissanayake et al., 2001). A group of landmarks is known as priori in applications of robotics (Cadena et al., 2016). The mobile robot is placed at a starting point in an unknown environment with no knowledge about the position of landmarks relative to the mobile robot. Observations of location of landmarks have been recorded and computed as the mobile robot roams around to compute the accurate position of the robot (Dissanayake et al., 2001). However, the presence of dynamic landmarks can lead to inaccuracy of building of map and error in SLAM algorithms. This is a matter that cannot be underestimated as most mobile robot applications are meant to work in a non-static environment (Wolf and Sukhatme, 2005).

Extended Kalman Filter (EKF) is the main component in SLAM algorithm (Riisgaard and Blas, 2004). Corners and edges of wall are also been considered as landmarks to improve the accuracy of SLAM method. EKF is used to estimate the accurate position of the robot and landmarks present in the unknown environment (Wolf and Sukhatme, 2005). The overview of SLAM process integrated with Extended Kalman Filter (EKF) is shown in Figure 2.2. Figure 2.3 to Figure 2.7 show the relationship between robot, sensor and odometry data and the compensation between the calculation of sensor and odometry data.

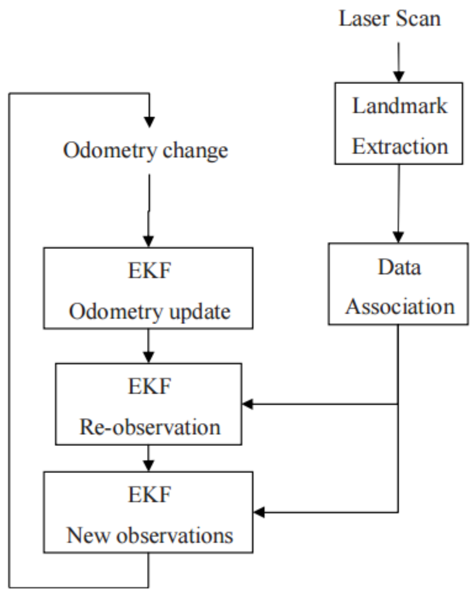


Figure 2.2: An overview of SLAM process integrated with Extended Kalman Filter, EKF (Riisgaard and Blas, 2004)

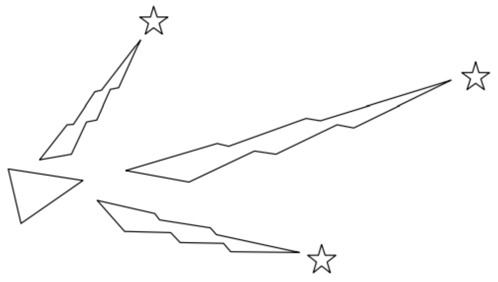


Figure 2.3: The triangle is a representation of robot. The stars are representation of landmarks. The lightning are representation of location of landmarks based on measurement of sensors (Riisgaard and Blas, 2004).

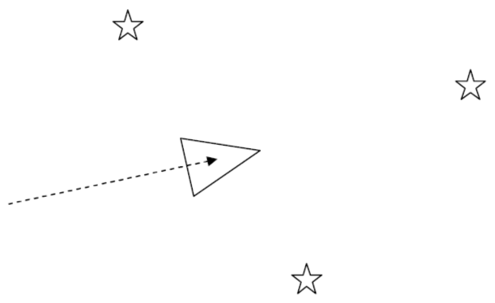


Figure 2.4: The robot estimates its current position and odometry provides distance travelled by robot (Riisgaard and Blas, 2004).

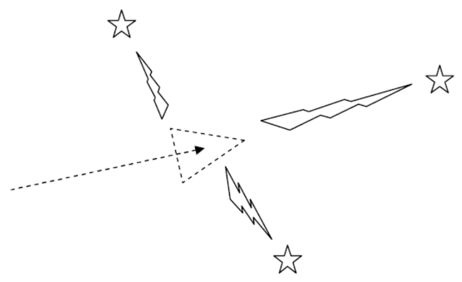


Figure 2.5: Sensors are used to measure the location of landmark relative to position of robot but it does not match with the location provided odometry data. Thus, the robot is not located at where it thinks it is (Riisgaard and Blas, 2004).

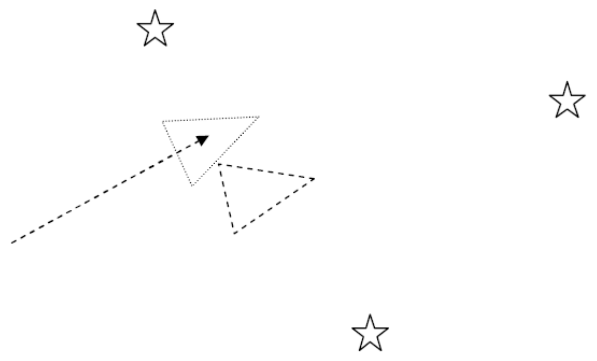


Figure 2.6: Generally, the robot relies more on sensors than its odometry. Location of landmarks are used to determine the current position of the robot. Dashed triangle represents the position of robot originally it thought it was (Riisgaard and Blas, 2004).

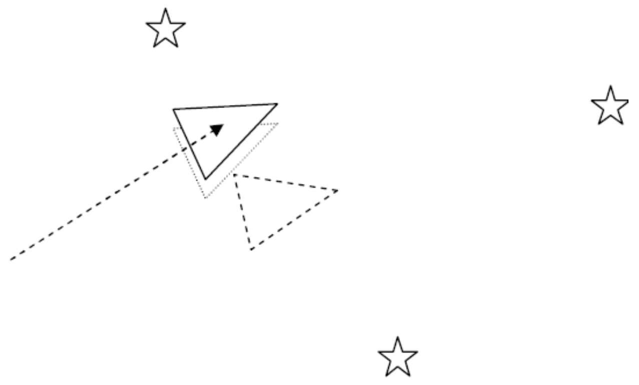


Figure 2.7: Straight line triangle represents the actual location of robot. Inaccuracy of sensors leads to inability of robot to know its precise location. However, the estimation is better when consider both odometry and sensor data. The dotted triangle is the representation of robot where it think it is. The dashed triangle tells where it was and last straight line triangle tells where it actually is (Riisgaard and Blas, 2004).

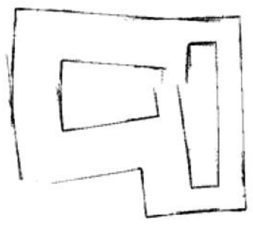


Figure 2.8: Map constructed by mobile robot through repeating observations of environment and landmark in an unknown environment (Riisgaard and Blas, 2004).

SLAM problem is solved by using a probability distribution which describes the past landmark locations and vehicle locations at certain time by given the control inputs and observations of vehicle towards landmarks. The probability distribution is shown below (Bailey and Durrant-Whyte, 2006):

$$P(\boldsymbol{x}_k, \boldsymbol{m} | \boldsymbol{Z}_{0:k}, \boldsymbol{U}_{0:k}, \boldsymbol{x}_0) \tag{2.01}$$

Given that:

$$\boldsymbol{X}_{0:k} = \{x_0, x_1, \dots, x_k\} = \{X_{0:k-1}, x_k\} \tag{2.02}$$

$$\boldsymbol{U}_{0:k} = \{u_1, u_2, \dots, u_k\} = \{U_{0:k-1}, u_k\} \tag{2.03}$$

$$\boldsymbol{m} = \{m_1, m_2, \dots, m_n\} \tag{2.04}$$

$$\boldsymbol{Z}_{0:k} = \{z_1, z_2, \dots, z_k\} = \{Z_{0:k-1}, z_k\} \tag{2.05}$$

where

$\boldsymbol{X}_{0:k}$ = the history of vehicle locations

$\boldsymbol{U}_{0:k}$ = the history of control inputs

$\boldsymbol{m}$ **=** the set of all landmarks

$\boldsymbol{Z}_{0:k}$ = the set of all landmark observations

By using the observation model, the probability of obtaining the landmark observations, $\boldsymbol{z}_k$ given the location of landmark, $\boldsymbol{m}$ and location of robot, $\boldsymbol{x}_k$. Assumption is made when the location of landmark and robot are obtained, the landmark observation is conditionally independent of map of surrounding unknown environment and the current position of robot. The observation model used in SLAM algorithm is shown below (Bailey and Durrant-Whyte, 2006):

$$\boldsymbol{P}(\boldsymbol{z}_k|\boldsymbol{x}_k, \boldsymbol{m}) \tag{2.06}$$

where
$\boldsymbol{z}_k$ = landmark observations
$\boldsymbol{x}_k$ = location of robot
$\boldsymbol{m}$ = location of landmark / map

By using the motion model, the probability distribution of state of robot can be determined. Markov process is used to predict the state transition of robot where the current position $\boldsymbol{x}_k$ depends on the previous position of robot $\boldsymbol{x}_{k-1}$ whereas the input to control the movement of robot is conditionally independent of both the landmark observations and the map of surrounding environment. The motion model used in SLAM algorithm is shown below (Bailey and Durrant-Whyte, 2006):

$$\boldsymbol{P}(\boldsymbol{x}_k|\boldsymbol{x}_{k-1}, \boldsymbol{u}_k) \tag{2.07}$$

where
$\boldsymbol{x}_k$ = current state of robot
$\boldsymbol{x}_{k-1}$ = previous state of robot
$\boldsymbol{u}_k$ = control input of robot

A major obstacle in overcoming the SLAM problem is associating data between previous observations of landmark and current observations of landmark. Inaccuracy in associating data may lead to serious failure of the SLAM algorithm (Bailey and Durrant-Whyte, 2006). Therefore, EKF which is a probabilistic method is used to limit the effect of inaccurate reading of sensor and the accuracy of map constructed by mobile robot. This is known as EKF-SLAM (Naminski, 2013). FastSLAM is introduced by integrating both Particle Filter and Extended Kalman Filter which leads to higher data accuracy. FastSLAM employs modified particle filter to estimate the posterior along the path of robot by breaking down SLAM problem into problems of collecting landmark estimation and problem of robot self-localization (Montemerlo et al., 2002). Particle filters were also used for multi-robot SLAM. Without prior knowledge of initial poses of robots, multi-robot SLAM is able to combine all data from all robots to construct a single map. When a Robot 1 encounter with another Robot 2, they measure their relative location and fed the measurements into a filter and then combines into a common map (Howard, 2006).

Despite of the high accuracy of SLAM algorithm, the computations are complex and intensive as robot constructs the map by every move of robot along the map (Zhang and Martin, 2013). Even though computational complexity can be deal with an advanced algorithm, there are still limitations to SLAM applications such as limitation to constructing map in outdoor environments (Thrun et al., 2004) and limitation to specific conditions and environments (Cheein et al., 2010). However, the performance of SLAM algorithm is good for the mobile robot to obtain information of the unknown environment to navigate and localize in unknown environment (Dissanayake et al., 2011).

2.1.1.3 Monte Carlo Localization (MCL)

Monte Carlo localization (MCL) is a combination of Kalman filter and particle filter algorithm (Chen et al., 2011). In order to navigate in a known indoor environment, a mobile robot must has knowledge about its position on the map. Monte Carlo localization is used to estimate the position of robot and orientation by employing particle filter (Naveed and Ko, 2014). MCL which is a probabilistic approaches is known as one of the reliable solution to provide real-time estimation of position in localizing robot (Dellaert et al., 2000).

There are two phases in computing the global position of mobile robot by using MCL which are prediction phase and update phase. Global position can be used to navigate and planning of path in a complicated known environment. Prediction phase is the first phase in which a motion model is used in predicting the current position of robot by using predictive Probability Density Function (PDF) in Bayes filter. Update phase is the second phase in which a measurement model is used to obtain readings from sensors and then compute to obtain a posterior PDF (Dellaert et al., 2000).

Fundamental idea of MCL algorithm is to collect a group of samples which is also known as particles. The samples represents the possible location of robot currently located in the known environment. Firstly, the samples are evenly distributed over a few possible location where the robot might be and every sample is given the same importance of weight. However, as the robot moves and times passes, those samples which are nearer to the current exact location will have more weightage than those who is further than the exact location (Fox et al., 1999). Figure 2.9 shows the distribution of the importance weights of particles p(z|x) assigned when a door need to be sensed in the environment (Zhenjun, Nisar and Malik, 2014).

The overview of MCL is as follows:

1. A set of samples is initialized by evenly distributing it over the possible locations with the same importance weightage.
2. The process is repeated until a spike of importance weight of samples is obtained:
   1. The robot is moved over a constant distance and readings from sensors have been taken.
   2. Movement model is used to update the distribution of each samples
   3. Sensor model is used to reassign the importance weightage of each sample based on their likelihood of new locations by using sensor readings.
   4. A new set of samples is created based on the updated importance weightage of each sample.
   5. This new set of samples has been assigned to be the current set of samples.

The accuracy of the computation results of MCL can be improved by increasing the total number of samples. However, there is a tradeoff between computational accuracy and efficiency as larger number of particles leads to lower computation efficiency in real-time applications (Naveed and Ko, 2014). Particle filters are easy to implement and it can be fused with various types of sensors, motion dynamics and it focus on the areas which has highest likelihood, therefore MCL has been used to solve many localization problems (Thrun et al., 2001). However, the performance and accuracy of MCL algorithm relies greatly on the symmetry of map and different configurations of maps might affect the result and number of iterations need to be computed to get the exact location of robot. MCL requires a relatively large amount of iterations to get accurate results and measurements from the same orientation. However, there is one problem faced in using MCL algorithm in real world, that is, the presence of noise will leads to uncertainties in the map built (Lee and Buitrago, 2015).

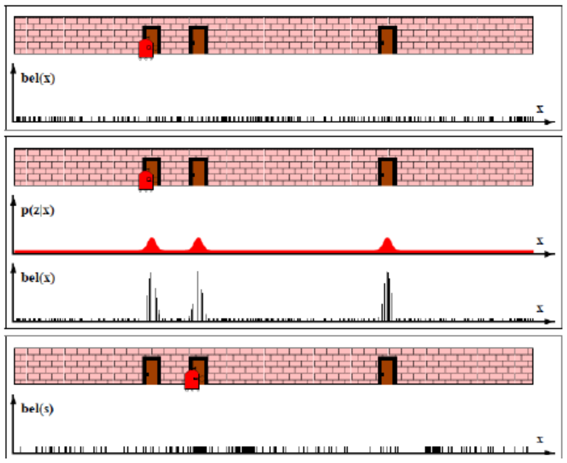


Figure 2.9: Distribution of the importance weights of particles p(z|x) assigned when a door need to be sensed in the environment (Zhenjun, Nisar and Malik, 2014).

2.1.2 Path Planning

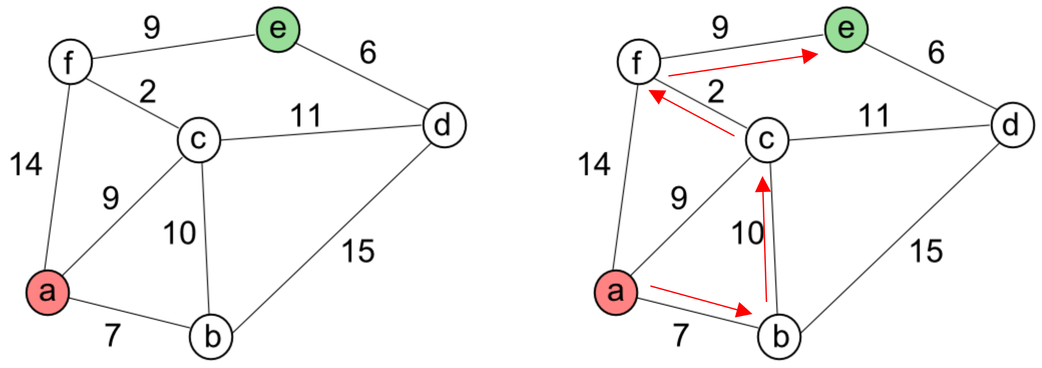
In the field of autonomous robotics, path planning is an important element in order to enable a robot to move from one point to another targeted destination by using the shortest path. By using the shortest path, many undesirable turning and braking can be avoided and this leads to less computations time and lower cost.

Path planning also helps to determine a path to avoid obstacle from a starting point to a targeted goal which in turns also enhances the performance of autonomous navigation in terms of time consumed, energy consumed and distance travelled (Raja and Pugazhenthi, 2012). In order to navigate optimally in an environment, the robot must be able to avoid obstacle and have a precise information about the map. Path planning involves a series of decision sequence (Duchoň et al., 2014). By dividing the map into nodes that are connected by edges, the shortest path would enable the robot to move from node to node as shown in Figure 2.7.

2.1.2.1 Dijkstra's Shortest Path Algorithm

Dijkstra's shortest path algorithm works on a basis of repeatedly computing the shortest distance from one source node to another vertices node and computes the nearest vertices node from the source node. For Dijkstra's algorithm to work correctly, the edges of the directed-weighted graph must not be negative.

Initially, the source node is chosen and the distance to source node itself is zero. Next, the distances to all other vertices are set to infinity to indicate that these vertices have not been processed yet. Then, the distance to nearest adjacent nodes are computed and the shortest distance path will be chosen. At this stage, the adjacent node will become the source nodes and the computation will be repeated until there is no outgoing edges from the vertices anymore. When there are no more vertices, the algorithm will be terminated. After terminating the algorithm, a shortest distance from one source node to another vertex is obtained (Abhishek et al., 2014). For example as shown in Figure 2.10, the node a is the source node. The adjacent nodes from the source node is node b, node c and node f. After computing the distance, the distance from node a to node b is the shortest path which is a weightage of 7. Next, node c now become the source node and the adjacent nodes are node f and node d. Node c to node f has the least weightage of 2. Then from node f, the targeted node has been reached which is node e. Since we have set node e as the targeted node, there should be no outgoing edges from node e. Thus, the Dijkstra's shortest path algorithm terminates and gives the shortest path from node a to node b to node c to node f and finally node e. Figure 2.10(b) shows the shortest path found.



(a) (b)

Figure 2.10: (a): The distance from source node to adjacent node is calculated and the shortest distance is chosen. (b): The shortest path is found from node a to b to c to f and finally to node e (Abhishek et al., 2014).

However, there is a disadvantage in Dijkstra's shortest path algorithm that it is computation intensive as it undergoes a blind search and it is time consuming and waste of resources (Abhishek et al., 2014).

2.1.2.2 A Star (A*) Algorithm

A* algorithm which is also known as A star algorithm is fundamentally the same as Dijkstra's but it also includes a heuristic approach. A* algorithm employs a heuristic approximation, *h(x)* which gives the estimation of the optimal route that goes through the starting point to the ending point (Abhishek et al., 2014). A star algorithm is also known as best first search approach as it visits the nodes by following the order of heuristic approximation and each cell in the map uses the equation below to compute their value:

$$f(v) = h(v) + g(v) \tag{2.08}$$

where

$h(v)$ = heuristic distance between the cell to the goal state

$g(v)$ = length of path chosen from the initial state to the desired goal state via the chosen sequence of cells

$f(v)$ = sum of heuristic distance and length of path chosen

The cell which has the lowest value of $f(v)$ will be the next sequence in the path. The reason A* algorithm has better advantage than Dijkstra's algorithm is because it starts with the favouring vertices that are near to the initial point and used Best First-Search method to find those favouring vertices that are near to the target point (Duchoň et al., 2014). Figures below illustrate more about the process of A* algorithm.

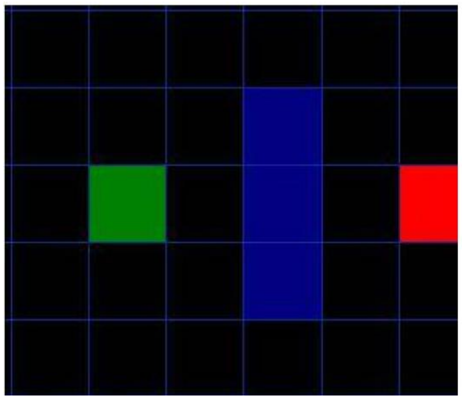


Figure 2.11: Assume green square represents the starting point and red point represents the goal point and the blue squares represent the obstacles that separates the two points (Abishek et al., 2014).

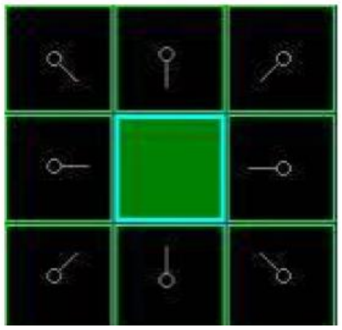


Figure 2.12: Search of 8-neighbouring nodes from the starting point (Abhishek et al., 2014).

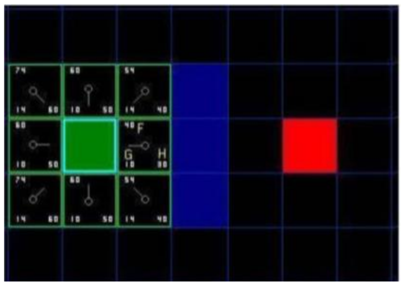


Figure 2.13: The Heuristics of 8-adjacent neighbouring nodes have been calculated by following *f(n) = g(n) + h(n)* (Abhishek et al., 2014).

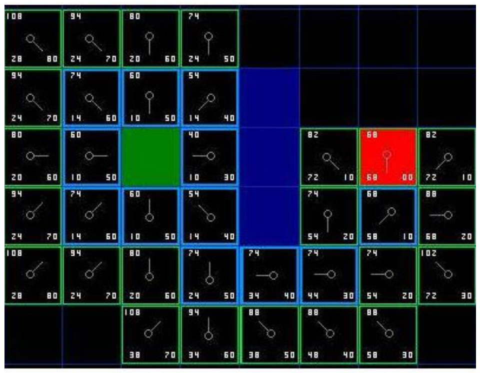


Figure 2.14: The nodes that are closer to the goal point from the starting point are chosen and shaded with blue border box. Then the nodes that have smallest value of *f(n)* will be the path chosen (Abhishek et al., 2014).

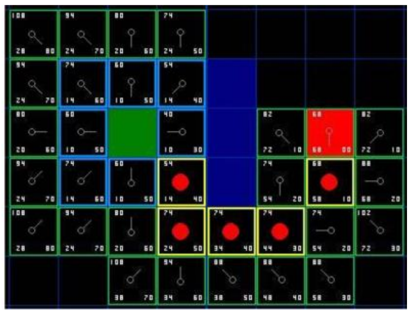


Figure 2.15: The red dots represent the path chosen with the smallest value of *f(n)*. This shows the shortest path computed by using A* algorithm (Abhishek et al., 2014).

2.2 GMapping Algorithm

Gmapping algorithm make use of Rao-Blackwellized particle filter to solve grid map SLAM problem by estimating the location of robot and landmark observations by using the map and past trajectory of robot. Another characteristics of Rao-Blackwellized particle filter in SLAM algorithm is the use of factorization where the path of robot is first estimated to compute the surrounding environment based on the path of robot. By estimating the path of robot, a map which relies greatly on the estimation of pose of robot can be generated in an efficient way. The factorization equation used in Rao-Blackwellized particle filter is shown below (Grisetti, Stachniss and Burgard, 2007):

$$P(x_{1:t}, m \mid z_{1:t}, u_{1:t-1}) = P(m \mid x_{1:t}, z_{1:t}) \cdot P(x_{1:t} \mid z_{1:t}, u_{1:t-1}) \qquad (2.09)$$

where

$x_{1:t}$ = trajectory of robot

$z_{1:t}$ = landmark observations

$u_{1:t-1}$ = odometry measurement

m = map

Particle filter used in Rao-Blackwellized is the sampling importance resampling (SIR) filter. There are four steps in building a map by using the sensor and odometry information and also Rao-

Blackwellized SIR filter which are sampling, importance weighting, resampling and map estimation. Sampling is the process to obtain the next set of particles with information from the previous set of particles. Motion model is used to sample the probability distribution by a factor of π stated in equation 2.10. Then each particle is assigned with an importance weight, w_t computed by using the equation below (Grisetti, Stachniss and Burgard, 2007):

$$w_t^{(i)} = \frac{P\left(x_{1:t}^{(i)} \,\middle|\, z_{1:t}, u_{1:t-1}\right)}{\pi\left(x_{1:t}^{(i)} \,\middle|\, z_{1:t}, u_{1:t-1}\right)} \tag{2.10}$$

where
w_t = importance weight
$x_{1:t}$ = trajectory of robot
$z_{1:t}$ = landmark observations
$u_{1:t-1}$ = odometry measurement

Then resampling is done by thresholding the number of particles for continuous distribution according to the importance weight of each particle. After resampling, all the particles will have the same value of importance weight. Finally, by taking into account both the past history of landmark observation and path of robot, an estimation of the map of surrounding environment can be generated (Grisetti, Stachniss and Burgard, 2007). Scan matching algorithm which is the matching of the previous laser scan input and current laser scan input is used to obtain the landmark observation. The laser scan input data is used in estimation of the pose of robot (Balasuriya et al., 2016).

2.3 RGB-D Simultaneous Localization and Mapping (SLAM)

In robotics application, it is important for a robot to know its location in respect with the world so that the robot can navigate around the world. In order to achieve this objective, the 3D models of the surrounding environment and the localization of pose of camera need to be estimated in parallel. The input datas will be from Kinect camera as Kinect is able to provide both depth images and colour images at 30 frame per second.

There are four stages in RGB-D SLAM which are features extraction, features matching, transformation estimation and lastly Octomap generation. Features extraction is done on the input colour images, then these features are matched with the previous images. The position of feature points on the depth images are used to compute the transformations between any two frames by using RANSAC (Random Sample Consensus). Finally, an Octomap library is used to generate a voxel occupancy map of the environment. The trajectory of robot can also be estimated by estimating the trajectory of pose of camera mounted on the robot (Endres et al., 2012). Trajectory estimated is divided into SLAM front-end and back-end as shown in Figure 2.16.

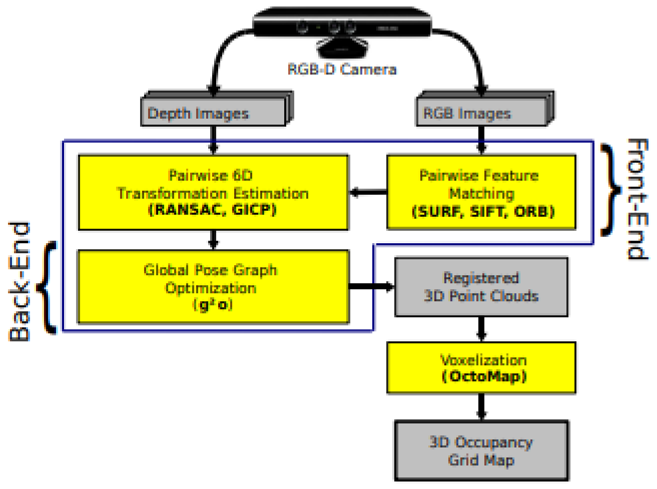

RGB-D Camera
Depth Images
RGB Images
Pairwise 6D Transformation Estimation (RANSAC, GICP)
Pairwise Feature Matching (SURF, SIFT, ORB)
Front-End
Back-End
Global Pose Graph Optimization (g²o)
Registered 3D Point Clouds
Voxelization (OctoMap)
3D Occupancy Grid Map

Figure 2.16: Schematic Overview of RGB-D SLAM (Endres et al., 2012)

In the front-end stage, keypoints are detected from the RGB images and descriptors are then extracted from the images by using various features descriptors such as SURF (Speeded Up Robust Features), SIFT (Scale-Invariant Feature Transform) and ORB (Oriented FAST and

Rotated BRIEF). The locations of features are then projected to 3D coordinates by using the depth measurements obtained from depth images of RGB-D camera. The transformation between current frame and previous frame is obtained through the transformation of pose of camera. Then RANSAC algorithm is used to eliminate the outliers and unstable data. This is done by eliminating those feature points that are less than the Euclidean distances. Those feature points that match with the pairwise Euclidean distances are consider as inliers and the inliers are used for computation of refined transformation of pose of camera. The transformation between camera poses is then used to form the edges of global pose graph (Endres et al., 2012).

In the back-end stage, the global pose graph is optimized by using g^2o framework which is a graph optimizer that is widely used in SLAM algorithm. By minimizing the non-linear error function presents in the pose graph, the pose graph can be optimized and loop closures can be formed. The non-linear error function is shown below (Endres et al., 2012):

$$F(x) = \sum_{<i,j>\in c} e\left(x_i, x_j, z_{ij}\right)^T \Omega_{ij} e(x_i, x_j, z_{ij}) \qquad (2.11)$$

$$x^* = argmin_x F(x) \qquad (2.12)$$

where

x = vector of pose representations

x_i = mean of poses

z_{ij} = information matrix

2.4 Robot Operating System (ROS)

Robot Operating System (ROS) is a framework that provides tools and libraries and is widely used in robotics field. ROS is fully compatible to Linux (Ubuntu) distributions and ROS usually are matched with respective Ubuntu distributions such as ROS Kinetic is compatible for Ubuntu 16.04.4 LTS. ROS fully supported some of the popular robot nowadays which are Turtlebot, Pepper and Robonaut. The ROS is an open source and free to use framework for research purpose and the functionalities of ROS can be expanded by contributing packages to ROS by developers. ROS also supports various programming languages such as Python, C++ and Libs (Joseph, 2017).

Programs created on ROS are known as nodes where communication between nodes are through topics by defining subscribers and publishers connections within the nodes. Then the programs is launched together with the executables created by using a launch file and the command *roslaunch*. In order for all the nodes to communicate with each other, a master node named *roscore* is used. Without running *roscore* while executing ROS programs, the nodes will not be able to find each other to exchange messages. To start using ROS, a ROS package which encompasses all the programs (ROS nodes), executable files and CMake text files for compiling are created under a workspace (Tawil, 2017). The relationship between ROS master, nodes and topics is shown in Figure 2.17.

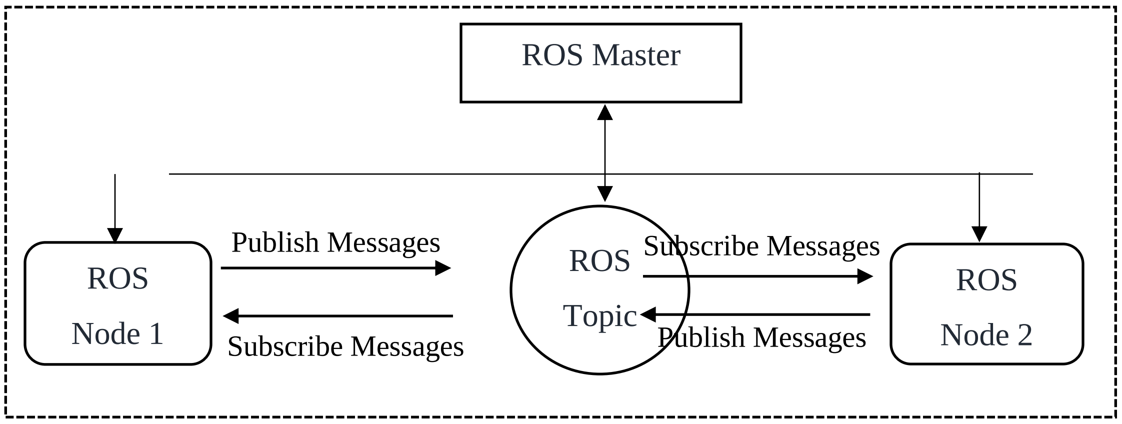


Figure 2.17: Relationship between ROS master, nodes and topics

2.5 Object Detection and Recognition

Object detection is important in the robotics field as during the process of mapping and executing tasks, a robot needs to have information about the surrounding environment and location of objects. The working principle of object recognition is to identify objects in the real world from input image of the world with prior information about the object models in object database. The ability of robot to recognize a known object which is placed in object database from different point of view will be helpful in assisting the search and rescue activity in locating the position of a victim or certain objects (Ekvall, Kragic and Jensfelt, 2007).

Features can be extract from the object to identify a match from the object database. There are usually two phase in object recognition process which are training phase and testing phase. Training phase is the stage where a set of interest points are selected by using feature descriptors algorithm whereas testing phase is the stage where the images in the current frame are compared to the database set by determining the matches of interest points (Bhosale Swapnali, Kayastha Vijay and Harpale Varsha, 2014). There are a few image processing algorithm that can be used such as SURF (Speeded Up Robust Features) and SIFT (Scale-Invariant Feature Transform) (Rublee et al., 2011).

2.5.1 SURF (Speeded Up Robust Features) Algorithm

SURF algorithm is a feature descriptor algorithm which focus on the number of feature pairs generated between the input image and image database. There are four steps in SURF algorithm which is shown in Figure 2.18. First step is interest point detection and then generate descriptor of the interest points in second step based on first and second order derivatives. Following step is to match the feature points descriptors are used to match with the input image where only inlier points within the object are considered. With interest point detection, local maxima of Fast-Hessian-like operator is used to determine the potential significant points which are located at the corners and junctions. The Hessian equation is shown as below (Matas and Mikolajczyk, 2012):

$$H(x,y) = \begin{bmatrix} L_{xx} & L_{xy} \\ L_{xy} & L_{yy} \end{bmatrix} \tag{2.13}$$

where

$L_{xx}(x,y,\sigma)$ = Convolution of input image with Gaussian second order differential operators

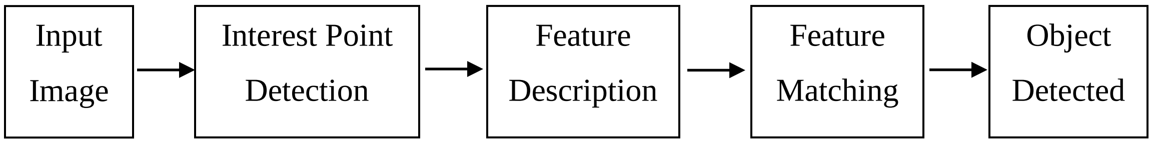


Figure 2.18: Flow of SURF Algorithm (Bhosale Swapnali, Kayastha Vijay and Harpale Varsha, 2014)

Then each of the keypoint in the neighbourhood is represented by distinctive feature descriptors which are invariant to orientation. This is done by calculating a set of pixels within a radius of 6σ in the neighbourhood by using the Haar wavelet in (x,y) directions. The σ stands for the scale of identified interest points. Haar wavelet filters is shown in Figure 2.19 where the dark side carries a weightage of -1 and the bright side carries a weightage of +1 (Mistry and Banerjee, 2017).

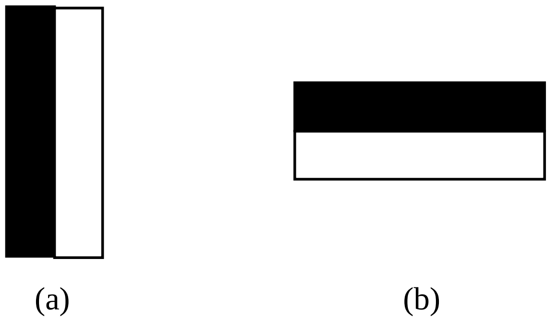


Figure 2.19: (a) Haar wavelet filters to compute responses in x direction
(b) Haar wavelet filters to compute responses in y direction

2.5.2 SIFT (Scale-Invariant Feature Transform) Algorithm

SIFT algorithm is an algorithm used to detect local features and generate descriptors of objects. SIFT algorithm is invariant to rotation, orientation and also changes in scales. SIFT algorithm is divided into four stages which are detection of extrema of scale-space by using Difference of Gaussians (DoG), localization of potential keypoints, computation of orientation of each keypoint and lastly extraction of descriptor of each keypoint. Difference of Gaussians (DoG) is used to determine the candidate interest points by computing the scale space extrema of the input images using the equations below (Hamid et al., 2012):

$$D(x,y,\sigma) = \left(G(x,y,k\sigma) - G(x,y,\sigma)\right) * I(x,y) \tag{2.14}$$

$$D(x,y,\sigma) = L(x,y,k\sigma) - L(x,y,\sigma) \tag{2.15}$$

where

$I(x,y)$ = Digital Image

$L(x,y,\sigma)$ = Scale-space Representation

$G(x,y,\sigma)$ = Variable-scale Gaussian kernel with standard deviation σ

Then the points with low contrast values and unstable edge responses are eliminated in the process of localization of keypoints. A threshold has been set by computing the ratio of eigenvalues of Hessian matrix. By using the threshold value, those interest points that has an unstable spatial value will be eliminated as high ratio of eigenvalues of Hessian matrix represents unstable corner interest points whereas low ratio represents stable corner interest points. Extrapolation across the DoG images were done to localise the remaining interest points. Then orientation of each keypoint are computed and assigned to respective interest point. In the final phase, feature descriptors for each keypoint are computed. An array of 4x4 histogram was created with eight orientation bins for each region as shown in Figure 2.20. Therefore, feature descriptors for SIFT algorithm is using a dimension of 128 (4 x 4 x 8 =128) (Khan, McCane and Wyvill, 2011).

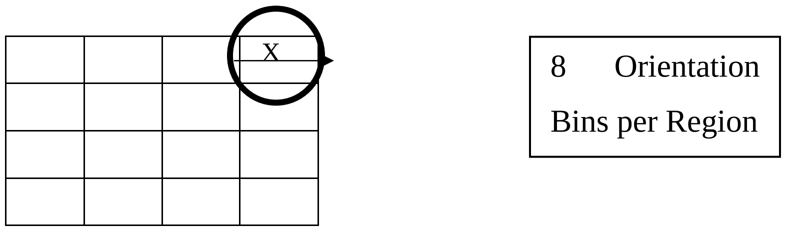


Figure 2.20: 4 x 4 Computed Orientation Histogram Arrays in 128 Dimension SIFT (Modified from Khan, McCane and Wyvill, 2011)

2.6 Summary for Robot Navigation (Localization and Mapping)

Table 2.1: Summary for Robot Navigation in Localization and Mapping

Author/Year	Data Collection Techniques	Advantages	Disadvantages	Accuracy	Algorithm Used
Park, Chung and Lee (1998)	Uses basic trigonometry operations and odometry datas.	Cheap, simple and fast process.	Accumulates error and short term accuracy.	Low	Dead Reckoning
Dissanayake et al. (2001)	Uses accelerometer, gyrometer, odometry data and EKF.	High accuracy and works in unknown environment.	Computational intensive and complicated algorithm.	Very high accuracy.	Simultaneous Localization and Mapping (SLAM)
Naveed and Ko (2014)	Uses odometry data, sensors and randomized particles.	Fast, reliable accuracy and less memory intensive.	Only works in unknown map.	Higher accuracy compared to dead reckoning.	Monte Carlo Localization (MCL)

2.7 Summary of Robot Navigation (Path Planning)

Table 2.2: Summary for Robot Navigation in Path Planning

Author/Year	Data Collection Techniques	Advantages	Disadvantages	Accuracy	Algorithm Used
Abhishek, Prateek, Rishabh and Neeti (2014)	Shortest path between nodes.	Fast, accurate and easy.	Slightly computational intensive.	Accurate	Dijkstra’s Algorithm
Abhishek, Prateek, Rishabh and Neeti (2014)	Shortest path between nodes by using Heuristic approach	Fast, accurate, and less computation than Dijkstra’s algorithm.	Does not mention.	Accurate	A* algorithm

2.8 Summary of Object Recognition

Table 2.3: Summary for Robot Navigation in Object Recognition

Author/Year	Data Collection Techniques	Advantages	Disadvantages	Accuracy	Algorithm Used
Mistry and Banerjee (2017)	Determine keypoints with Hessian matrix and non-maxima suppression.	Fast, invariant to blur.	Not stable to rotation and scale changes	Accurate	SURF Algorithm
Mistry and Banerjee (2017)	Use local extrema detection, non-maxima suppression and eliminate edge response with Hessian matrix.	Invariant to rotation, scale changes and blur.	Computational intensive and not good at illumination changes.	Accurate	SIFT algorithm

CHAPTER 3

METHODOLOGY

3.1 Design Specifications

In this project, a framework that uses a mobile robot to generate a map in unknown environment and which is used to localize the victim is designed. The project interfaces include the use of a personal computer (PC) to transmit commands and receive signals from Kinect Xbox 360 (images, depth information and pointcloud) and Arduino Mega 2560 Robot (sensory and odometry data) as shown in Figure 3.1. PC is the command center for mobile robot and Kinect module. Wired connection is set up between PC and mobile robot using a common A to B Male/Male type peripheral USB 2.0 cable is used to receive and transmit signal from Arduino.

The robot navigation process starts with mapping. The starting point is set arbitrarily and the ending point is the point when the SLAM Gmapping algorithm done loop closing thread. However, the movement of wheel’s rotations, distance covered by wheels and encoder odometry data are used in computing the location of the robot. Once the robot is localized, the robot will detect the depth of the object in the map built and will navigate to the destination set by user through the path planned by the PC.

Next, RGBD SLAM is also launched to build a 3D map of the environment and also detecting the object desired by the user. Object counting and localization is carried out after detecting the object that matches with the database. If robot recognized the landscape as object, the location of object will be computed after obtaining a 3x3 Homography Matrix from Kinect module.

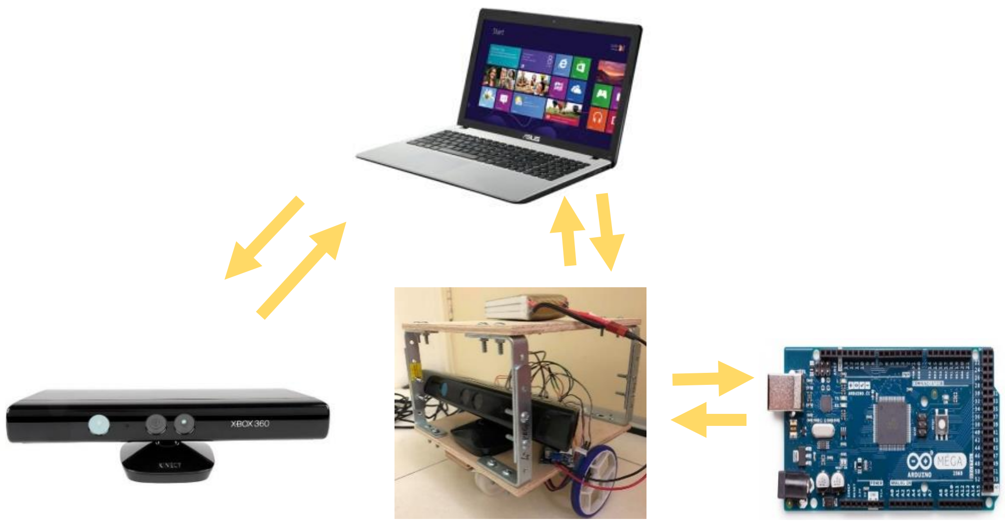


Figure 3.1: The interfaces between PC, mobile robot, Arduino Mega 2560 and Kinect Xbox 360

CHAPTER 4

RESULTS AND DISCUSSIONS

4.1 Preliminary Work

In this project, there are a few preliminary work was carried out to ensure that the project's objectives are met and verify the design methodology as stated in previous chapter.

4.1.1 Robot's Specification

As mentioned in Section 3.4, the robot is constructed using differential steering to turn right or left and also to move forward and backward. By setting the DC motor with encoder (left motor and right motor) to rotate at same speed but different direction, the direction of spinning of robot can be controlled. Table 4.1 below shows the relationship between the motor's setting and the movement of robot.

Table 4.1: Relationship between the motor's setting and the movement of the robot.

Movement of Robot	**Setting of Motor (CW = clockwise ; CCW = counter clockwise)**	
	Right Motor	**Left Motor**
Forward	CW	CW
Backward	CCW	CCW
Turn to Right	CCW	CW
Turn to Left	CW	CCW

In this project, the process of building of map in an unknown environment and navigation rely greatly on the robot's ability to navigate accurately. Therefore, robot's specification plays a very important role in determining the accuracy of the map build and also in navigating around without colliding into obstacles. Hence, a few experiments had to be carried out in order to calibrate the robot's specifications which involves the wheel diameter, wheel width and also track width. Determining the correct set of robot's specifications will ensure the robot is able to move and the wheels will rotate accurately when a command signal is sent from PC to the robot.

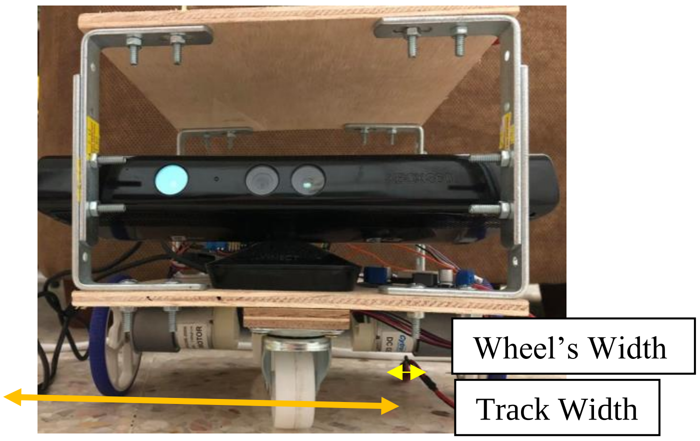


Figure 4.1: Track width and wheel's width of differential steering of robot

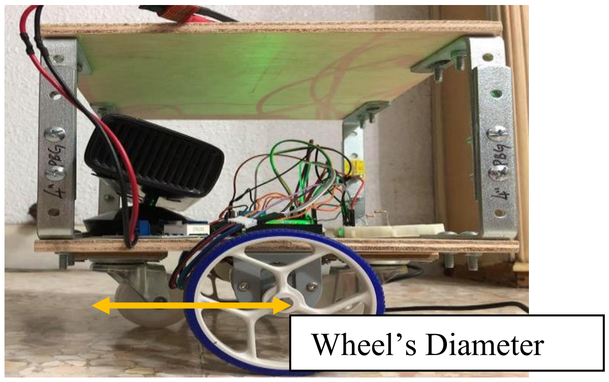


Figure 4.2: Wheel’s diameter of differential steering of robot

The calibrating experiment is carried out with two different sets of wheel diameter and track width. Wheel width will remain constant in both sets of experiment. The results are shown in Table 4.2 and Table 4.3. Table 4.2 demonstrates the case when wheel’s diameter and track width are set at 8 cm and 23.7 cm respectively. For Table 4.3, wheel’s diameter and track width are set at 8 cm and 24.7cm respectively.

Table 4.2: Total angle of rotation of robot when wheel’s diameter and track width are set at 8 cm and 23.7 cm respectively.

Angle of Rotation of Robot in PC (°)	**Actual Robot’s Rotation Made (°)**			**Average Actual Robot’s Rotation Made (°)**	**Error (°)**
	1	**2**	**3**		
10	9.0	11.4	11.5	10.7	0.7
30	32.6	32.9	30.6	32.0	2.0
60	58.4	58.8	64.0	60.4	0.4
90	90.8	88.4	90.1	89.8	0.2
150	156.8	154.7	153.0	154.8	4.8
180	172.8	178.5	180.3	177.2	2.8
270	260.0	268.3	267.7	265.3	4.7
300	299.6	297.2	298.6	298.5	1.5
330	337.2	334.6	334.8	335.3	5.3
360	367.0	356.0	361.0	361.3	1.3
Average Actual Robot’s Rotation Error (°)					**2.4**

Table 4.3: Total distance travelled of robot when wheel's diameter and track width are set at 8 cm and 23.7 cm respectively.

Distance Travelled by Robot in PC (cm)	**Actual Robot's Distance Travelled (cm)**			**Average Actual Robot's Distance Travelled (cm)**	**Error (cm)**
	1	**2**	**3**		
5	5.0	5.0	5.0	5.0	0.0
10	10.0	10.0	9.7	9.9	0.1
13	12.9	13.2	13.3	13.1	0.1
17	17.4	17.3	16.8	17.2	0.2
20	20.5	20.4	19.7	20.2	0.2
25	24.6	24.7	24.5	24.6	0.4
30	28.6	28.6	28.4	28.5	1.5
45	40.0	41.0	40.5	40.5	4.5
60	52.0	55.0	55.1	54.0	6.0
Average Actual Distance Travelled Error (cm)					**1.4**

Table 4.4: Total angle of rotation of robot when wheel's diameter and track width are set at 8 cm and 24.7 cm respectively.

Angle of Rotation of Robot in PC (°)	**Actual Robot's Rotation Made (°)**			**Average Actual Robot's Rotation Made (°)**	**Error (°)**
	1	**2**	**3**		
10	13.4	12.6	12.1	12.7	2.7
30	42.9	37.5	42.5	41.0	11.0
60	69.3	68.7	73.3	70.4	10.4
90	91.7	93.6	99.5	94.9	4.9
150	153.8	161.1	167.6	160.8	10.8
180	185.8	191.0	193.1	190.0	10.0
270	286.1	287.7	287.7	287.2	17.2
300	316.9	315.1	316.0	316.0	16.0
330	357.9	355.7	369.6	361.1	31.1
360	377.8	373.6	396.8	382.7	22.7
Average Actual Robot's Rotation Error (°)					**13.7**

Table 4.5: Total distance travelled of robot when wheel's diameter and track width are set at 8 cm and 24.7 cm respectively.

Distance Travelled by Robot in PC (cm)	Actual Robot's Distance Travelled (cm)			Average Actual Robot's Distance Travelled (cm)	Error (cm)
	1	2	3		
5	5.0	5.0	5.0	5.0	0.0
10	10.0	10.0	10.0	10.0	0.0
13	13.0	13.2	13.1	13.1	0.1
17	16.5	16.7	17.0	16.7	0.3
20	19.5	20.3	20.5	20.1	0.1
25	22.0	21.7	22.7	22.1	2.9
30	28.6	28.0	28.8	28.5	1.5
45	40.0	41.0	39.5	40.2	4.8
60	48.0	47.1	52.0	49.0	11.0
Average Actual Distance Travelled Error (cm)					**2.3**

From the results shown above, the wheel's diameter and track width should be set to 8 cm and 23.7 cm respectively in order to get a minimum average robot's rotation error of 2.4° from Table 4.2 and distance travelled error of 1.4 cm from Table 4.3. By using this set of robot's specification, the robot can move and rotate more accurately based on command given from PC.

The actual rotation angle of robot is measured by using a measurement lever from Beckhoff with the range from 0° to 225°. The angle of rotation of robot in PC is observed by using RVIZ (ROS visualization) tool which is a 3D visualizer that display the joint rotation and state information of *base_link* based on the virtual robot model. The Quaternions angle (x, y, z, w) was then converted into axis angle in radian. Both position and orientation of virtual robot model can be obtained from the RVIZ (Lehman, 2015). Two equations are used, Equation 4.1 is used to change Quaternion angle to axis angle in radian and Equation 4.2 is used to change the axis angle in radian to axis angle in degree.

$$angle_{(rad)} = 2cos^{-1}(w) \tag{4.1}$$

$$angle_{(degree)} = \frac{angle_{(radian)}}{\pi} \times 180° \tag{4.2}$$

4.1.2 Range Detection towards Flat Surface (Wall) by using Kinect Xbox 360 sensor

In this project, Kinect Xbox 360 sensor will be used for object detection in both map building process and also object recognition in Section 4.6. In order to build a map of an unknown environment, the robot should be able to detect the objects in the environment and the laser scan towards the object will be used as one of the input of SLAM-GMapping algorithm in Section 4.2. Therefore it is important to test the accuracy of Kinect Xbox 360 sensor towards flat surface such as wall of room. From the results shown below in Table 4.6, the average error is 3.7cm where the error increases when the distance of wall from the robot exceed 180cm. The minimum range of Kinect sensor towards an object is set at 45cm so any object that is placed at a range less than 45cm, the robot will not be able to detect the object.

Table 4.6: The relationship between the actual distance of the wall from the robot and the range measured by Kinect Xbox 360 sensor towards the wall from the robot (cm)

Actual Distance of Wall from the Robot (cm)	**Range Measured by Kinect Xbox 360 Sensor towards the wall from the robot (cm)**			**Average Range Measured by Kinect Xbox 360 Sensor towards the Wall from the Robot (cm)**	**Error (cm)**
	1	**2**	**3**		
40	-	-	-	-	-
45	45.0	46.5	47.0	46.2	1.2
50	50.5	50.6	49.5	50.2	0.2
55	55.5	55.0	56.0	55.3	0.3
60	61.0	61.5	59.5	60.7	0.7
70	69.0	71.0	71.0	70.3	0.3
90	91.0	89.0	90.0	90.0	0.0
100	100.5	99.0	100.0	99.8	0.2
120	120.0	119.0	119.0	119.3	0.7
150	149.0	145.0	146.0	146.7	3.3
180	175.0	174.0	173.0	174.0	6.0
220	211.0	208.0	207.0	208.7	11.3
250	230.0	230.0	231.0	230.3	19.7
Average Range Measured by Kinect Xbox 360 Sensor towards the Wall from the Robot Error (cm)					**3.7**

4.1.3 Range Detection towards Curved Surface (Bag) by using Kinect Xbox 360 Sensor

In an unknown environment, there might be presence of curved object such as chairs, bags and so on. Therefore, range detection towards curved surface must also be measured by using Kinect Xbox 360 sensor. From the results shown in Table 4.7, the range error is 6.7cm. This shows that the accuracy and reliability of Kinect sensor is reduced when measuring the range towards curved object due to irregular surface reflection (Manap et al., 2015).

Table 4.7: The relationship between the actual distance of the curved object from the robot and the range measured by Kinect Xbox 360 sensor towards the curved object from the robot (cm).

Actual Distance of Curved Object from the Robot (cm)	**Range Measured by Kinect Xbox 360 Sensor towards the Curved Object from the Robot (cm)**			**Average Range Measured by Kinect Xbox 360 Sensor towards the Curved Object from the Robot (cm)**	**Error (cm)**
	1	**2**	**3**		
45	47.0	48.0	48.0	47.7	2.7
50	48.0	49.0	50.0	49.0	1.0
55	54.0	53.5	54.3	53.9	1.1
60	59.0	60.5	59.5	59.7	0.3
70	65.0	66.0	65.7	65.2	4.8
90	79.5	80.0	80.3	79.9	10.1
100	103.0	102.0	102.5	102.5	2.5
120	119.0	116.4	117.0	117.5	2.5
150	147.5	146.0	144.0	145.8	4.2
180	174.5	176.0	175.5	175.3	4.7
220	204.0	204.6	207.0	205.2	14.8
250	218.7	218.0	219.1	218.6	31.4
Average Range Measured by Kinect Xbox 360 Sensor towards the Curved Object from the Robot Error (cm)					**6.7**

4.1.4 Range Detection towards Distance between Objects by using Kinect Xbox 360 sensor

In order for the robot to accurately navigate in an unknown environment with different type of objects, the robot must be able to estimate the distance between objects. This can help the robot in determining whether robot can go through the two objects or need to bypass the object.

Table 4.8: The relationship between the actual distance between objects and Kinect sensor reading for the distance between objects (cm).

Actual Distance between Objects (cm)	**Kinect Sensor Reading for the Distance between objects (cm)**			**Average Kinect Sensor Reading for the Distance between objects (cm)**	**Error (cm)**
	1	**2**	**3**		
2	3.8	2.3	2.4	2.8	0.8
4	5.0	4.2	4.4	4.5	0.5
6	5.2	5.2	5.7	5.4	0.6
8	7.7	7.7	7.5	7.6	0.4
10	9.2	9.7	9.4	9.4	0.6
12	11.6	11.8	12.1	11.8	0.2
14	13.2	13.9	13.5	13.5	0.5
20	19.2	19.6	19.3	19.4	0.6
30	29.4	30.6	30.5	30.2	0.2
40	40.6	40.2	41.4	40.7	0.7
Average Distance between Objects Error (cm)					**0.5**

CHAPTER 5

CONCLUSION AND FUTURE WORKS ENHANCEMENT

5.1 Introduction

The aim of this project is to give the rescuers in search and rescue activity an edge about the unknown environment by deploying SLAM algorithm to build a map of the unknown environment. Therefore, a custom-made robot is designed and constructed for SLAM algorithm to be implemented on the custom-made robot working under ROS framework for both practical implementation and simulation. The robot is then used to generate a 2D map about the unknown environment using SLAM-GMapping algorithm and 3D map using RGBD-SLAM algorithm. After the 2D map is built, navigation stack is incorporated into the robot for autonomous navigation to the 2D navigation goal set by user within the map built. Lastly, by running RGBD-SLAM algorithm and find_object_2D algorithm in parallel, the robot is able to identify, recognize and localize the object without repeating and missing whenever the robot come across the objects saved in the object database.

5.2 Review

This section will review how the project was carried out.

5.2.1 Design of Robot

In section 3.4, the process of designing and constructing a mobile robot is discussed. The mobile robot based on a differential drive structure with a two layer chassis to accommodate all the components and devices needed. The robot is successfully built and it is able to navigate around and feedback accurate sensory and odometry data for building of map, localizing itself in the unknown environment, detecting and recognizing of objects desired.

5.2.2 SLAM Algorithm

SLAM algorithm is used in this project to build map of unknown environment in practical implementation and simulation. Two types of SLAM algorithm have been successfully implemented in this project which are SLAM-GMapping algorithm for 2D map and RGBD-SLAM algorithm for 3D map. A Kinect Xbox 360 sensor is used to input depth images and RGB images into the RGBD-SLAM algorithm. A ROS node named *depthimage_to_laserscan* is used to convert the depth images into laser scan to be input into the SLAM-GMapping algorithm. In short, the robot is able to navigate autonomously to the 2D navigation goal set by user within the map built by using SLAM algorithm.

5.2.3 Object Recognition

In this project, SIFT algorithm is used to extract the features from the current frame of Kinect sensor while building the 3D map of the unknown environment. Two flat objects which are books and two curved objects which are human like dolls are saved into the object database. Therefore, whenever the robot come across the objects saved in the database while navigating around the unknown environment, the robot is able to detect, recognize and localize the objects. The objects are successfully being detected, recognized and localized without repeating and missing.

5.3 Conclusion

The objectives of this project are achieved by building a map of an unknown environment using SLAM algorithm and using the map for object recognition. A 2D occupancy grid map of office room E108 is built by using a custom-made robot which works under the Robot Operating System (ROS) framework. The SLAM-GMapping algorithm is being implemented on both practical implementation and simulation. A simulation environment which is similar to the office room E108 is designed by using Gazebo. Both practical implementation and simulation generate a similar 2D map of the environment of office room E108. Navigation stack is then incorporated into the robot for autonomous navigation and 2D navigation goal. After the map is built, the robot is able to localize itself by using Adaptive Monte Carlo Localization (AMCL) and navigate autonomously to the 2D navigation goal set by user. Dijkstra's algorithm is used to compute the shortest path which is the path with lowest cost values between the current positon of robot to the destination point. In a search and rescue area, the robot should has the ability to identify, recognize and localize the victim. Therefore, RGBD-SLAM algorithm is used to construct a 3D map of the surrounding environment and by implementing find_object_2D algorithm in parallel, the robot is able to identify and recognize the object in the object database when the robot come across the objects when navigating around. The location of object with respect to the pose of camera can also be obtained in terms of location of image pixels.

5.4 Future Works

In future, deep learning algorithm can be applied to SLAM algorithm so that loop closure detection and estimation of the position of robot can completed more accurately and efficiently. However, implementation of deep learning algorithm is much more complicated and it cannot be accomplished within the given period of time. Deep learning algorithm can be used to construct a dense-depth point cloud with the depth measurements obtained from Kinect sensor. By using convolutional neural network in deep learning algorithm, the depth prediction of points in image can be obtained and can be input into the SLAM algorithm.

Furthermore, image processing algorithm can also be implemented to detect human being instead of a flat object such as book or curved object such as human-like doll in this project. By detecting human being directly, it will be more convincing that the robot is able to localize the victim in the disaster area.

Lastly, the range of distance that can be travelled by the robot can be increased by powering the Kinect sensor with a 11.1V Lipo battery instead of a wall adapter. A replacement of Kinect sensor with Hokuyo laser scanner can also generate a higher accuracy map because a wider range laser scan input will be obtained.

In conclusion, this project has demonstrated that it is feasible to implement SLAM algorithm on the custom-made robot working under ROS framework in practical implementation and simulation. 2D map about the unknown environment using SLAM-GMapping algorithm and 3D map using RGBD-SLAM algorithm have also been successfully generated. Navigation stack has successfully been incorporated into the robot for autonomous navigation to 2D navigation goal set by user within the map built. Lastly, the robot is able to identify, recognize and localize objects saved in object database without repeating and missing.

REFERENCES

Abdelrasoul, Y., Saman, A.B.S.H. and Sebastian, P., 2016. A quantitative study of tuning ROS Gmapping parameters and their effect on performing indoor 2D SLAM. *2016 2nd IEEE International Symposium on Robotics and Manufacturing Automation (ROMA)*, pp. 1-6.

Abhishek, G., Prateek, M., Rishabh, L. and Neeti, S., 2014. PATH FINDING: A* OR DIJKSTRA'S?. *International Journal in IT and Engineering*, 2(1).

Allevato, A., 2017. *camera_calibration/Tutorials/MonocularCalibration - ROS Wiki*. [online] Available at: <http://wiki.ros.org/camera_calibration/Tutorials/MonocularCalibration> [Accessed 20 March 2018].

Bailey, T. and Durrant-Whyte, H., 2006. Simultaneous localization and mapping (SLAM): Part II. *IEEE Robotics & Automation Magazine*, 13(3), pp. 108-117.

Balasuriya, B.L.E.A., Chathuranga, B.A.H., Jayasundara, B.H.M.D., Napagoda, N.R.A.C., Kumarawadu, S.P., Chandima, D.P. and Jayasekara, A.G.B.P., 2016. Outdoor robot navigation using Gmapping based SLAM algorithm. *Moratuwa Engineering Research Conference (MERCon)*, pp. 403-408.

Bhosale Swapnali, B., Kayastha Vijay, S. and Harpale Varsha, K., 2014. Feature extraction using surf algorithm for object recognition. *International Journal of Technical Research and Applications*, 2(4), pp. 197-199.

Birk, A. and Carpin, S., 2006. Rescue robotics—a crucial milestone on the road to autonomous systems. *Advanced Robotics*, 20(5), pp. 595-605.

Blanco, J., n.d.. *Computer Vision Group - File Formats*. [online] Available at: <https://vision.in.tum.de/data/datasets/rgbd-dataset/file_formats> [Accessed 4 April 2018].

Borenstein, J., Everett, H.R., Feng, L. and Wehe, D., 1997. Mobile robot positioning-sensors and techniques. *Journal of Robotic Systems*, 14(4), pp. 231-249.

Brownlee, J., 2017. *A Gentle Introduction to the Bag-of-Words Model - Machine Learning Mastery*. [online] Available at: <https://machinelearningmastery.com/gentle-introduction-bag-words-model/> [Accessed 26 March 2018].

Cadena, C., Carlone, L., Carrillo, H., Latif, Y., Scaramuzza, D., Neira, J., Reid, I. and Leonard, J.J., 2016. Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age. *IEEE Transactions on Robotics*, 32(6), pp. 1309-1332.

Cha, S., 2015. *My Personal Robotic Companion*. [online] Available at: <https://sungjik.wordpress.com/> [Accessed 25 March 2018].

Cheein, F.A.A., Lopez, N., Soria, C.M., di Sciascio, F.A., Pereira, F.L. and Carelli, R., 2010. SLAM algorithm applied to robotics assistance for navigation in unknown environments. *Journal of neuroengineering and rehabilitation*, 7(1), p. 10.

Chen, L., Sun, P., Zhang, G., Niu, J. and Zhang, X., 2011. Fast Monte Carlo Localization for Mobile Robot. *Advanced Research on Electronic Commerce, Web Application, and Communication*, pp. 207-211.

Cho, B.S., Moon, W.S., Seo, W.J. and Baek, K.R., 2011. A dead reckoning localization system for mobile robots using inertial sensors and wheel revolution encoding. *Journal of mechanical science and technology*, 25(11), pp. 2907-2917.

Clearpathrobotics.com., 2015. *ROS Navigation Basics — ROS Tutorials 0.5.1 documentation*. [online] Available at: <http://www.clearpathrobotics.com/assets/guides/ros/ROS%20Navigation%20Basics.html> [Accessed 31 March 2018].