

ROOM CATEGORIZATION USING SIMULTANEOUS LOCALIZATION AND MAPPING AND CONVOLUTIONAL NEURAL NETWORK

by

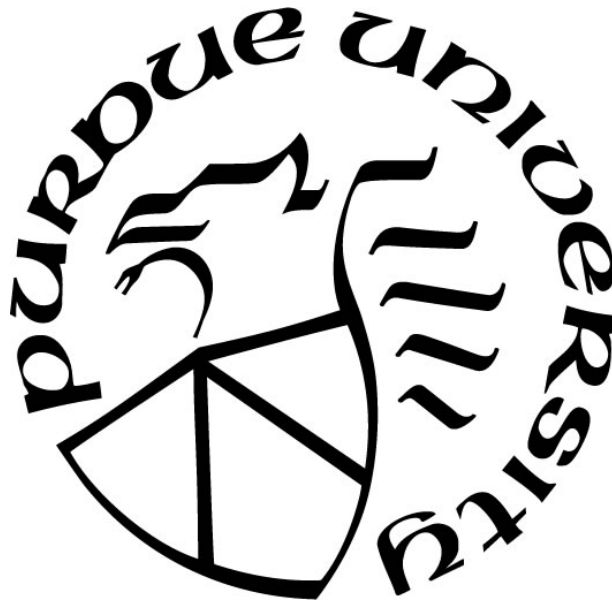
Iman Yazdansepas

A Thesis

Submitted to the Faculty of Purdue University

In Partial Fulfillment of the Requirements for the degree of

Master of Science in Electrical and Computer Engineering



Department of Electrical and Computer Engineering

Hammond, Indiana

August 2020

THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF COMMITTEE APPROVAL

Dr. Nasser Houshangi, Chair

School of Engineering

Dr. David Kozel

School of Engineering

Dr. Donald Gray

School of Engineering

Approved by:

Dr. Xiaoli Yang

Dedicated to my lovely wife and my parents.

ACKNOWLEDGMENTS

I wish to thank my advisor, Dr. Nasser Houshang, for his scientific insights and guidance in this research. Besides teaching technical expertise within me, he also took every opportunity to make me a well- rounded researcher. I have always marveled at his scientific and engineering knowledge across various fields. I have learned work ethic, focus, and dedication from the best. His passion for research and scientific truth will always inspire me.

I would also like to thank my other committee members, Dr. David Kozel and Dr. Donald Gray, for their patience, inputs, and feedback. I would like to thank the peers in my research group, for all the ideas and brainstorming with them.

TABLE OF CONTENTS

LIST OF TABLES	7
LIST OF FIGURES	8
ABSTRACT.....	10
1. INTRODUCTION	11
1.1 Motivation and Objective	12
1.2 Literature Search.....	12
1.2.1 Simultaneous Localization And Mapping (SLAM)	12
1.2.2 Room Categorization with LiDAR.....	13
1.3 Thesis Overview	16
2. SIMULTANEOUS LOCALIZATION AND MAPPING WITH LIDAR	17
2.1 Gmapping.....	20
2.1.1 LiDAR	20
2.1.2 Occupancy Grid Mapping.....	22
2.1.3 Grid-Based SLAM with Rao-Blackwellized Particle Filter	24
3. ROOM CATEGORIZATION	28
3.1 Deep Learning.....	28
3.1.1 Biological Neural Network.....	28
3.1.2 Artificial Neural Network.....	29
3.2 Convolutional Neural Network.....	36
3.3 Room Categorization with Convolutional Neural Network	45
4. SLAM IMPLEMENTATION	46
4.1 Simulations	46
4.2 Experimentations	51
4.2.1 Hardware Setup	51
4.2.2 Software Setup.....	53
4.2.3 Robot Navigated in a Research Lab	53
4.2.4 Robot Navigated in an Empty Room.....	54
4.2.5 Navigation in an Apartment.....	57

5. ROOM CATEGORIZATION SIMULATIONS	58
5.1 Simulation Setup.....	60
5.2 Training the Network.....	62
5.3 Simulation Results	63
6. CONCLUSIONS AND FUTURE WORK.....	67
APPENDIX A. ROBOT SPECIFICATIONS	68
APPENDIX B. ROBOTIC OPERATING SYSTEM SOFTWARE PACKAGES AND SOFTWARE TOOLS USED.....	71
APPENDIX C. MATLAB SIMULATION PROGRAMS	74
REFERENCES	80

LIST OF TABLES

Table 3.1 Results of SoftMax function for the 4 different number	36
Table 4.1 Comparison between the actual and estimated dimensions derived from the map constructed using Gmapping.....	49
Table 4.2 Comparison between the actual and estimated length.....	56
Table 5.1 Dataset of the raw odometry and LiDAR used[26].	59
Table 5.2 Accuracy for each room classification.....	64
Table 5.3 Comparision between the results of our work with [21]	65
Table 5.4 Confusion matrix for the apartment rooms that obtained in section 4.2.5.....	65

LIST OF FIGURES

Figure 2.1 Robot trajectory (a) without SLAM and (b) with SLAM.....	17
Figure 2.2 Actual position of the robot versus estimated.	18
Figure 2.3 Thickness of the red lines become thicker, while the robot is moving around in the environment, showing the stronger correlation applied each time new observation occurred.	19
Figure 2.4 LiDAR working concept illustration.	21
Figure 2.5 LiDAR SICK LMS111 reading angle.	22
Figure 2.6 2D examples of an occupancy grid map.....	23
Figure 2.7 Robot range finder (LiDAR) is seeing the occupied part of the environment.	23
Figure 2.8 Motion model for odometry and scan estimation.....	25
Figure 2.9 Particle filter distribution for grid mapping. (a) in an open area, the odometry data used for particle filter. (b) In this situation, the laser data only available for the sides of the corridor. Because of the uncertainty for the end of the corridor, the odometry data used for distribution. (c) all the sides and the end of the corridor is distinguishable with LiDAR.	26
Figure 3.1 (a) Left Neuron and myelinated axon, with signal flow from inputs at dendrites to outputs at axon terminals. (b) Right anatomy of a multipolar neuron and how synaptic terminal connect.	28
Figure 3.2 Simple Perceptron	29
Figure 3.3 Perceptron workflow diagram. A simple example of how one single neuron works in Artificial Neural Network (ANN).....	30
Figure 3.4 List of the activation functions used in perceptron	31
Figure 3.5 Simple perceptron with one hidden layer between input and output.	32
Figure 3.6 Sigmoid function output for value in the range of (-10,10)	35
Figure 3.7 Sigmoid graphs with three different bias.....	35
Figure 3.8 The interpretation of the character "X" in binary format. (a) is the pixels of the captured image and (b) is the constructed matrix in computer memory.	37
Figure 3.9 Three different images for the character 'X'	37
Figure 3.10 Different features in the X character image	38
Figure 3.11 Filtering process with one of the features.....	38
Figure 3.12 Results of the filtering for the left diagonal arm of the X	39
Figure 3.13 Convolutional operation illustration.....	40

Figure 3.14 Rectified linear unit graph	41
Figure 3.15 (a) Convolved matrix, (b) ReLu function applied to the convolved matrix.....	41
Figure 3.16 MaxPolling the left arm of X character	42
Figure 3.17 All three features passed to one simple hidden layer	42
Figure 3.18 Connecting few hidden layers.	43
Figure 3.19 Fully connected layer	43
Figure 3.20 Fully connected layer feeds the input of simple ANN	44
Figure 4.1 Gazebo simulation of rectangle rooms, yellow color shows the trajectory of the robot (a) simple room (b). corridor-like room.....	46
Figure 4.2 (a) Simple room in RVIZ and (c) results of Gmapping in the PGM file. (b) corridor-like room in RVIZ and (d) results of Gmapping in PGM file.	47
Figure 4.3 YAML content.....	48
Figure 4.4 A magnified image of the left side of a corridor-like room that shows the tiny pixels	50
Figure 4.5 Map errors vs. actual dimensions	50
Figure 4.6 Jackal Robot. (a) inside the robot equipped with a microcontroller board for driving motors and sensor drivers and the battery. (b) shows the antenna for WiFi and also GPS antenna. (c) the second robot inside the processor and the CPU of the robot. (d) Joystick for moving the robot. (e) LiDAR and camera sensors.....	52
Figure 4.7 Potter 104 maps generated by Gmapping.....	53
Figure 4.8 Potter 315 maps generated by Gmapping, (a – d) in each stage that map constructed by Gmapping ROS package, RVIZ visualization shown. (e) the PGM file that created by the end of the Gmapping process.....	54
Figure 4.9 Potter 315 with actual and estimated dimensions. Blue color is Estimated size from Gmapping, and orange color indicates the actual length	55
Figure 4.10 Dimension's error in the generated map	56
Figure 4.11 (a) two beds, two baths apartment layout shown in separate parts. (b) actual layout (c) estimated map	57
Figure 5.1 Group (a & c) are the [26] output and (b & d) are the output map of our MATLAB	58
Figure 5.2 Layers construction.	62
Figure 5.3 Number of different rooms and sample of each room.....	63
Figure 5.4 Classification results for the bedroom.	64

ABSTRACT

Robotic industries are growing faster than in any other era with the demand and rise of in home robots or assisted robots. Such a robot should be able to navigate between different rooms in the house autonomously. For autonomous navigation, the robot needs to build a map of the surrounding unknown environment and localize itself within the map. For home robots, distinguishing between different rooms improves the functionality of the robot. In this research, Simultaneously Localization And Mapping (SLAM) utilizing a LiDAR sensor is used to construct the environment map. LiDAR is more accurate and not sensitive to light intensity compared to vision. The SLAM method used is Gmapping to create a map of the environment. Gmapping is one of the robust and user-friendly packages in the Robotic Operating System (ROS), which creates a more accurate map, and requires less computational power. The constructed map is then used for room categorization using Convolutional Neural Network (CNN). Since CNN is one of the powerful techniques to classify the rooms based on the generated 2D map images. To demonstrate the applicability of the approach, simulations and experiments are designed and performed on campus and an apartment environment. The results indicate the Gmapping provides an accurate map. Each room used in the experimental design, undergoes training by using the Convolutional Neural Network with a data set of different apartment maps, to classify the room that was mapped using Gmapping. The room categorization results are compared with other approaches in the literature using the same data set to indicate the performance. The classification results show the applicability of using CNN for room categorization for applications such as assisted robots.

1. INTRODUCTION

Nowadays, robots are part of the everyday life of humans more than any other eras. The technological advancement in computers, improving computational power, provides the robotic industry to develop many systems such as assisted robots or autonomous robots. Mobile robots are an important part of the robotic industries to perform challenging tasks such as autonomous navigation in known and unknown environments without any human supervision. For achieving this goal, many Simultaneously Localization And Mapping (SLAM) algorithms were developed.

Simultaneous Localization and Mapping (SLAM) is the process by a mobile robot to build or update a map of an unknown environment and simultaneously use the map to localize itself in the unknown environment. SLAM estimates the robot's trajectory and landmarks position in real-time without any prior knowledge of the environment. SLAM can be performed using different sensors. One of the sensors used in mobile robots is LiDAR. LiDAR can measure the distance between the robot and any environment's landmark. The output of the SLAM with LiDAR is the 2D or 3D layout map of the environment based on the LiDAR configuration. In this research, 2D LiDAR was used, and the 2D map of the environment is generated. Another useful sensor utilized in the SLAM algorithm is the Inertia Measurement Unit (IMU), providing the orientation, velocity, and position of the robot. These two sensors provide the key information to the SLAM algorithm for localization and mapping. Many researchers have used data from the camera in their SLAM algorithms. The reason for selecting a LiDAR as the main sensor is because it can function and detect accurate dimensions even in dark environments, unlike the camera. The second part of this research discusses classifying the home's room types (Bedroom, Bathroom, Kitchen, etc.) by using the generated 2D map from SALM and Convolutional Neural Network (CNN). CNN techniques are used in many different applications in today's life and are one of the branches of Artificial Intelligent (AI). CNN uses an artificial neural network, which is inspired by the human brain, to detect and distinguish between different object's images as applied to many real-time applications like voice recognition or automatic real-time translation.

1.1 Motivation and Objective

As mentioned before, building the map of an unknown environment and localization within the environment is one of the critical issues that need to be resolved for mobile robots in many applications. There are many different approaches to solving the SLAM problem. This research uses Grid Mapping (Gmapping) technique because it has better performance and accuracy compared to others. Gmapping is also implementable under the Robotic Operating System (ROS). The second part of this research, as stated before, uses CNN because it has good accuracy in detecting and classifying images. The convolution neural network can distinguish between different rooms in an apartment or houses performing room categorization.

1.2 Literature Search

SLAM problem is one of the oldest issues in the robotic field, and scientists have been doing research for the last two decades, and there are many papers published from different scientists around the world. Artificial neural network and convolutional neural network approaches are not new, but research on CNN rose up significantly for the last few years. As stated before, the reason is the advancement of the CPU and computer computation power. In this section, the corresponding literatures are reviewed, as one section covering SLAM, and the other discussing CNN.

1.2.1 Simultaneous Localization And Mapping (SLAM)

One of the basic and fundamental tasks for home service robots is building a map of unknown dynamic environments. A primitive work in SLAM had been done by R.C. Smith and P. Cheeseman, titled "On the Representation and Estimation of Spatial Uncertainty" in 1986[1]. This research describes the general approach for estimating the location of the mobile robots using its sensors. The research explained how a Kalman filter is used to reduce the uncertainty of the robot's location by estimating the future location of the robot. Another work in this area was Leonard, J.J., and Durrant-Whyte, H.F in 1991, which presented the research titled "Simultaneous Map Building and Localization for an Autonomous Mobile robot"[2]. As was discussed, the problem of the mobile robot to simultaneously making the map and also localize itself within the map without any prior data about the environment.

Durrant-Whyte, H.F et al., in 2000, published the paper[3], which is shown the approach of how deleting some of the landmarks from the map is not increased the error of the SLAM but it can reduce the computational power requirement. Around the same time, Ayache and Faugeras [4] using visual information and Laumond and Chatila [5] and Crowley [6] by using an ultrasonic sensor, implemented the SLAM based on Kalman filter. These researches show when the robot is moving in an unknown environment and observing the location of the landmarks, the estimates of these landmarks locations are highly correlated since all of them have a similar error at the time of observation.

Although the Kalman filter was used more to solve the SLAM problem in the earlier research, the FastSLAM introduced by Montemerlo et al. [7] was proposed later deal with uncertainty. The researchers that used the Kalman filter focused on improving the performance of Kalman Filter SLAM, assuming linear Gaussian Distribution. FastSLAM, on the other hand, uses particle filter, which is based on non- linear process and non-Gaussian distribution. FastSLAM inspired by probabilistic mapping by Thrun [8] and Murphy [9]. Later on, Doucet and et al.[10] introduced Rao-Blackwellized particle filters, which are followed by Montemerlo et al. [7] research. One of the biggest problem with Rao-Balackwellized particle filter is the number of particle required for build a map. This issue increases the number of computations significantly.

Reducing the number of particles is one of the significant challenges in the Rao-Blackwellized particle filter. Giorgio Grisetti and Cyrill Stachniss [11] have introduced an approach that increases the performance of the Rao-Balackwellized particle filter to address the SLAM problem by using the grid map and reducing the number of particles. They were adapted the resampling method to make the map accurately by less number of particles. Giorgio Grisetti and et al. developing the Robotic Operating System (ROS) software package called Gmapping[12] which is the impilimintation of their research [11].

In this research Gmapping used to construct the map, since Gmapping is one of the robust approaches which is implemented in ROS Platform for making map of enviroment. Gmapping will be discussed in chapter 2.

1.2.2 Room Categorization with LiDAR

SLAM can be used in many different applications, from home service robots to driverless cars or even underwater robots. In all applications, the very first and major goal is to get

an accurate map and localize the robot precisely within that map. Constructing an accurate map is essential because the other mobile robot applications will utilize the information.

Another crucial primary task for a service robot is the ability to distinguish between different spaces. Recently, many groups of scientists shown interest in a topic called room categorization. Room categorization or room classification is the name of the method that the robot can distinguish and classifies the room by observing a specific object or signature of the spatial model of a place. At the end of the scanning and mapping of the environment, the robot should be able to recognize and label the rooms. Room labeling or room categorization capability of the robot can improve the capability of the assistance robot.

There are number of research performed for the representation of the space. Albert Elfes in 1990 [13] introduced the Occupancy Grids map, which uses the sensor data to make the geometry of space. The Occupancy Grid is a 2D or 3D map that estimates the occupancy state of each part of the spatial environment. H. Choset and et al. [14] uses graphs to represent the spatial model of the environment. Both approaches have some strengths and weaknesses. The [13] approach requires accurate determination of the robot's position, which in some cases, is very difficult to obtain such accuracy at the beginning of the map building. Although [14] approach, not requires the accurate determination of the robot position, it has difficulty constructing the map in a larger-scale environment. S. Thrun and et al. [15] uses both methods to make a more accurate map by covering the downside of each method with the other one. All the mentioned works [12-14] using different techniques to solve the SLAM problem by maintaining the spatial model of the environment.

Having an accurate map is very important in the work of the room categorization for service robots since the accurate labeling of the spatial model relies on a precise map of the environment. O.M Mozos and et al. [16] uses AdaBoost [17] method to extract the features of the spatial model from LiDAR sensor data. In [18] the AdaBoost method is used to extract the spatial features to distinguish between Corridor, Room, and Doorway. In [19], A. Swadzba and et al. used 3D LiDAR data to distinguish between office or meeting room by focusing on the object extracted from the 3D LiDAR data. Although approaches discussed in [16] and [19] are very useful for some applications, they can not classify all rooms category like kitchen, bedroom, etc.

Wu J, Christensen, and et al. [20] introduced the method which uses a Visual Place Categorization (VPC) utilizing the information from the camera to predict the room category.

Despite the Visual SLAM and Visual categorization benefits such as ease of use in an outdoor environment in terms of building the map and by utilizing the visual feature extraction gives better results for room categorization, it has some downsides as well. The biggest problem with visual SLAM and visual categorization are the dependencies to the light intensity and the accuracy in comparison to the LiDAR sensor. Assumes the home service robot wants to do the task at night when all the rooms are dark. The camera wouldn't be the best option in this situation, and LiDAR is the best choice. Because of the mentioned advantages, many works either utilize LiDAR alone[21] or in combination with other sensors [16].

In [22] [23], LiDAR is used for visual odometry by taking the large data sets of the LiDAR data and compute the motion of the LiDAR between two consecutive sweeps. In [24], LiDAR data is used for computation and implementation of the loop – closure algorithm. The loop-closure is the name of the algorithm that detects the previously visited location in a constructed map by recognition of distinctive regions of the map from sensor data. In [25], LiDAR data is used with Convolution Neural Network (CNN) to make the robot able to learn semantic place labels. Peter Ursic and et al. [26] utilizing LiDAR data and adapt the hierarchical compositional models to detect the spatial feature of the environment for classifying the room categories.

The algorithm used in[26] is the extended version of Learning the Hierarchy of Parts(LHoP) from [20] call it Spatial the Hierarchy of Parts(sHoP). While [26] using the Hierarchical Spatial Model for categorizing rooms, the other work trying to implement semantic place labels by using a Convolutional Neural Network (CNN) [25].

As stated before [27] and [25], it can just classify the doorway, corridor, and rooms. These works use the environmental structure for detecting the shape of the space from semantic information. For instance, the corridor is the long narrow open space which door appears as short gape. In this work, the LiDAR and odometry data collected in [26] used to train the CNN for categorizing the rooms. Unlike the [27] and [25], the goal of this research is not to detecting only doorway or corridor, but like [26], it aims to classify the rooms between Bedroom, Bathroom, Kitchen, etc. categories.

1.3 Thesis Overview

The goal of this research is to show the applicability of using a Convolutional Neural Network to solve the room categorization problem. The purpose of this work is to train the Convolutional Neural Network to distinguish and to label the rooms for home service assistance robots. This research is divided into two major parts. First, the method is discussed for making the 2D map of the environment using the Gmapping SLAM approach. The second part explains the training of a convolutional neural network with the generated maps from the first part and the maps collected in [26] for room categorization.

Chapter 2 describes the Grid mapping (Gmapping) Simultaneous Localization And Mapping (SLAM) approach. The approach uses particle filters for constructing the map and to localize the robot within the map. Chapter 3 describes the convolutional neural network method in detail. It shows how the layers are working in neural networks and how the method is utilized in this research. Chapter 4 describes the experimental results on using Grid mapping (Gmapping) both in simulation and real environments. The results are analyzed to indicate the performance of the approach. Chapter 5 shows the simulation results of the room categorization Using CNN Followed by Chapter 6, presenting the conclusions and suggest future works.

2. SIMULTANEOUS LOCALIZATION AND MAPPING WITH LIDAR

For making the map of the environment, the robot must be able to take the observation of several unknown landmarks using its sensors and process those observation data to make the map and localize itself. These sensors can be one or a combination of sensors like Light Detection and Ranging (LiDAR), sonar, and camera. The robot also must be equipped with the odometry system.

The odometry system can consist of Inertia Measurement Unit(IMU), shaft encoder, accelerometer, etc. With the odometry system, the robot can determine its position and orientation. The robot should scan the environment, and find the landmarks during motion. After a small movement, the robot should be able to scan the environment again and update the estimated location of the robot. Also, the estimated positions of the landmarks are determined. Ideally, the robot can be programmed to follow a trajectory moving from point A to point B. Since even the most accurate motors and sensors have small errors, in the real world, these errors accumulate during the robot movement, and the robot will never arrive at the final destination following the desired trajectory. The robot should be able to update its map every time new observation occurred to avoid error accumulation and trajectory failure. Such a robot usually observation occurred to avoid error accumulation and trajectory failure. Such a robot usually requires a high-performance computer and accurate sensors.

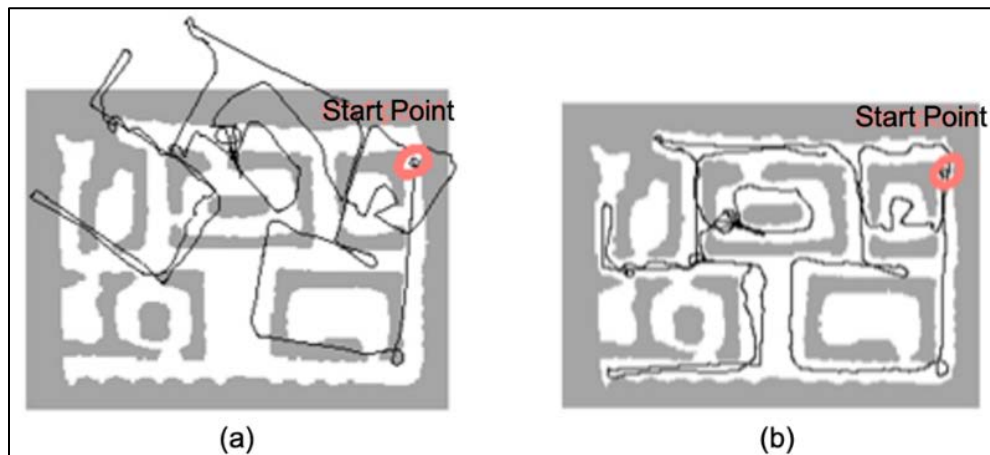


Figure 2.1 Robot trajectory (a) without SLAM and (b) with SLAM.

The most popular sensors that used in robots to capture environments are sonar, LiDAR, and camera. Each of them has its advantage and disadvantage. Figure 2.1 shows the trajectory of the robot with and without the error correction. In figure 2.1 (a), the robot doesn't correct its movement error during motion, which in figure 2.1 (b) shows the robot's trajectory while performing Simultaneously Localization and Mapping(SLAM). SLAM is one of the methods which builds the map, and within that map, localize itself to correct the estimated position through the trajectory.

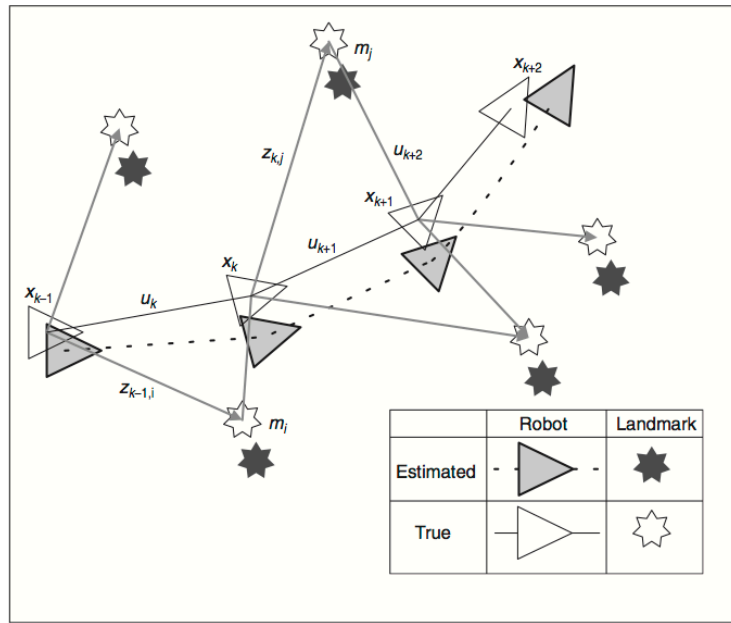


Figure 2.2 Actual position of the robot versus estimated.

Figure 2.2 [28] illustrates a moving robot in the environment which observes the landmarks with its sensors. At time instance k the following parameters are defined:

- x_k : location and orientation of the robot
- u_k : control vector, applied at time k-1 to drive the robot to state X_k at time k
- m_i : location of the ith landmark whose exact location is assumed time-invariant
- z_{ik} : an observation is taken from the robot of the location of the ith landmark at time k
- $X_{0:K} = \{x_0, x_1, \dots, x_k\} = \{X_{0:K-1}, x_k\}$ indicates robot positions history
- $U_{0:K} = \{u_0, u_1, \dots, u_k\} = \{U_{0:K-1}, u_k\}$ control inputs history
- $m = \{m_1, m_2, \dots, m_n\}$ includes all landmarks

- $Z_{0:K} = \{z_1, z_2, \dots, z_k\} = \{Z_{0:K-1}, z_k\}$ the set of all landmark's observations

Figure 2.2 shows the estimated and actual positions of the robot and landmarks. As stated before, the errors in sensors and actuators cause uncertainty in the estimated positions of the robot and the observed landmarks. The observation errors are almost similar for all landmarks. It means the estimated locations of landmarks are correlated. It can be concluded that the relative position of the landmarks m_i and m_j can be determined with good accuracy even without knowing the true location of the m_i and m_j . The reason is the source of all errors are robots itself, and therefore, those errors cancel each other. This is a crucial fact that in the SLAM algorithms, when the number of landmarks observation increases, the correlation between landmarks increases as well. As a result, the relative locations of the landmarks will improve, and it never diverges without any dependency on robot movements. For understanding this fact better, refer to figure 2.2 again. Assume the robot in location x_k and observing m_i and m_j . As the robot moves to x_{k+1} and observe m_j and the other landmarks, it still can be update m_i even the landmark is not observed in the new position. This is because the two landmarks location is highly correlated and by knowing the position of the m_j the position of the m_i can also estimated. In the other hand when the robot is in location x_{k+1} it observes the other landmarks as well, which this result to update the position of m_i .

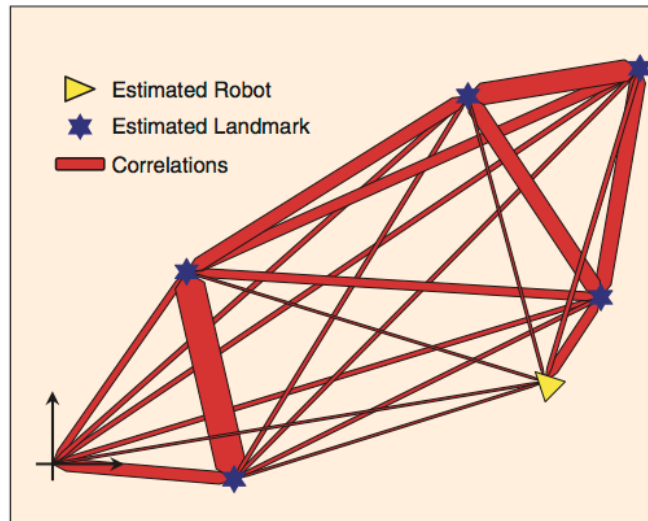


Figure 2.3 Thickness of the red lines become thicker, while the robot is moving around in the environment, showing the stronger correlation applied each time new observation occurred.

Figure 2.3 [28] illustrated the above elaboration. While the robot is moving and observing new landmarks, the correlation of the previously observed landmarks gets stronger. The thickness of the red lines between the landmarks illustrated the correlation between them.

For solving the probabilistic SLAM problem, an appropriate representation of the motion and observation model should be found. There are too many different approaches to solving such a problem. Three most common approaches are:

- Filter-based method, which estimates and updates the map recursively. In this approach, the map of the environment and the estimation of the robot location are updated as a probability density function. Kalman filter (EKF, UKF, SEIF, etc.) and particle filter are usually used for estimation under this category.
- Another approach is based on storing some keyframes in the environment to estimate the movement. These approaches are mostly used in visual SLAM like ORB-SLAM or Google's cartographer.
- The use of the Convolutional Neural Network to perform SLAM is another approach. RatSLAM is categorized under this approach.

In this research, the approach used is a Grid mapping(Gmapping), which is utilized Rao-Blackwellized particle filter [11] to construct the grid maps from 2D LiDAR data.

2.1 Gmapping

Gmapping is a highly efficient Rao-Blackwellized Particle Filter (RBPF) to learn grid maps from laser range data. In the next sections, LiDAR, occupancy grid map, and at the end, the Rao-Blackwellized particle filter will be discussed as all are utilized in the Gmapping approach for SLAM.

2.1.1 LiDAR

Light Detection and Ranging(LiDAR) is a sensor that uses light speed to measure the distance. LiDAR can be used in different applications, from highway traffic control to mobile robots and autonomous vehicles. As mentioned in the previous section, the mobile robot should collect environmental data. For finding the distance between the mobile robot and landmarks, there

are two different far sensing sensors, sonar and LiDAR. Both of them have their advantage and disadvantage, which limits their capability in different applications.

Sonar sensors are very cheap, useable in underwater applications, and able to detect glass. Although, due to a wide reading angle (more than 30 degrees), has low accuracy. LiDAR has good accuracy and a small reading angle (less than 0.5 degrees), but LiDAR is not a perfect candidate in environments that has mirrors or glasses. However, a number of researchers have worked on this problem, but this is still one of the challenging issues when LiDAR is used in mobile robots and SLAM applications. Despite the problem with mirrors and glasses, one of the biggest advantages is the independence of the light, making LiDAR more effective to perform scanning in dark environments.

In this research, the LiDAR sensor SICK LMS111 was used for performing SLAM. Detailed technical specifications of SICK LMS111 are discussed in Appendix A. Figure 2.4 illustrates the basic concept of the laser sensors. As shown in figure 2.4, the sensor has a light emitter and receiver and a motor. The sensor also has a processor to control all the components. When LiDAR emits the beam's light and hits an obstacle, the beam's lights reflect back from the surface of the obstacle to the LiDAR sensor receiver. The processor in the sensor measures the time of light travel between emitter and receiver. Since the speed of light is a constant, the distance can calculate by

$$Distance = (Speed\ of\ Light * Time\ of\ the\ Flight)/2 \quad (2.1)$$

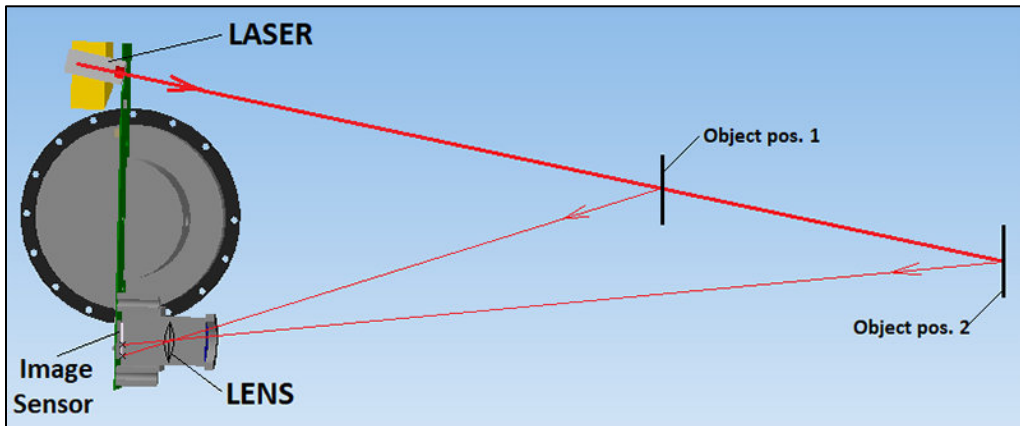


Figure 2.4 LiDAR working concept illustration.

The motor inside the sensor changes the orientation of the LiDAR beam to measure the distance from different angles. The spatial shape of the environment will be constructed by recording the length of each orientation.

Each LiDAR sensor has a different reading angle. For instance, the LiDAR sensor, that used in this work, has 270 degrees of reading angle. It also has the stride, about 0.5 degrees. Another parameter is the scanning frequency, which defines how many scans per second from 0 to 270-degree sensors can perform. The LiDAR used in this research has a 50 Hz scanning frequency. Figure 2.5 shows the LiDAR reading angle. If a LiDAR doesn't see any object or landmark, it returns the null or any other character that is an indication of not seeing any objects in its range. Each different LiDAR has a limited range. For instance, the LiDAR that is used in this work has a range of 20 meters. The LiDAR sensor is one of the sensors used to collect the environment information. This information, later on, used to construct the map of the environment. In the next section, the use of the LiDAR sensor to perform grid mapping is discussed.

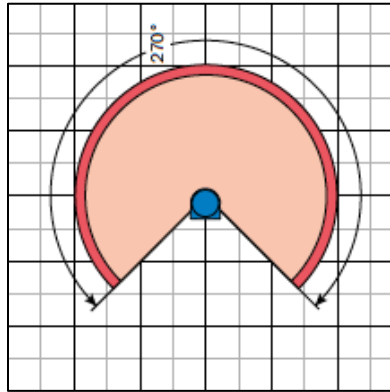


Figure 2.5 LiDAR SICK LMS111 reading angle.

2.1.2 Occupancy Grid Mapping

Occupancy Grid mapping is one of the methods for constructing the map of the environment by dividing the environment into the grid map. Then, estimating the occupancy of each cell of the grid map by reading the scan data. Figure 2.6 illustrating the 16 cell grid map. Each time the laser scan has reading, it estimates the probability of the cell to be occupied or not. The

terms occupancy is defined as the random variable between 0 and 1. Zero indicates the cell is free, and one indicates occupied as shown below.

$$m_{x,y}: \{free, occupied\} = \{0,1\} \quad (2.2)$$

Figure 2.6 shows the 2D example of Occupancy Grid Map. In the figure, the whole environment map, divided into the small cells in which it has shown only 16 of them in this figure.

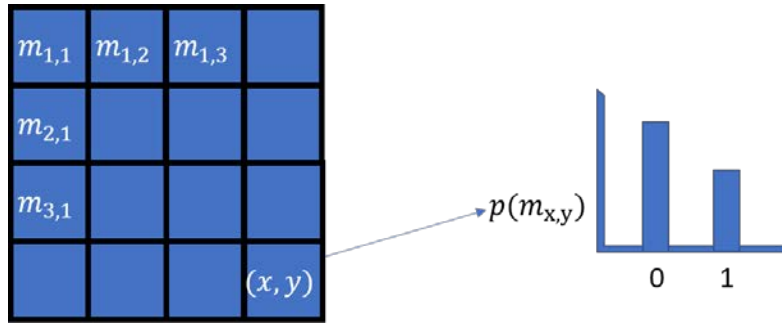


Figure 2.6 2D examples of an occupancy grid map.

The Bayesian filtering recursively updates $P(m_{x,y})$ each cell. For updating the recursive Bayesian filter, range measurement data needed from the environment. The range measurement data comes from a LiDAR sensor.

Figure 2.7 illustrated the way the robot observes the environment. By reading the LiDAR measurement for every part of the environment, the robot can find if that particular range is occupied or not. In other words, every time LiDAR can read the distance, it means in that distance, there is an occupied area. Since the LiDAR sensor scanning by rotating, the map can be constructed. Without robot movements, the robot can only make the map within the scanning range of LiDAR.



Figure 2.7 Robot range finder (LiDAR) is seeing the occupied part of the environment.

This short overview of the occupancy grid map just gives a brief understanding of how the grid map works. For a real robot to construct a complete map of the environment, the robots need to move around and utilize the odometry data within the algorithm to perform the complete validation for different locations of the environment.

2.1.3 Grid-Based SLAM with Rao-Blackwellized Particle Filter

Murphy, in [29], shows that the Rao-Blackwellized particle filter (RBPF) is based on the idea of estimating the joint posterior $p(x_{1:t}, m \mid z_{1:t}, u_{1:t-1})$ where m is the map, and $x_{1:t}$ is the trajectory of the robot given the observation $z_{1:t}$ and the odometry measurements of $u_{1:t-1}$ from robots sensors. The joint posterior probability density function can be expanded as follows:

$$p(x_{1:t}, m \mid z_{1:t}, u_{1:t-1}) = p(m \mid x_{1:t}, z_{1:t}) * p(x_{1:t} \mid z_{1:t}, u_{1:t-1}) \quad (2.3)$$

As it shows in equation 2.3, it consists of two parts. $p(x_{1:t} \mid z_{1:t}, u_{1:t-1})$ can estimate the trajectory of the robot, given observation, and the odometry measurement of the robot. Then, the trajectory of the robot will use in $p(m \mid x_{1:t}, z_{1:t})$ to obtain the map. This approach, referred to as a Rao-Blackwellized particle filter, gives a very efficient computation algorithm since the map of the environment highly correlates to the pose estimate of the robot.

A particle filter is the name of a method that uses a set of particles to represent the posterior distribution. Particle filter method used in the models that are not normally distributed. For instance, consider posterior distribution $p(x_{1:t} \mid z_{1:t}, u_{1:t-1})$, for modeling the probability of the robot trajectory, the Gaussian function cannot use since the trajectory of the robot is not normally distributed.

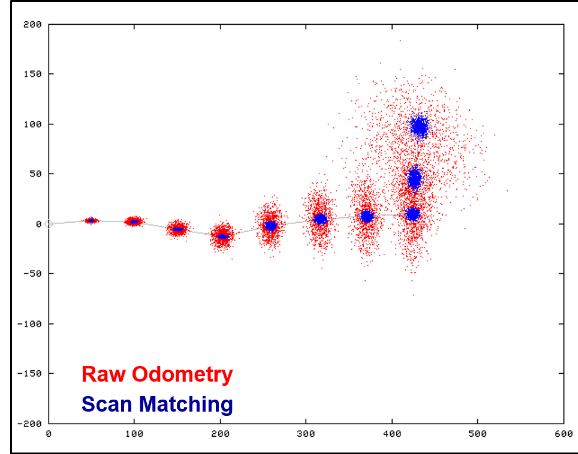


Figure 2.8 Motion model for odometry and scan estimation.

Figure 2.8[30] illustrated the simple actual robot trajectory versus the estimation of the trajectory based on raw odometry and scan. The red dots in figure 2.8 are the particle distribution for the location of the robot based on the odometry, and the blue dots are the estimation based on the scan matching. Scan matching is the method to find the relative pose (or transform) between the two robot positions where the scans were taken. The scans can be aligned based on the shapes of their overlapping features. In figure 2.8, the red and blue dots are the distribution of the particles, and they represent the location of the robot. As it shows in figure 2.8, the red dots get divergent while the robot is moving ahead. The reason is the error of the odometry error, which is accumulated, and therefore the uncertainty of the robot location increases. The blue particle in figure 2.8 shows the uncertainty is much smaller because the observation took into account as well as odometry information and causing optimization of the pose of the robot.

For mapping, a similar process in terms of the particle distribution performs. Figure 2.9 [30] shows the particle distribution in 3 different circumstances. From section 2.1.2, the grid map is constructed based on the existence of the obstacle. In Figure 2.9, the red lines are representing the LiDAR beam.

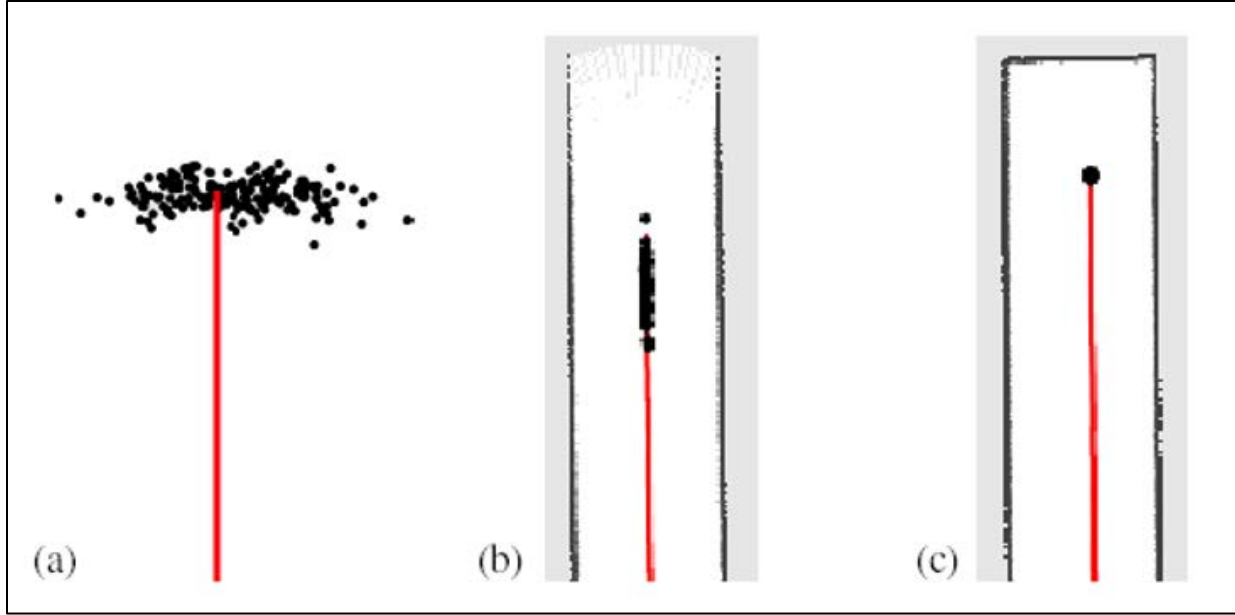


Figure 2.9 Particle filter distribution for grid mapping. (a) in an open area, the odometry data used for particle filter. (b) In this situation, the laser data only available for the sides of the corridor. Because of the uncertainty for the end of the corridor, the odometry data used for distribution. (c) all the sides and the end of the corridor is distinguishable with LiDAR.

Figure 2.9 (a) shows the situation that the robot is moving in a free area, and there is no object to reflect the laser beams causing zero observation information. In this case, the odometry particle becomes the distribution automatically. The black dot in figure 2.9 (a) shows the particle filter distribution based on the odometry. In figure 2.9 (b), the robot is moving inside the corridor, which the robot cannot see the end of the corridor. Therefore, the robot utilizes its observation to align itself to left and right since it obtains the accurate distance to the side walls. Although the sides are aligned very well, it has high uncertainty to the end of the corridor since there is no observation information for the end of the corridor. Therefore, the particle distribution is aligned in the line as it shows in figure 2.9 (b). Figure 2.9 (c) shows the case that the robot observes the end of the corridor and the particle distribution is concentrated in a very small area as it shows in the figure. It is obvious that constructing the map in Figure 2.9 (a) is not possible, but in figure 2.9 (b) and (c), the map can be constructed by having the observation data and the trajectory data. Gmapping is the name of an improved Rao-Blackwellized particle filter algorithm [11]. The improvement performs by using the less particle to get the same and even better results. Using less particle is important since it will reduce computational power.

In this research, the Gmapping is used under the Robotic Operating System(ROS) platform. Besides the less computational requirement, other reasons for using Gmapping is the accuracy of the generated map and easy implementation in the ROS. The generated map and the results in terms of the accuracy obtained by the Gmapping approach are discussed in detail in chapter 4.

3. ROOM CATEGORIZATION

Room categorization is an important phenomenon for real-time applications such as assisted robots to distinguish between different room categories, providing an ability to perform tasks better in the home environment. In this research, six different room categories are used, such as a bedroom, bathroom, kitchen, Livingroom, toilet, and corridor. Categorizing with the 2D Lidar and IMU data is challenging since the environmental map has a 2D layout. This challenge comes from the 2D layout, which is, in many cases, is difficult to distinguish the differences between rooms even with human eyes. In this research, the Jackal robot equipped with a SICK LM100 laser sensor is used. The room categorization approach taken is one of the deep learning methods called Convolutional Neural Network (CNN). Deep learning and CNN approaches are discussed in the next sections.

3.1 Deep Learning

3.1.1 Biological Neural Network

Deep learning or Artificial Neural Network (ANN) is a method inspired by brain neurons. Although, Neural networks theory introduced by Warren McCulloch and Walter Pitts [31] in 1943, the method was not applied to any real-time applications until 2011 because of the lack of processing and computational power. Figure 3.1[32] shows the neurons in our brains. In the figure, the neuron receives the signals from the input.

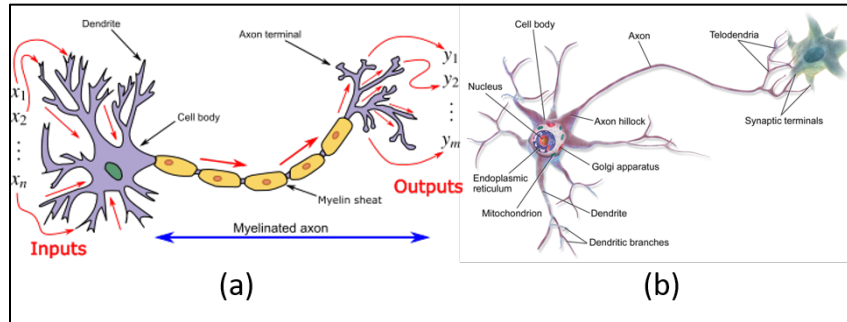


Figure 3.1 (a) Left Neuron and myelinated axon, with signal flow from inputs at dendrites to outputs at axon terminals. (b) Right anatomy of a multipolar neuron and how synaptic terminal connect.

This input can be any human sense data such as smell, vision, etc. and process them through the cell body and send them out throughout the axon to another cell input. In figure 3.1(a), the x_1 to x_n are the inputs, with raw values like 0 or 1.

3.1.2 Artificial Neural Network

Artificial Neural Network (ANN) is based on a simple model of the brain neural connections called a perceptron. Frank Rosenblatt, developed perceptron inspired by earlier work by Warren McCulloch and Walter Pitts. Frank Rosenblatt, created the machine called “Mark I Perceptron”[33] back in 1957 base on the idea that the weights of the artificial connections between neurons could change through a supervised process to minimize the error between actual and expected output. This supervised process keeps comparing the expected output with the actual one and base on the misfit between them, improving the learning performance. Figure 3.2 shows the simple perceptron with three inputs and one output. A perceptron takes several inputs (in this case, just three inputs) and each input, weighted and summed. If the result more than a defined threshold, the output is one; otherwise, it’s zero[34].

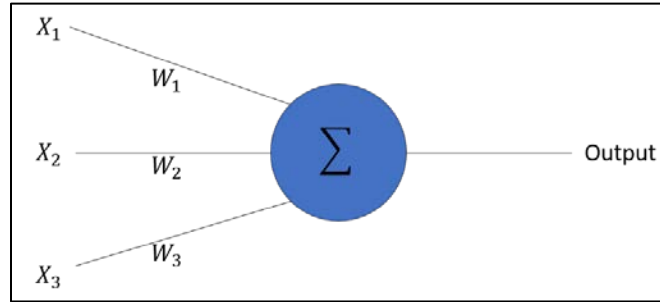


Figure 3.2 Simple Perceptron

Equation 3.1 shows the above description in the form of the algebraic term.

$$Output = \begin{cases} 0, & \sum_{j=1}^3 w_j x_j < threshold \\ 1, & \sum_{j=1}^3 w_j x_j \geq threshold \end{cases} \quad (3.1)$$

Assume the output of the perceptron is pre-known as one. If in the first iteration, the output of the sum term is less than the threshold, the output of the perceptron gives zero, which is different than the actual output, which is one. In the process of finding the best weights, all three weights should be changed until the sum term of equation 3.1 becomes greater than the threshold. After the output reaches the threshold, all the weights are stored and remain fixed, and the network considered trained. When the neural network model trained by all the known inputs (x_1, x_2, x_3) the model can use for predicting the unknown inputs. Figure 3.3 shows the complete perceptron workflow. Equation 3.1, in this figure, illustrated as an activation function.

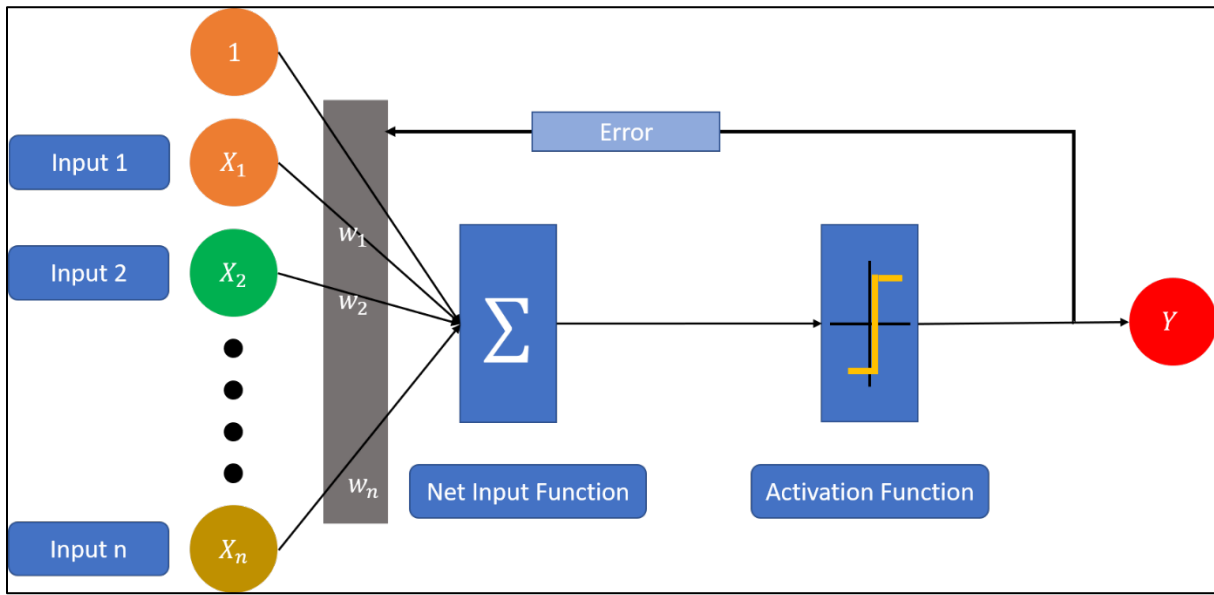


Figure 3.3 Perceptron workflow diagram. A simple example of how one single neuron works in Artificial Neural Network (ANN).

For simplicity, equation 3.1 is written as

$$Output = \begin{cases} 0, & w \cdot x + b < 0 \\ 1, & w \cdot x + b \geq 0 \end{cases} \quad (3.2)$$

where w and x are vectors whose components are the weight and inputs, respectively. Other change is to move the threshold to the other side of the inequality and to replace it by what's known as the perceptron's bias, $b \equiv \text{threshold}$. The bias can be a measure of how easy it is to get the one on perceptron output. For a perceptron with tremendous bias, it's straightforward for the

perceptron to output one. But if the bias is very negative, then it's difficult for the perceptron to output one. The activation function that is used within the perceptron can be changed according to the various applications. The above example uses unit step activation function.

Linear and the unit step is not practical for most applications as will be discussed later. Two of the popular activation functions are sigmoid, and the other is a Rectified Linear Unit (ReLU). ReLU is the activation function that is used in the convolutional neural network. Figure 3.4[35] shows more activations functions with their equations.

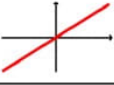
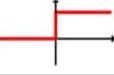



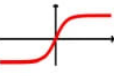

Activation Function	Equation	Example	1D Graph
Linear	$\phi(z) = z$	Adaline, linear regression	
Unit Step (Heaviside Function)	$\phi(z) = \begin{cases} 0 & z < 0 \\ 0.5 & z = 0 \\ 1 & z > 0 \end{cases}$	Perceptron variant	
Sign (signum)	$\phi(z) = \begin{cases} -1 & z < 0 \\ 0 & z = 0 \\ 1 & z > 0 \end{cases}$	Perceptron variant	
Piece-wise Linear	$\phi(z) = \begin{cases} 0 & z \leq -1/2 \\ z + 1/2 & -1/2 \leq z \leq 1/2 \\ 1 & z \geq 1/2 \end{cases}$	Support vector machine	
Logistic (sigmoid)	$\phi(z) = \frac{1}{1 + e^{-z}}$	Logistic regression, Multilayer NN	
Hyperbolic Tangent (tanh)	$\phi(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	Multilayer NN, RNNs	
ReLU	$\phi(z) = \begin{cases} 0 & z < 0 \\ z & z > 0 \end{cases}$	Multilayer NN, CNNs	

Figure 3.4 List of the activation functions used in perceptron.

The simple perceptron described above shows the basic functionality of the single artificial neural network. Although the single perceptron can solve some simple problems, this research required more complex ANN for image processing. The complex ANN consists of three important layers.

- An input layer, this layer gets the direct inputs from the data. In our case, each pixel of the image is received as an input in this layer.
- Hidden layer(s), this layer or layers consist of all summation and multiplication and activation functions. The input of this layer(s) is the input layer.

- Output layers, this layer usually has a classifier or regression function to make the decision.

The neural network can have as many hidden layers as required. Having a more hidden layer requires more powerful computational systems. For simplicity of describing the hidden layers figure, 3.5 shows a simple perceptron with just one hidden layer between input and output. Each circular node represents one single artificial neuron, and each arrow represents a connection from the output of one node to the input of another. Each neuron connection has carried a different weigh (w_k) for the next nodes. In layer L_1 three nodes in the input layer and four nodes in the hidden layer L_2 .

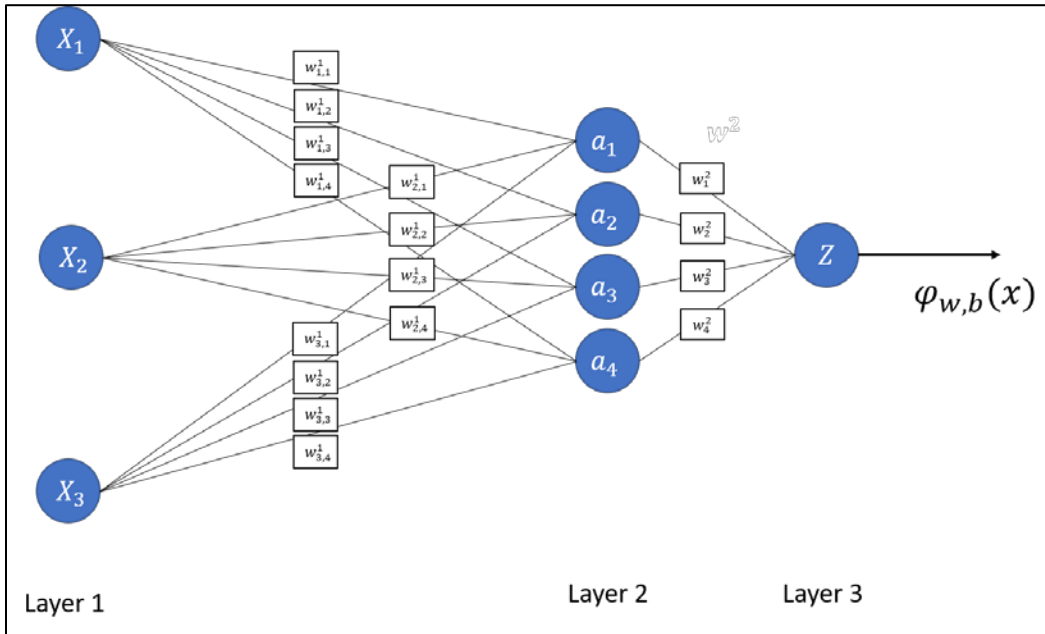


Figure 3.5 Simple perceptron with one hidden layer between input and output.

From equation 3.2, the output of layer 2 calculated as

$$A = W^1 * X + B \quad (3.3)$$

where A is the output of a hidden layer with a dimension of 4×1 , W^1 is the 4×3 weight matrix between layer one and layer 2, X is 3×1 input matrix, and B is the 4×1 bias matrix. The output of layer three is calculated by

$$Output = \varphi (W^2 * A + B^2) \quad (3.4)$$

where W^2 is the 1 x 4 weight matrix between layer 2 and layer 3, φ is the activation function, and B^2 is the scalar bias of the last layer threshold.

The dimension of the weight matrix of each layer is achieved by a number of the second layers' node, time to the number of the first layer. For instance, if the network has “n” nodes, in layer j and “m” nodes in layer j+1 the W^j has $m \times n$, which is, in our case, is 4×3 for W^1 . Components of W^1 and W^2 are

$$W^1 = \begin{bmatrix} w_{11}^1 & w_{21}^1 & w_{31}^1 \\ w_{12}^1 & w_{22}^1 & w_{32}^1 \\ w_{13}^1 & w_{23}^1 & w_{33}^1 \\ w_{14}^1 & w_{24}^1 & w_{34}^1 \end{bmatrix} \quad W^2 = [w_1^2 \quad w_2^2 \quad w_3^2 \quad w_4^2] \quad (3.5)$$

Therefore, equation 3.3 can be expanded as

$$A = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \end{bmatrix} = \begin{bmatrix} w_{11}^1 & w_{21}^1 & w_{31}^1 \\ w_{12}^1 & w_{22}^1 & w_{32}^1 \\ w_{13}^1 & w_{23}^1 & w_{33}^1 \\ w_{14}^1 & w_{24}^1 & w_{34}^1 \end{bmatrix} * \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix} \quad (3.6)$$

$$\begin{aligned} a_1 &= (w_{11}^1 x_1 + w_{21}^1 x_2 + w_{31}^1 x_3) + b_1 \\ a_2 &= (w_{12}^1 x_1 + w_{22}^1 x_2 + w_{32}^1 x_3) + b_2 \\ a_3 &= (w_{13}^1 x_1 + w_{23}^1 x_2 + w_{33}^1 x_3) + b_3 \\ a_4 &= (w_{14}^1 x_1 + w_{24}^1 x_2 + w_{34}^1 x_3) + b_4 \end{aligned} \quad (3.7)$$

Substituting the result of equation 3.6 to equation 3.4 follows

$$Output = \varphi (W^2 = [w_1^2 \quad w_2^2 \quad w_3^2 \quad w_4^2] * A = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \end{bmatrix} + B^2) \quad (3.8)$$

$$Output = \varphi(w_1^2 a_1 + w_2^2 a_2 + w_3^2 a_3 + w_4^2 a_4 + B^2) \quad (3.9)$$

For avoiding the complexity of equation 3.9, a_1, a_2, a_3 didn't substitute with the values from equation 3.7. From equation 3.2, the network output can be described as

$$Output = \begin{cases} 0, & w_1^2 a_1 + w_2^2 a_2 + w_3^2 a_3 + w_4^2 a_4 + B^2 < 0 \\ 1, & w_1^2 a_1 + w_2^2 a_2 + w_3^2 a_3 + w_4^2 a_4 + B^2 \geq 0 \end{cases} \quad (3.10)$$

As described earlier, for training the network, the output of equation 3.10 compares with the actual value, and if the mismatch happened, all the weights and biases change in order to get the correct output. The activation function in a neural network identifies if the output of the weighted sum value is above the defined threshold or not. For instance, if the unit step is chosen as an activation function, the output of the activation layer is either 0 or 1, causing a problem for classifying multiple outputs. This is one of the drawbacks of using the unit step as an activation function. The linear function also is not a good candidate because the outcome of each layer would be linear, and when it goes to the next layer, it will remain linear as well. Causing the nonlinear data to never map to this kind of function.

The sigmoid function is one of the most popular activation functions that use in many applications, as shown below.

$$s(x) = \frac{1}{1 + e^{-x}} = \frac{e^x}{1 + e^x} \quad (3.11)$$

The sigmoid function has a lot of properties that make it perfect for many applications [36].

- Non-linear function
- Output values range between (0,1) make it perfect for probabilistic problems
- Has a vast input range can be any large number or any small negative number

As shown in figure 3.6, the function output is between 0 and 1, make it useful for applications that have a probabilistic nature.

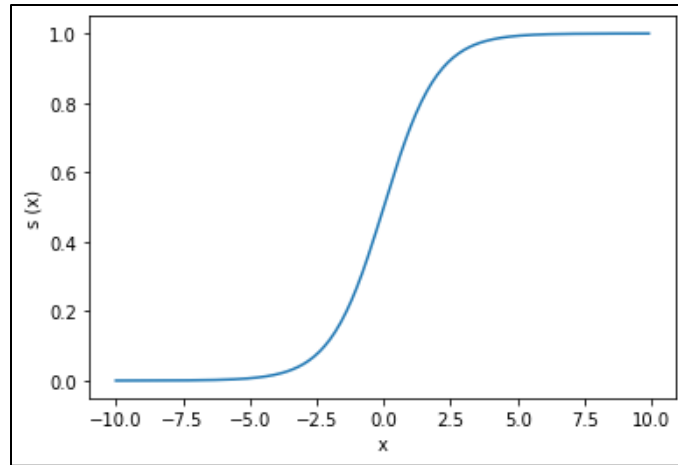


Figure 3.6 Sigmoid function output for value in the range of (-10,10).

Another essential part of the ANN is the bias node. A bias value allows the activation function to move left and right. Having good flexibility to shift the activation function left and right helps the data of the model fit better. The function usually used in the last layer of the ANN for making probabilities of the prediction. For instance, to recognize the picture of the car or airplane, this function can tell us the probability of an airplane or car. This function is useful for binary classification, since, in our case of room categorization, the classification is between 6 different rooms, the generalized version of the sigmoid function, which is called SoftMax function is used.

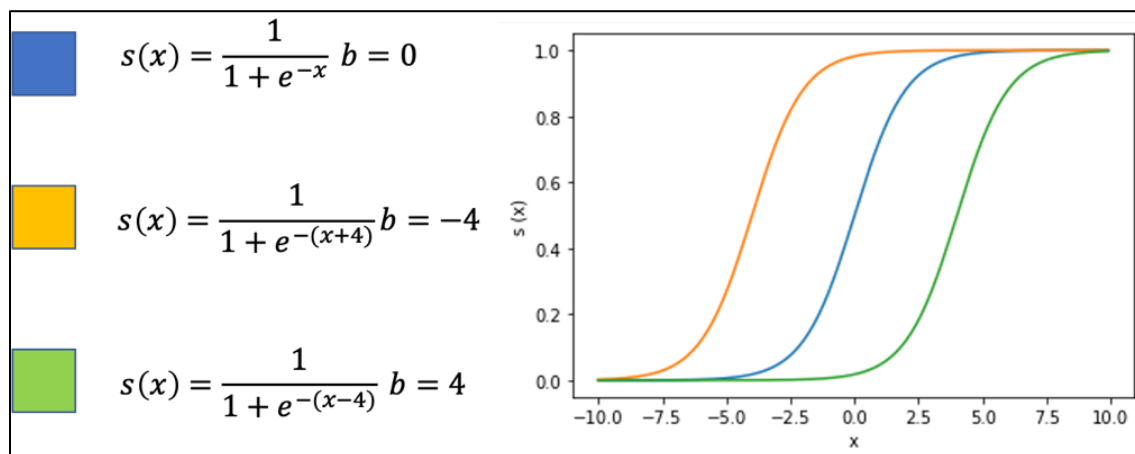


Figure 3.7 Sigmoid graphs with three different bias.

SoftMax function can handle multi classes, as described in equation 3.12. The SoftMax function is useful for some applications that need to identify the probability of more than two events. The output of the function is always between [0, 1] make it perfect for probability problems. Assume there are 4 data values of [-1, 0, 3, 5][37]. The probability of each data value is shown in table 3.1. These four numbers could be any values like the output of the hidden layer of the neural network for distinguishing four different objects. And the result in table 3.1 shows the probability of correctness of identifying the objects.

$$s(x_i) = \frac{e^{x_i}}{\sum_{j=1}^n e^{x_j}} \quad (3.12)$$

Table 3.1 Results of SoftMax function for the 4 different number

x	Numerator (e^x)	Probability($\frac{e^{x_i}}{169.87}$)
-1	0.368	0.002
0	1	0.006
3	20.09	0.118
5	148.41	0.874

In this research, a 2D map constructed from LiDAR is used to classify the rooms. After discussing the basics of the neural network, the Convolutional Neural Network, or in short CNN, is explained in section 3.2.

3.2 Convolutional Neural Network

Convolutional Neural Network (CNN) is used in many applications. From unlocking your phone with face recognition to autonomous cars, all such applications use the CNN algorithm. One of the popular usages of CNN is image processing and recognition. Before discussing CNN, let's look at an image. In figure 3.8, the image of the character “X” taken by a camera is represented by 8 x 8 pixels in monocular. Monocular means 1 represents black, and -1 interpreted as white. Figure 3.8 (a) shows the image with its matrix representation, as shown in figure 3.8 (b). Recognizing this character is easy.

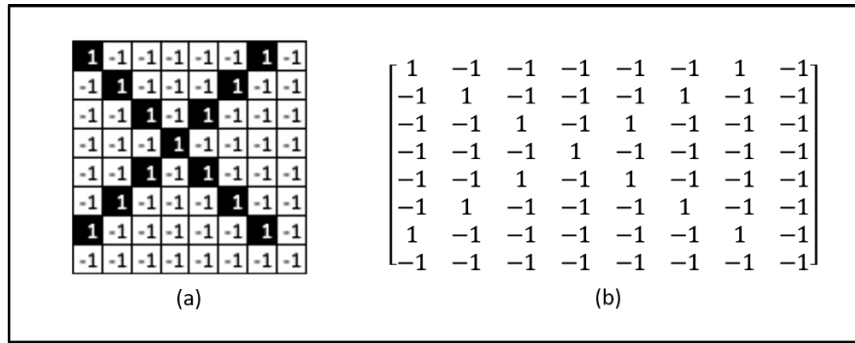


Figure 3.8 The interpretation of the character "X" in binary format. (a) is the pixels of the captured image and (b) is the constructed matrix in computer memory.

This task can be done by reading each member of the eight by eight matrix as an input and compare them with the matrix for known characters, and if all members value in the input matrix match with the known character 'X,' then the character X is detected. The problem with this approach is it can only detect the input character that exactly looks like the X. For instance, if the X character shape looks different, as shown in figure 3.9, the algorithm is no longer detect the character 'X.'

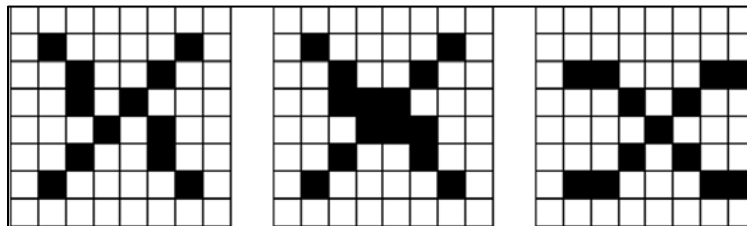


Figure 3.9 Three different images for the character 'X'.

As a result, a more complicated algorithm is needed to identify the desired character or object. In this case, instead of looking to match the entire image, start to match small pieces of the image. In the example presented, the X character can consist of three different features, which are shown in figure 3.10. Each feature framed with a different color for more clarity. The yellow and green frame shows the arms of the X, and the red one shows the core of the X. These features can match with the different pieces of the image.

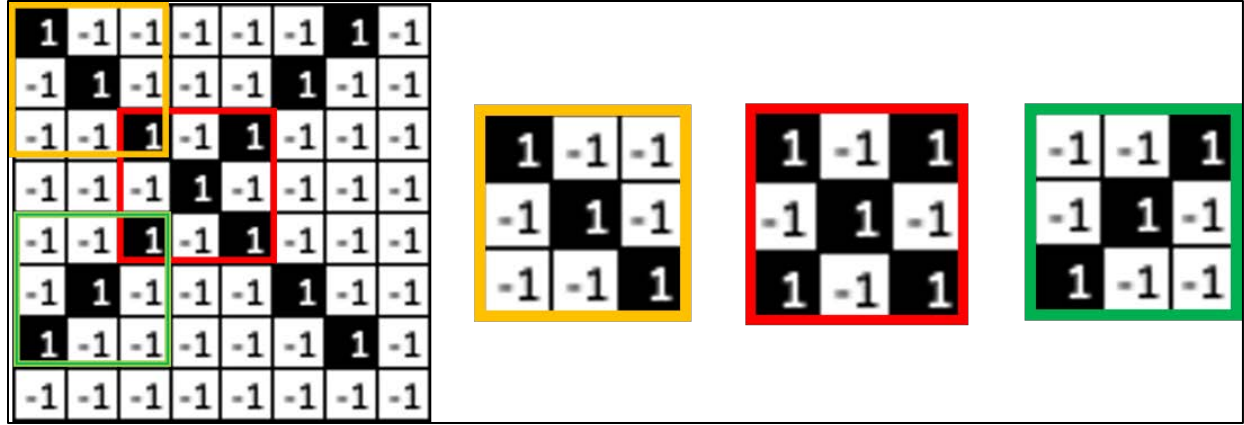


Figure 3.10 Different features in the X character image.

Convolutional filtering is used to match features piece by a piece. Figure 3.11 illustrates the convolutional filtering process for the left arm of the character “X.” The yellow frame in figure 3.11(a) is the feature frame in figure 3.11(b).

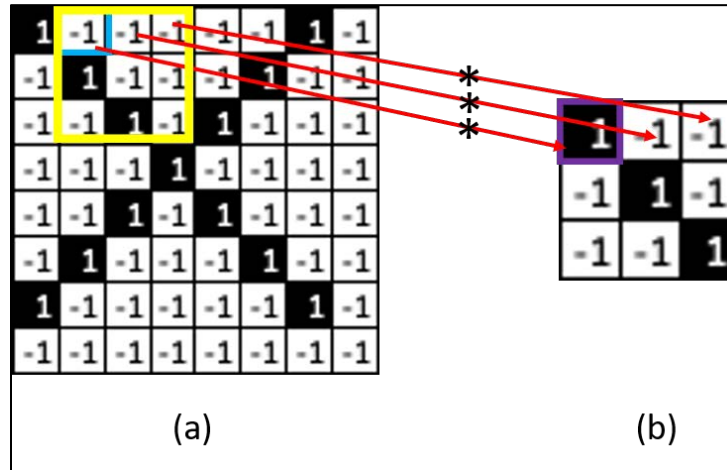


Figure 3.11 Filtering process with one of the features.

The yellow frame sweeps into the entire picture and multiplies the values in the yellow frame in the image cells by corresponding feature cells and divided by the number Of cells, as shown below.

$$\frac{(-1 * 1) + (-1 * -1) + (-1 * -1) + \dots + (-1 * 1)}{9} = \frac{-1}{9} = -0.11 \quad (3.13)$$

The green frame sweep all over the pictures, and each time it generates a number based on the algorithm mentioned. These numbers are illustrated in figure 3.12. This process function called convolution since the actual image somehow convoluted with the feature image and called filtering because, as in figure 3.12, the left diagonal arms of the X character filtered in figure 3.12 in dark green color. The result in figure 3.12 shows which cell of the image has the highest probability estimation to matched with the selected feature.

1	-0.11	0.11	0.33	0.55	-0.11	0.33
-0.11	1	-0.11	0.33	-0.11	0.11	-0.11
0.11	-0.11	1	-0.33	0.11	-0.11	0.55
0.33	0.33	-0.33	0.55	-0.33	0.33	0.33
0.55	-0.11	0.11	-0.33	1	-0.11	0.11
-0.11	0.11	-0.11	0.33	-0.11	1	-0.11
0.33	-0.11	0.55	0.33	0.11	-0.11	1

Figure 3.12 Results of the filtering for the left diagonal arm of the X.

Figure 3.13 shows the above process applied to the image for all three features. The “©” icon represents the convolutional operator. Figure 3.13 (b), showing the filtered pattern of each feature. This is what the convolutional layers do. It takes the set of features, three features in our case, and then returns a set of filtered images.

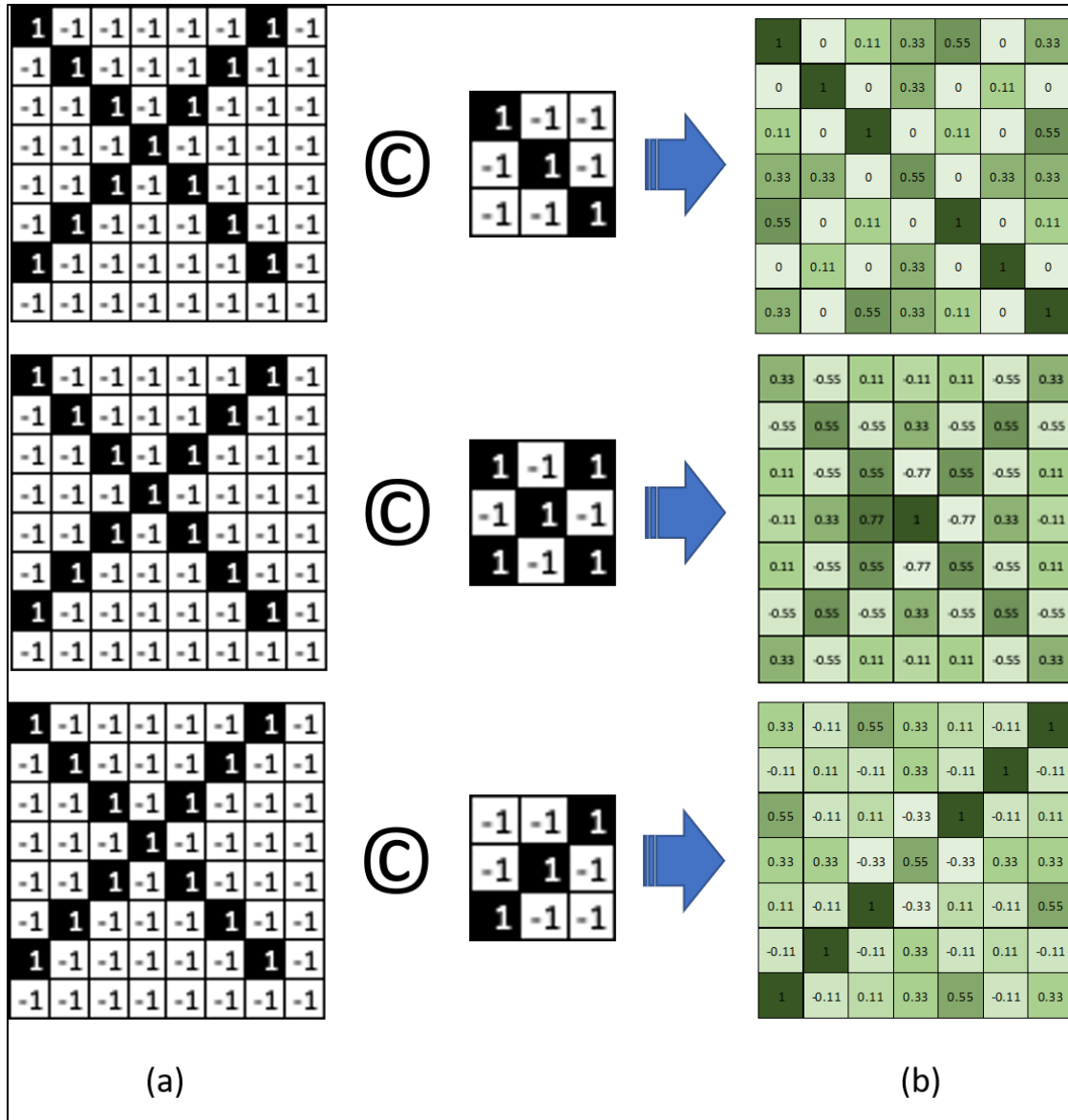


Figure 3.13 Convolutional operation illustration.

The data in each cell should normalize by changing negative ones to zeros to making network training an easier optimization problem. One of the activation functions mentioned in figure 3.4 is called Rectified Linear Unit or “ReLU.” The main advantage of using the ReLU function over other activation functions is that it does not activate all the neurons at the same time. This means that the neurons will only be activated if the output of the linear transformation is more than 0. Equation 3.14 shows the function equation, and figure 3.14 shows the corresponding graphs.

$$f(x) = \begin{cases} 0, & x < 0 \\ x, & x \geq 0 \end{cases} \quad (3.14)$$

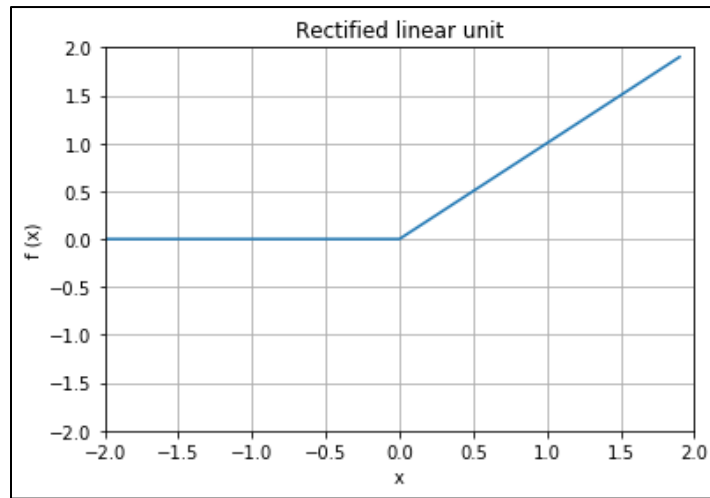


Figure 3.14 Rectified linear unit graph.

The output of the image after applying the ReLu function is shown in figure 3.15 (b) which all the negative vlues became zero.

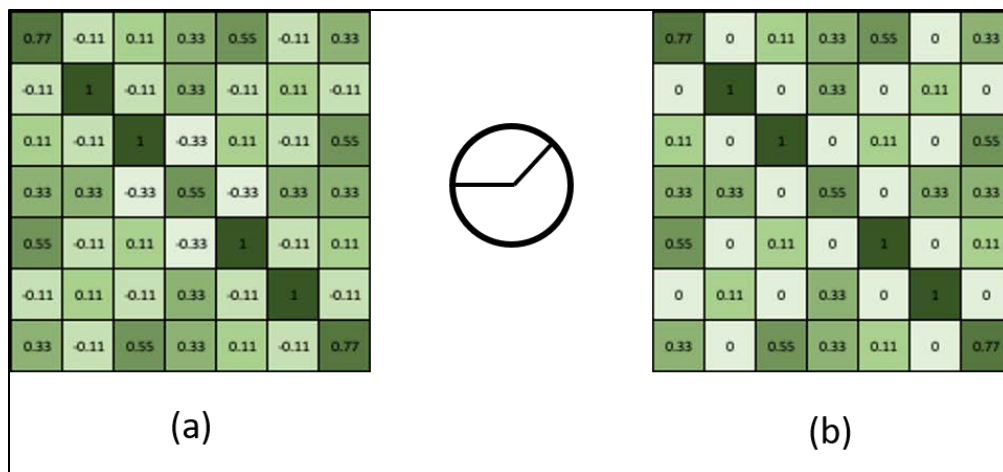


Figure 3.15 (a) Convolved matrix, (b) ReLu function applied to the convolved matrix.

The next operation called the MaxPolling. MaxPolling shrinks the size of the image matrix without losing its essential information. For MaxPolling operation, a window size (usually 2 x 2 or 3 x 3) and the step size (usually 2) will be selected to sweep across the convoluted images. Each

iteration should return the maximum value within the 2 x2 frames. Figure 3.16 illustrates the convoluted image of the left arm of the X character. The first red window is taking the first four-member of filtered images and put the maximum value.

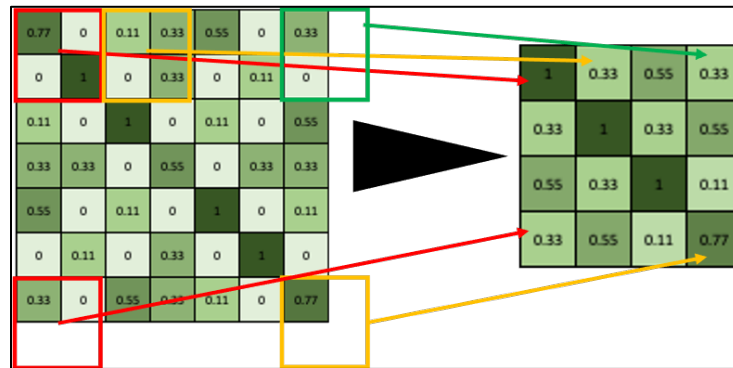


Figure 3.16 MaxPolling the left arm of X character.

Figure 3.17 shows all layers, convoluted, ReLu(ed), and MaxPool(ed) image for each different feature. These layers together called hidden layers. Figure 3.5 shows the perceptron with only one hidden layer. Although the hidden layer in figure 3.5 is slightly different than the one in figure 3.16, in terms of the functionality, in deep learning any layer between input and output layers called hidden layers. In hidden layers, artificial neurons take in a set of weighted inputs and produce an output through an activation function. In hidden layers of CNN, each matrix element in the convolution filter is the weight that is being trained.

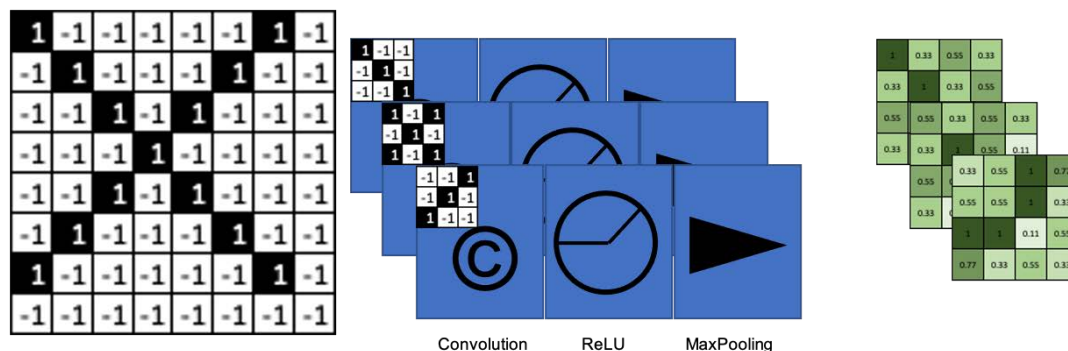


Figure 3.17 All three features passed to one simple hidden layer.

The neural network could have one or more hidden layers depending on the application. Each hidden layer can have a different arrangement in terms of the Convolutional filter, MaxPooling, or ReLU layers. Figure 3.18 shows two different arrangement of different modules in the hidden layer.

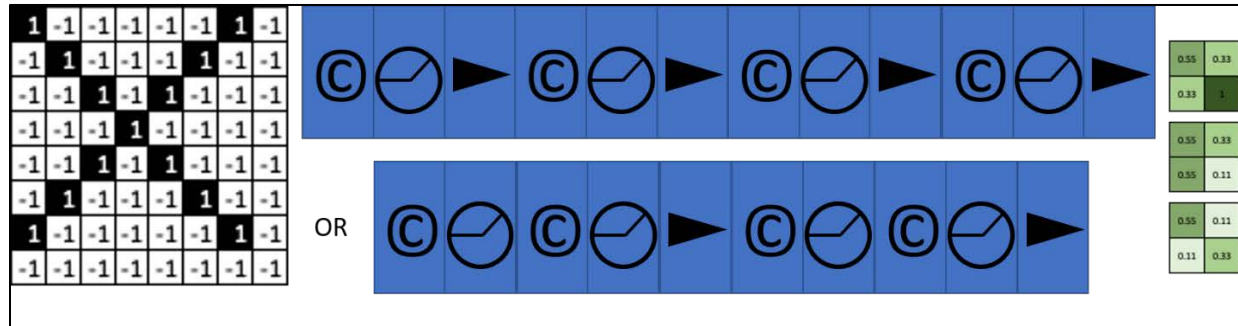


Figure 3.18 Connecting few hidden layers.

After these steps, all the output from hidden layers matrices expands to one linear layer called a fully connected layer. Figure 3-19 shows the fully connected layers constructed by the output of hidden layers shown in figure 3.18.

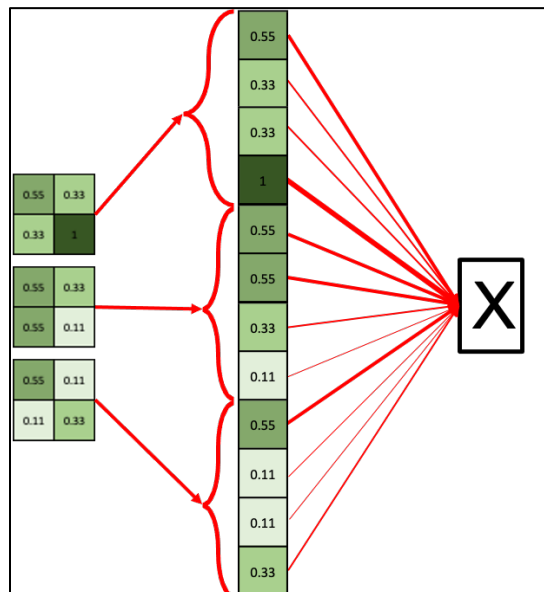


Figure 3.19 Fully connected layer.

Figure 3.20 shows the fully connected layer obtained from figure 3.19, feed as the input to simple ANN shown in figure 3.3.

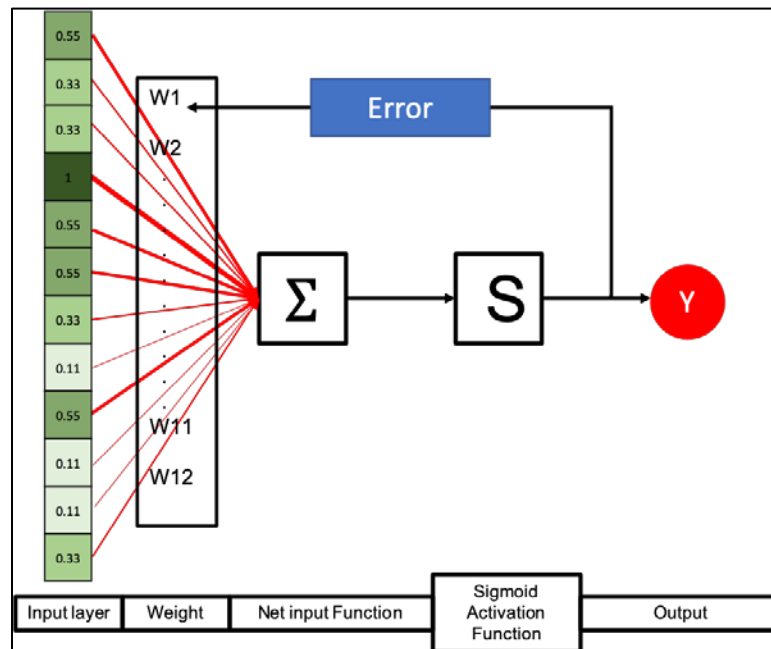


Figure 3.20 Fully connected layer feeds the input of simple ANN.

Since the character is classified by this layer, it called classification layers. As stated in the previous section, the data in an input layer multiplied by an arbitrary weight and summed. Then the Sigmoid function applied to the summed value. The output of the Sigmoid function compares to the highest possible value, which is one. After that, all the weights change, and multiplication and summation repeat. In each iteration, the error calculated by subtracting the output of the Sigmoid function from the expected value. This process repeated until optimal error achieved. When the minimum error achieved, all the weights are stored. For the simplicity of this chapter, three features of the only one kind of font picture were chosen. For having the network to able detect any X character with any font, more X character images with different font should feed to the network. These different X characters called dataset. The bigger dataset means a better training process, and a better training process means a more accurate network.

3.3 Room Categorization with Convolutional Neural Network

Section 3.2 explained how Convolutional Neural Network train to detect the simple X character out of other alphabet characters. As stated before, the data set is used to train a Neural Network. For instance, if the Convolutional Neural Network supposes to detect the apple image out of other fruits, the data set of all fruits, including apple images, should be used to train the network. In this research, CNN will use to detect a room category out of 6 different room categories. The data set used in this research is obtained from work done in [26] MATLAB's Deep Learning toolbox is utilized for training and classifying the room categories. After training the network with the dataset, the images of the maps obtained by Gmapping, as implemented in chapter 4, are classified using CNN to indicate the applicability of the approach for room categorization. The implementation of the CNN algorithm mentioned in Section 3.2 handled by MATLAB Deep Learning toolbox. In the next two chapters, the Gmapping implementation and result will be discussed, followed by chapter 5, which elaborates on the simulation for room categorization result.

4. SLAM IMPLEMENTATION

Gmapping is the ROS package that implements SLAM using a Rao-Blackwellized particle filters algorithm. In this chapter, the performance of the Gmapping approach solving the SLAM problem is evaluated by simulation and experimentation. For simulation, Gazebo is used to construct the environment and navigate the robot within that simulated environment. Gmapping is used to generate the map using LiDAR information. Afterward, the estimated map and the actual map are compared to evaluate performance. The second part of this chapter implements the Gmapping approach in the real environment with an actual robot. In the experiment performed, a Jackal robot is moved in two different campus's rooms to map the environments. Another environment mapped is an apartment with furniture. In both the simulation and experimentation, RVIZ is used to visualize the environments.

4.1 Simulations

The Gazebo is a 3D simulator software that has the ability to simulate the robot in indoor and outdoor environments. The Gazebo has a precise physic simulator similar to game engines. In Gazebo, any desired environment or robot can construct in the simulator with any features and abilities. For instance, the environment can be designed to have different gravity than earth gravity. The details on how to construct the environment and the robot are not provided here. More details can be found in Appendix B. Two different environments are considered, as shown in figure 4.1.

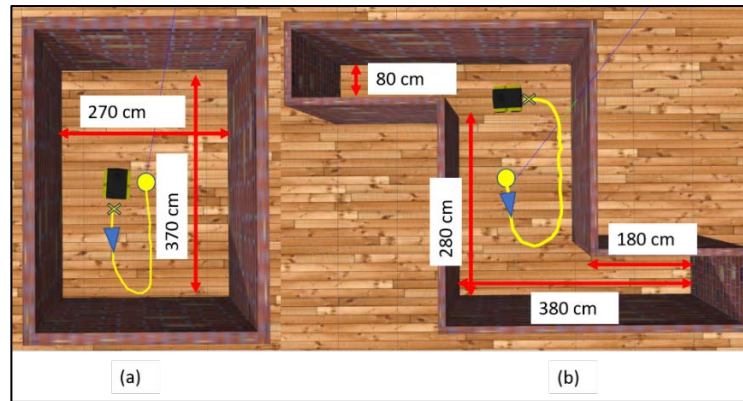


Figure 4.1 Gazebo simulation of rectangle rooms, yellow color shows the trajectory of the robot
(a) simple room (b). corridor-like room.

The first environment is a rectangular shaped room with a dimension of 270 x 370 centimeters, and the other one is the simple corridor shape environment with dimensions of 80 x 180 x 280 x 380. The reason for choosing two different environments is to show the robustness of the approach. In this simulation Jackal, robot features, and all its capability, simulated and defined in the XML file called Jackal description. This file content of all the physics laws and mechanical laws of the robot, speed, weight, LiDAR specification, etc. The second step is to make the desired environments by defining the dimensions and materials or even the obstacle inside the environments. The environment constructed in Gazebo called the world. For simplicity of this simulation, the world constructed in Gazebo does not have any obstacle or any shape complexity, as shown in figure 4.1. After constructing the desired world, in a Linux terminal, the gazebo engine calls the world and robot within it, as shown in figure 4.1.

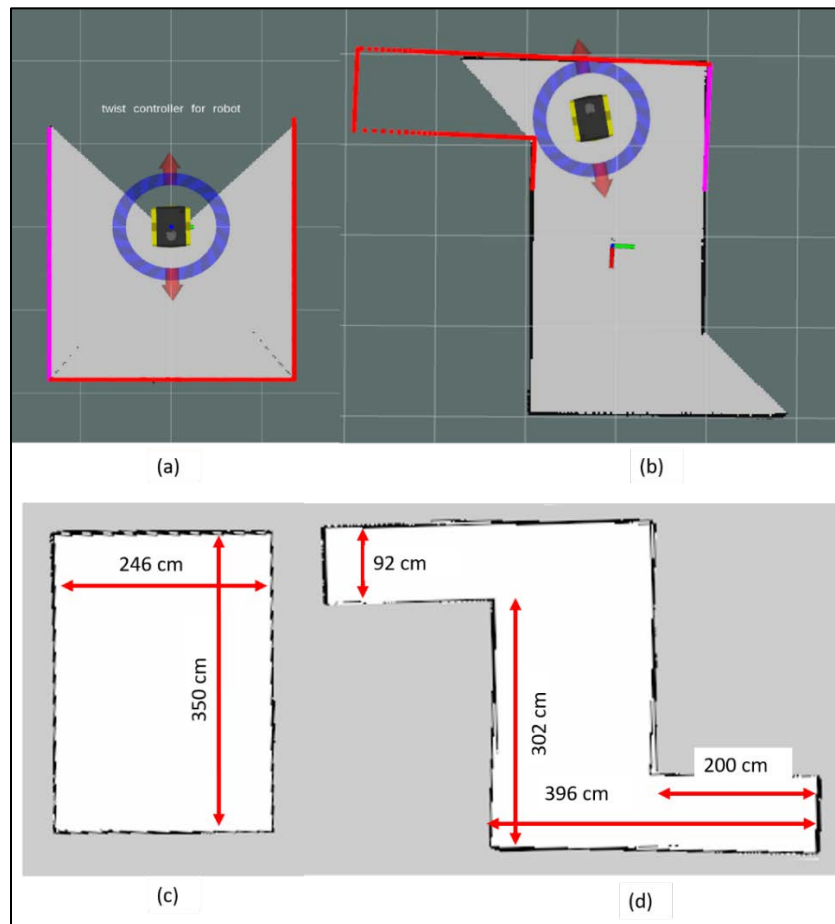


Figure 4.2 (a) Simple room in RVIZ and (c) results of Gmapping in the PGM file. (b) corridor-like room in RVIZ and (d) results of Gmapping in PGM file.

In another Linux terminal, the Gmapping ROS package should be run. For visualization and also moving the robot in the Gazebo world, the RVIZ can be run in the third Linux terminal. The Gmapping ROS package reads the information from the simulated Jackal robot and uses it to build a map. These data include laser scanning data and odometry data, which obtained from simulated Jackal in the simulated Gazebo world. RVIZ is used to navigate the robot and visualizing the constructed map. Figure 4.2 (a,b) shows the graphical presentation of the simulated world in Figure 4.1 (a,b), and 4.2 (c,d) shows the map build from figure 4.1(a,b). The blue wheel around the robot in 4.2 (a,b) is the tool used for moving the robot around. The Red and pink colors in figure 4.2(a,b) are the LiDAR beam representation and the gray color area, representing the portion of the map that already built. As shown in figure 4.2(a,b), the map is not built completely, because the robot is not visited the entire environment. As described in chapter 2, the Gmapping makes the map based on grid mapping. When the LiDAR sees the obstacle, it makes the black color on a gray background, and when the LiDAR doesn't see any obstacle, it generates the white color. Gmapping creates two files as an output. One file is consists of the graphical information in PGM format, shown in figure 4.2(c,d), and the other one is consists of the image information in YAML format, which is some information in human-readable form as shown in figure 4.3. A PGM file format, which stands for Portable Gray Map, is a grayscale image file that Gmapping created.

```
image: mymap2.pgm
resolution: 0.020000
origin: [-10.000000, -10.000000, 0.000000]
negate: 0
occupied_thresh: 0.65
free_thresh: 0.196
```

Figure 4.3 YAML content

As shown in Figure 4.3, the YAML file contains the following information provided by the Gmapping software package.

- **Image:** path to the image file mymap2.PGM containing the occupancy data.
- **Resolution:** resolution of the map in meters/pixel

- **Origin:** 2-D pose of the lower-left pixel in the map, as (x, y, yaw), with yaw as counterclockwise rotation (yaw=0 means no rotation).
- **Negate:** whether the white/black free/occupied semantics should be reversed (interpretation of thresholds is unaffected)
- **Occupied_thresh:** pixels with occupancy probability greater than this threshold are considered completely occupied. This parameter pre-defined by the Gmapping package.
- **Free_thresh:** pixels with occupancy probability less than this threshold are considered entirely free. This parameter pre-defined by the Gmapping package.

As shown in figure 4.3, the resolution of the map is 2 cm /pixel. Table 4.1 shows the actual versus the estimated dimensions. The first column in table 4.1 indicates the actual dimensions of each room, which is defined at the time of constructing the environment in Gazebo. The second column is the value extracted from PGM images by counting the pixels for each side of the image. For instance, in figure 4.4, one of the room sides magnified to shows the tiny pixels of the images. Those pixels are counted to obtain the dimensions of the corresponding side.

Table 4.1 Comparison between the actual and estimated dimensions derived from the map constructed using Gmapping

	Actual Output	PGM Output	Estimated dimensions from the map with 2 cm/pixel resolution	Gmapping Error %
Simple Rectangle room	270 cm	123 pixel	123 X 2 cm =246 cm	15
	370 cm	175 pixel	175 X 2 cm= 350 cm	11
corridor like room	380 cm	198 pixel	198 X 2 cm = 396 cm	8
	280 cm	151 pixel	151 X 2 cm = 302 cm	7
	180 cm	100 pixel	100 X 2 cm = 200 cm	5
	80 cm	46 pixel	46 X 2 cm = 92 cm	4

For As an example, in figure 4.4, there are 46 pixels. According to the YAML file, each pixel represents 2 cm, which results in an estimated dimension of 92 cm. The third and fourth columns show the conversion between the pixels and the estimated dimension in centimeters. The errors between the actual measurement and estimated dimensions are shown in the last column. All the dimensions are extracted from the Gmapping.

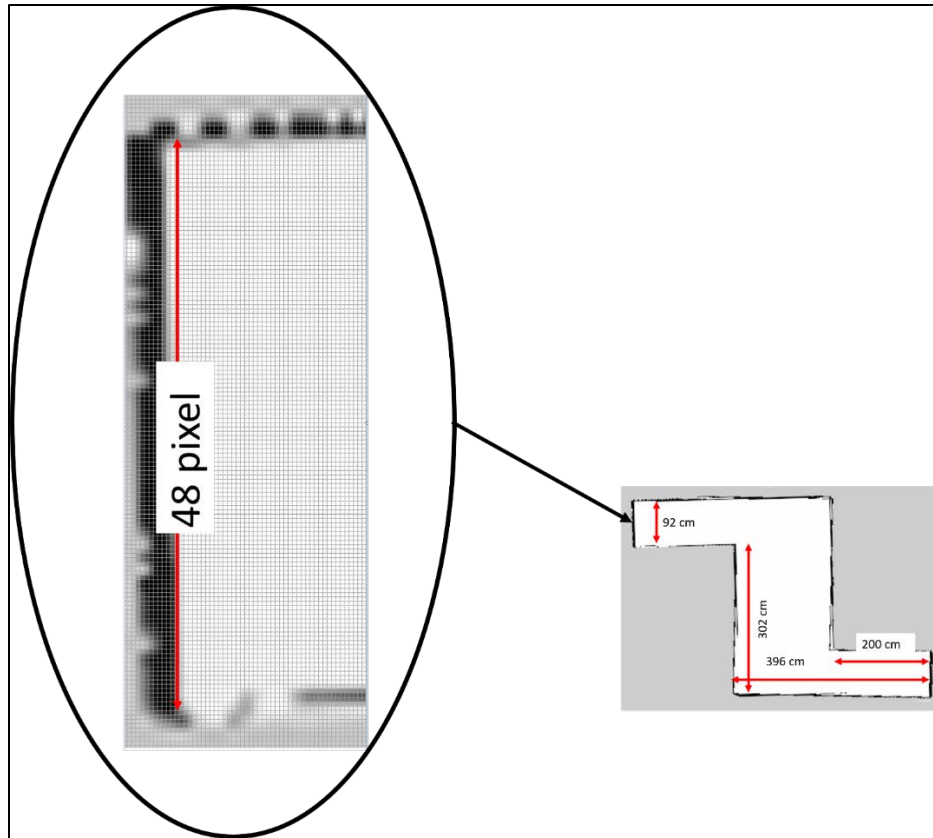


Figure 4.4 A magnified image of the left side of a corridor-like room that shows the tiny pixels.

The percentage of errors versus dimensions are shown in Figure 4.5. As shown, the accuracy of the map is acceptable.

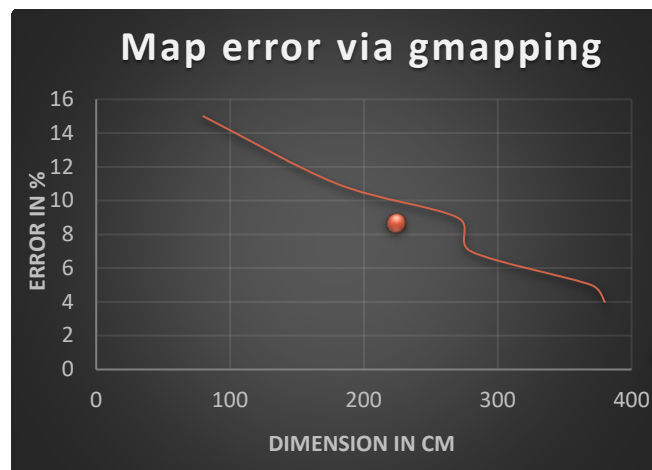


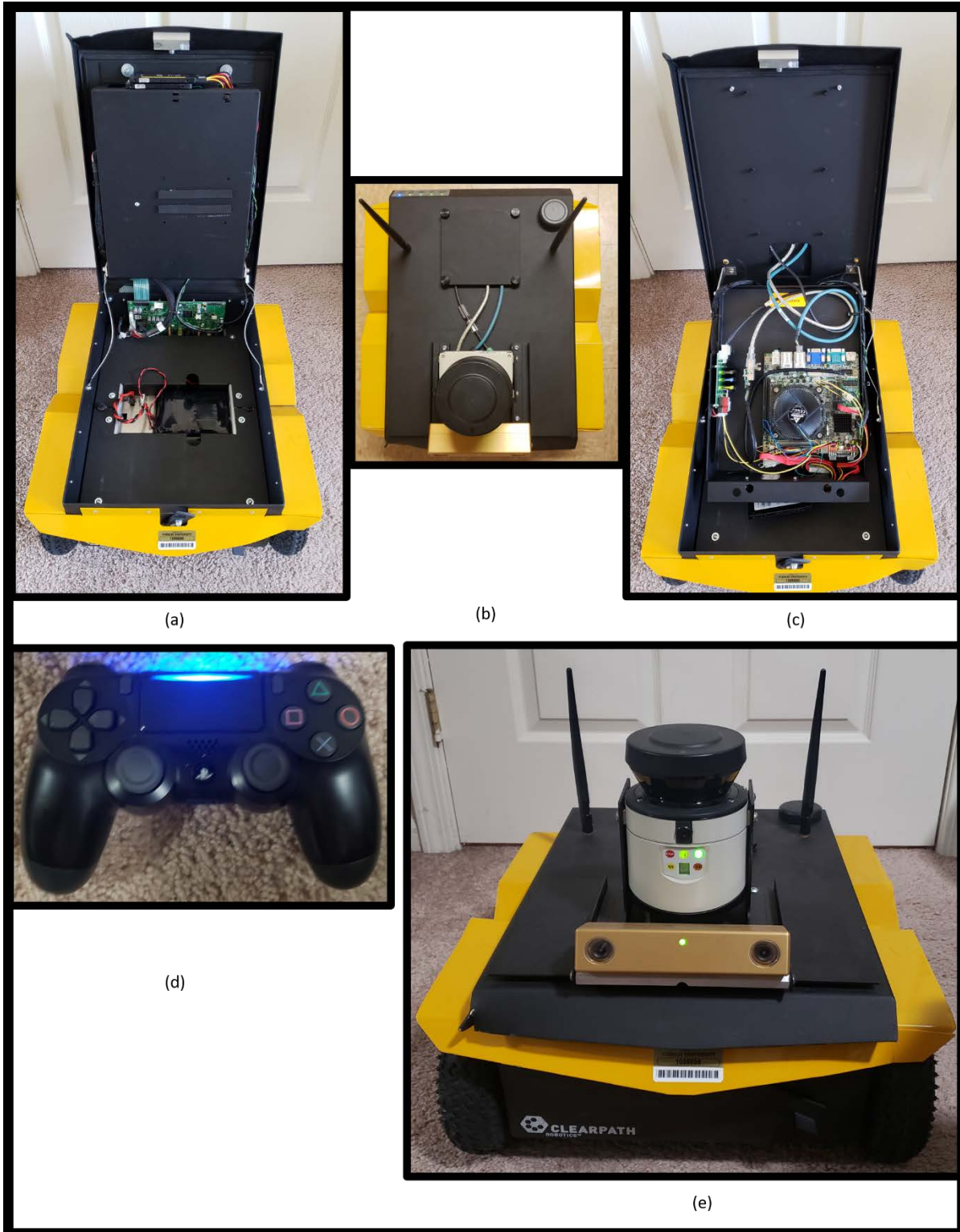
Figure 4.5 Map errors vs. actual dimensions.

4.2 Experimentations

For implementing the Gmapping algorithm to perform SLAM, the Jackal robot equipped with the LiDAR and IMU sensor is used. Jackal robot specifications are provided in Appendix A. The robot operating system(ROS) provides the software platform as explained in Appendix B. The Kinetic version of ROS is installed in the robot processor. The environments considered are three different rooms in different buildings. Two rooms are on Purdue Northwest campus, and the other one is an apartment. One of the rooms (Potter 104) is a very messy environment since the room is used for different purposes, but the other room (Potter 315) was empty at the time of this experiment. Potter 315 is good for easily measuring the actual dimensions of the room and comparing it to the estimated values derived from the SLAM algorithm to determine the accuracy of the approach. Another environment used is a room with furniture. The map for this environment is later used in the room categorization algorithm in chapter 5. The approach of Gmapping used for SLAM is based on Blackwellized Particle Filters, as was explained in chapter 2. In this section, the approach is implemented by performing a number of experiments. The experimental hardware and software platforms are discussed next, followed by SLAM results for different environments.

4.2.1 Hardware Setup

As was mentioned, the Jackal robot specification and capabilities are explained in appendix A. The robot is equipped with a LiDAR sensor that provides the distances between the robot and the obstacles as described in chapter 2. Jackal has the IMU sensor, which provides information about the movements of the robot. Jackal can be commanded to move around with a joystick. Figure 4.6 shows different parts of the Jackal robot. Figure 4.6 (a) and (c) show the inside of the robot. In figure 4.6 (a), the microcontroller drives motors, IMU, and other parts that need the low-level software driver, and in figure 4.6 (c), the Jackal computer which is used for all the computational process is shown. Figure 4.6 (b) shows the outside of the robot with WIFI and GPS antennas. WiFi is used for communication between the Robot and the host computer. In this research, GPS was not used. Figure 4.6(e) shows the LiDAR and stereo camera. In this research, only the LiDAR sensor used. Figure 4.6(d) shows the joystick that uses for moving the Jackal robot around utilizing a PS4 controller with Bluetooth capability installed on the Jackal computer.



4.2.2 Software Setup

Robotic Operating System or ROS is the robotic middleware designed on the Linux platform to handle specifically for developing robotic applications. ROS provides services such as motor and sensor drivers, intercommunication messaging, and etc.[38]. More details in ROS can be found in Appendix B. RVIZ is the graphical user interface in ROS for visualizing the robot data such as sensor output, odometry information, etc. RVIZ can be configured to a specific sensor or actuator. In this research, RVIZ is set up to show the LiDAR measurements and visualize the map generated by the Gmapping approach in SLAM. In order to run the SLAM, two different programs are needed, one is the Gmapping app, and the other one is the RVIZ for visualization of the estimated map from Gmapping. The constructed map using Gmapping is explained in the next section.

4.2.3 Robot Navigated in a Research Lab

In this experiment, the objective is to map a laboratory room, Potter 104 using Gmapping. Figure 4.7 shows the map generated. The lab is so messy due to equipment for many projects. Because of this issue, it's very hard to find the dimension of the room since some of the places in the room are unreachable. But still, the experiment give us some information about the capability of Gmapping in making a map. For example, the red circle in figure 4.7, shows the door while it's open, or the orange circle shows three big trash bins in the corridor behind the lab. The robot trajectory is shown with a blue color.

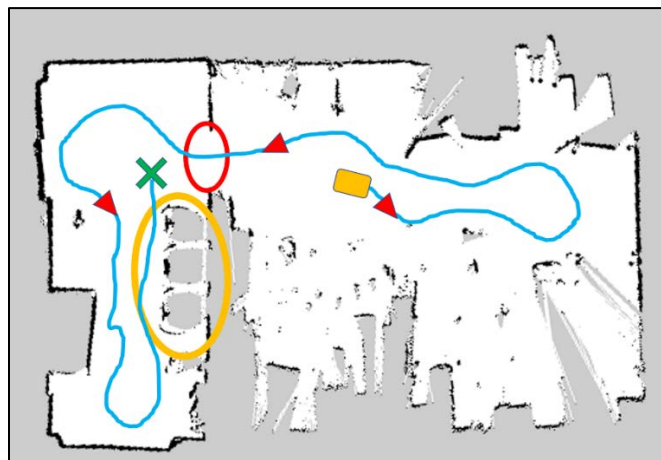


Figure 4.7 Potter 104 maps generated by Gmapping.

The yellow rectangular indicates the starting position of the robot, and the Green cross marked the end position. The robot movement controlled by a human operator using the joystick. At the same time, the host computer established a connection via wifi with the Jackal computer. The Gmapping algorithm is executed by a robot computer and, at the same time, the RVIZ is used to visualize the map constructed with Gmapping in the host computer. After the robot reaches the end position, the map constructed by Gmapping and visualized by RVIZ is saved as shown in figure 4.7. For evaluating the accuracy of the map, the next experiment uses an empty room as dimensions can easily be measured.

4.2.4 Robot Navigated in an Empty Room

As mentioned before, an empty room on campus with a known dimension is used in this experiment. The map generated by the Gmapping and visualizing with RVIZ, shown in Figure 4.8.

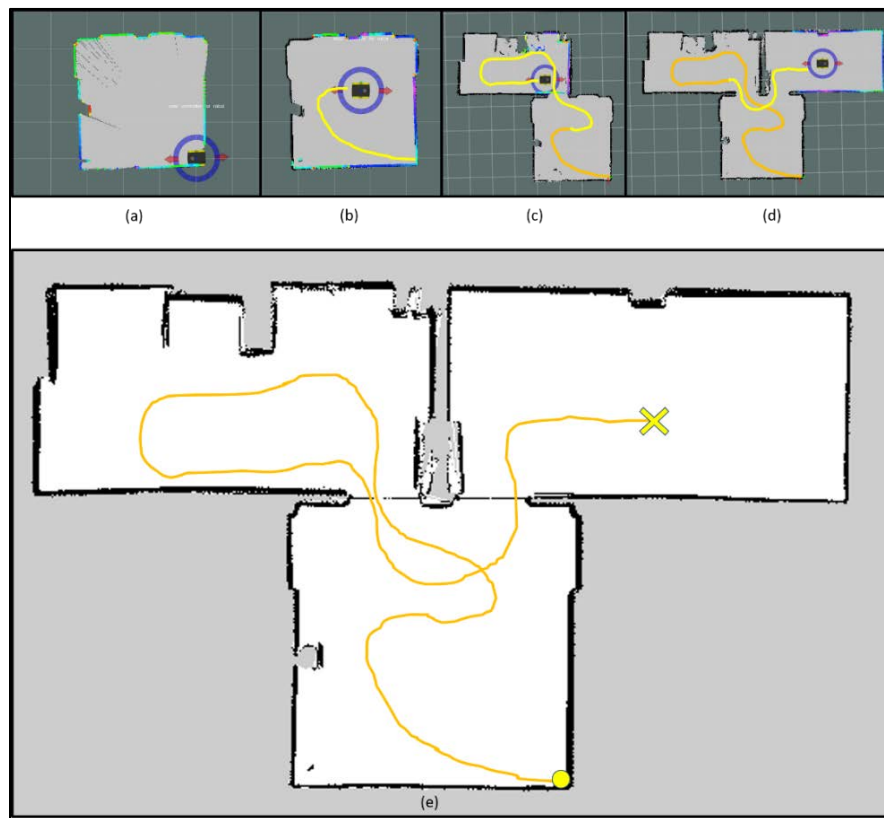


Figure 4.8 Potter 315 maps generated by Gmapping, (a – d) in each stage that map constructed by Gmapping ROS package, RVIZ visualization shown. (e) the PGM file that created by the end of the Gmapping process.

Figure 4.8 shows steps that the robot took to complete the map. In figure 4.8-(a), the robot started from the right corner of the room. When the Gmapping ROS package executed, the location of the robot is considered the origin point of the map. The trajectory of the robot in the current constructed map is shown with yellow color. Figure 4.8 (b), (c), and (d) show different stages of constructing the map in RVIZ using Gmapping. In this experiment, the robot connects to the host computer via wifi. The RVIZ runs in a host computer, and the Gmapping ROS package runs on a Jackal computer. Each piece of the map constructed by Gmapping sends via wifi to RVIZ for visualization in the host computer. When all the environment mapped, figure 4.8 (d), the map is saved in the host computer by calling the ROS node “map_server” which explained in Appendix B. The estimated dimensions are calculated by counting the pixels in the map image like the process described in Section 4.1. The actual room dimensions measured by the ruler for comparison. Figure 4.9 shows the map with the actual and estimated measurements.

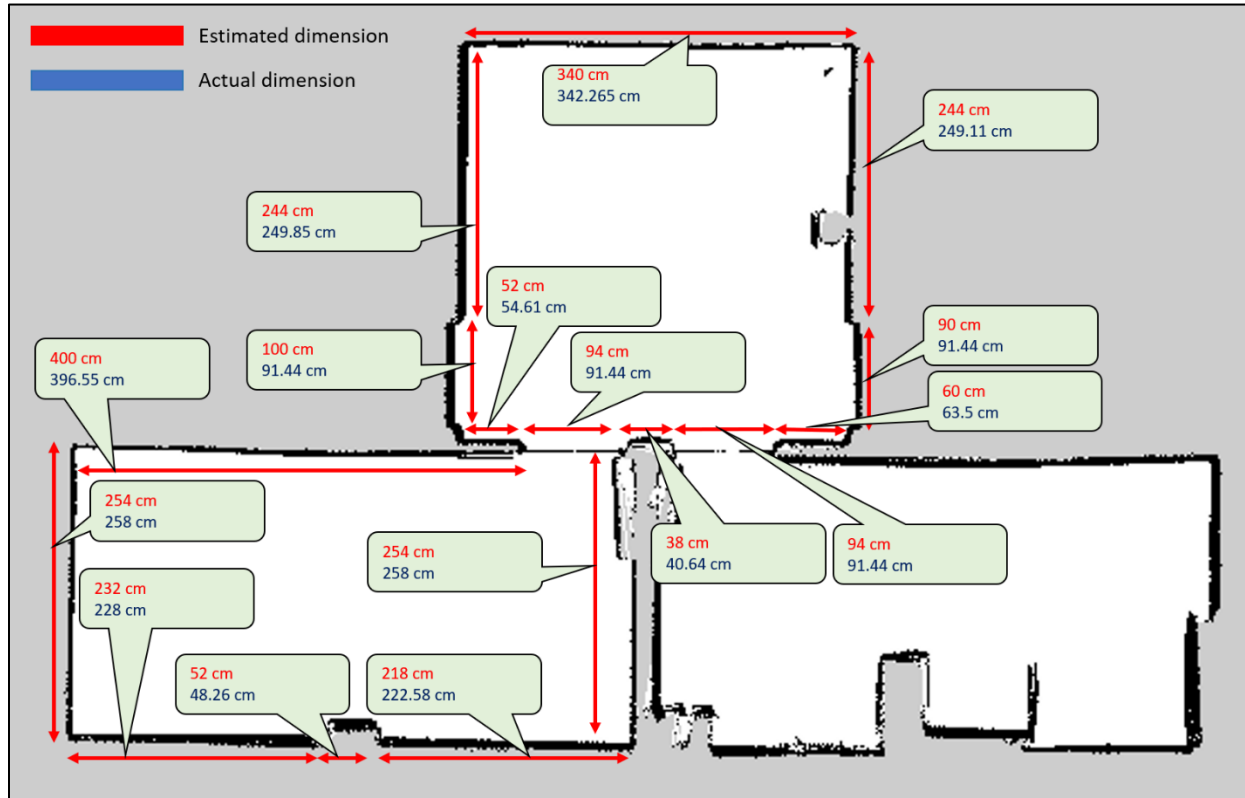


Figure 4.9 Potter 315 with actual and estimated dimensions. Blue color is Estimated size from Gmapping, and orange color indicates the actual length.

Table 4.2 Comparison between the actual and estimated length

Actual Dimension (cm)	PGM Output Pixel	Estimated dimensions from the map with 2 cm/pixel resolution	Gmapping Error %
40.64	19	19 X 2 cm=38 cm	7
48.26	26	26 X 2 cm=52 cm	8
54.61	26	26 X 2 cm=52 cm	5
63.5	30	30 X 2 cm=60 cm	6
91.44	48	48 X 2 cm=96 cm	5
91.44	47	47 X 2 cm=94 cm	3
91.44	47	47 X 2 cm=94 cm	3
91.44	45	45 X 2 cm=90 cm	2
222.58	109	109 X 2 cm=218 cm	3
228	116	116 X 2 cm=232 cm	2
249.11	122	122 X 2 cm=244 cm	3
249.85	122	122 X 2 cm=244 cm	3
258	127	127 X 2 cm=254 cm	2
258	127	127 X 2 cm=254 cm	2
342.265	170	170 X 2 cm=340 cm	1
396.55	200	200 X 2 cm=400 cm	1

Table 4.2 shows the map error between the actual dimension and the estimated one. The errors are graphed in figure 4.10. As shown, errors are reduced for larger dimensions. This is similar result as was obtained from simulation.

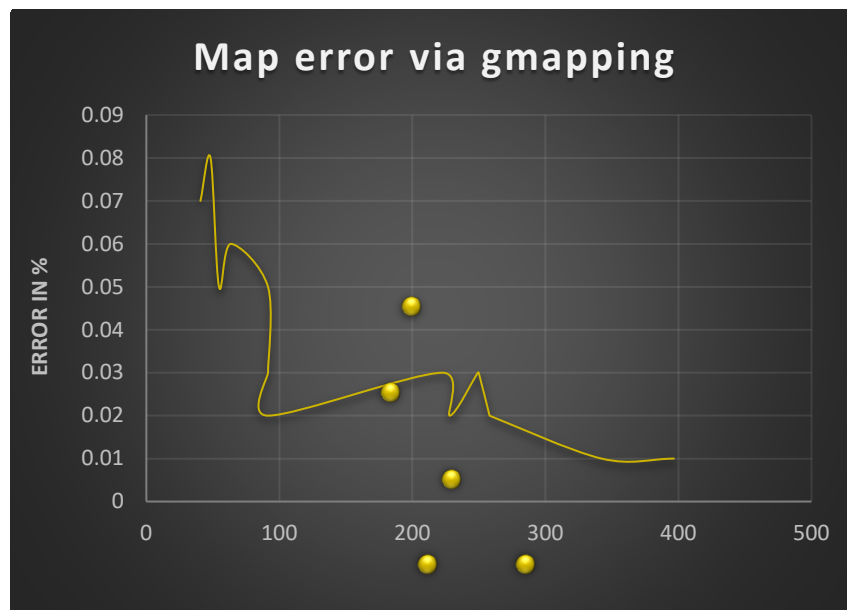


Figure 4.10 Dimension's error in the generated map.

4.2.5 Navigation in an Apartment

So far, two different environments have been considered. In Potter 104, the approach distinguishes between the lab and the corridor behind it. In Potter 315, the accuracy of the approach was evaluated. Both environments considered do not have any furniture. Since the objective is to categorize the rooms in an apartment or a house, in this experiment, an environment with some furniture is considered. The experimental objective is to map a 2-bedroom 2-bathroom apartment with the furniture using the Jackal robot with LiDAR. Figure 4.11(a) shows the layout of the two beds, two baths apartment with each different room shown separately. Figure 4.11(b) and 4.12(c) show both the layout of the apartment and also the generated PGM file for comparison between the actual and estimated map. As shown in figure 4.11(c), the estimated map is pretty accurate in terms of shape in comparison to actual layout 4.11(b). Red lines in figure 4.11(b) and (c) indicate the boundary of each room. The map constructed in this experiment is used for room categorization, as explained in the next chapter.



Figure 4.11 (a) two beds, two baths apartment layout shown in separate parts. (b) actual layout
(c) estimated map.

5. ROOM CATEGORIZATION SIMULATIONS

The objective of this research is to utilize the map generated by the Gmapping SLAM algorithm using LiDAR for room categorization. The approach used for room categorization is Convolutional Neural Network explained in detail in chapter 3. To evaluate the applicability of the approach, simulations are performed in this chapter. From chapter 4, the 2d map for different rooms was constructed in an experiment performed using Jackal with a LiDAR. The SLAM algorithm is to make the 2D map in the PGM format and use this 2D map to train and classify the room category with CNN. The PGM format is converted to jpeg since the program that performs the CNN can only read the jpeg format. In addition, to improve training, another dataset that was obtained from another research [26] is used in this simulation and our results are compared with the approach in [25] using the same dataset. These datasets collected from 24 different homes, including apartments and houses obtained by a P3-DX robot equipped with Hokuyo URG laser rangefinder. In Table 5.1, the IMU and laser scan data used in [26] to build the map are shown. These data are used to make an image from these raw data with the “MapBuilder_Dataset.m” MATLAB program. Figure 5.1 shows the map created with “MapBuilder_Dataset.m” alongside the map created in [26]. The “MapBuilder_Dataset.m” code is provided in Appendix C.1.

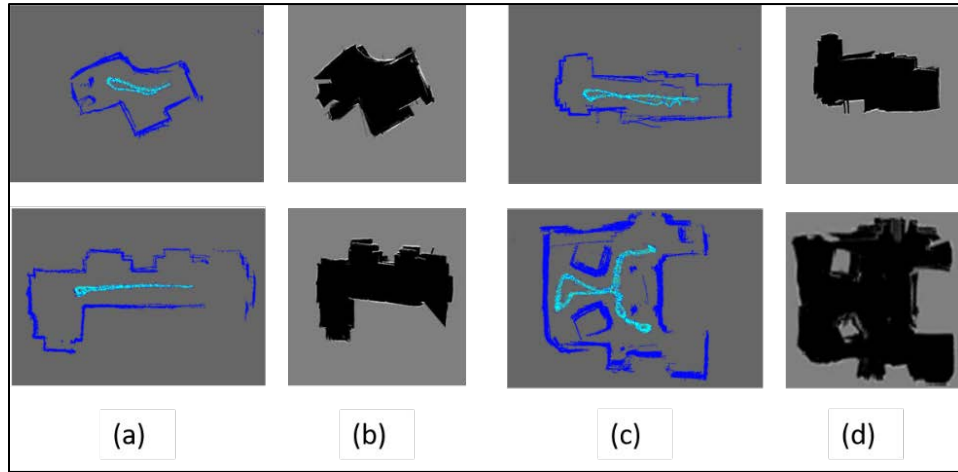


Figure 5.1 Group (a & c) are the [26] output and (b & d) are the output map of our MATLAB.

Table 5.1 Dataset of the raw odometry and LiDAR used[26].

Number	Time Stamp	Room category	Odometry x	Odometry y	Odometry Phi	Laser range finder start angle	Angel step	Number of beams	Min Range	Max range	Range reading	2	3	681	682
1	2.266699	5	0	0	0	-2.08621	0.00613592	682	0.01	5.6	0.814	0.814	0.814	1.166	1.162
2	2.357944	5	0	0	6.283	-2.08621	0.00613592	682	0.01	5.6	0.821	0.817	0.815	1.171	1.166
3	2.456573	5	0.002	0	0	-2.08621	0.00613592	682	0.01	5.6	0.821	0.821	0.818	1.165	1.164
4	2.556433	5	0.007	0	6.283	-2.08621	0.00613592	682	0.01	5.6	0.821	0.82	0.818	1.173	1.166
5	2.656459	5	0.013	-0.001	6.266	-2.08621	0.00613592	682	0.01	5.6	0.825	0.82	0.813	1.181	1.178
6	2.756388	5	0.023	-0.001	6.248	-2.08621	0.00613592	682	0.01	5.6	0.826	0.827	0.827	1.19	1.188
7	2.856478	5	0.036	-0.001	6.231	-2.08621	0.00613592	682	0.01	5.6	0.836	0.831	0.828	1.211	1.211
8	2.956438	5	0.052	-0.002	6.213	-2.08621	0.00613592	682	0.01	5.6	0.844	0.844	0.844	1.238	1.238
9	3.056482	5	0.067	-0.003	6.196	-2.08621	0.00613592	682	0.01	5.6	0.858	0.856	0.858	1.262	1.265
10	3.156477	5	0.083	-0.005	6.161	-2.08621	0.00613592	682	0.01	5.6	0.862	0.863	0.863	1.099	1.117
11	3.256413	5	0.097	-0.007	6.144	-2.08621	0.00613592	682	0.01	5.6	0.875	0.875	0.875	1.051	1.068
12	3.356408	5	0.109	-0.008	6.109	-2.08621	0.00613592	682	0.01	5.6	0.878	0.878	0.878	1.005	1.017
473	49.896594	5	-0.936	-0.006	0.698	-2.08621	0.00613592	682	0.01	5.6	0.352	0.352	0.355	0.383	0.387
474	49.995908	5	-0.936	-0.006	0.698	-2.08621	0.00613592	682	0.01	5.6	0.349	0.353	0.353	0.385	0.386

5.1 Simulation Setup

For simulating the room categorization, MATLAB, with “Deep Learning” toolbox is used. Deep Learning Toolbox™ (formerly Neural Network Toolbox™) provides a framework for designing and implementing deep neural networks with algorithms, pre-trained models, and apps. The convolutional neural networks (ConvNets, CNNs) is used to perform classification and regression on image, time-series, and text data. In chapter 3, the roles of different parameters in the neural network were discussed. Below are the settings that configure the simulation parameters. As is described in chapter 3, CNN has different layers. Figure 5.2 shows the sequence of the layers used. There are three hidden layers followed by Softmax and classification layer. The descriptions of each layer is given below

- Input layer: This is the layer that read all the images, the size of all images which feed into the CNN should be normalized and be the same. The image size normalization, performed in the same program which performs CNN. In this research, 227 x 227-pixel size was chosen since it is one of the standard sizes for the CNN process in MATLAB. Selecting the mentioned image size is not too big to cause memory overflow, and it's not too small to lose features of the image due to low resolution.
- Hidden layers consist of three different layers
 - The convolutional layer, three parameters, needs to define for this layer.
 - Filter size, as is discussed in chapter 3, is the width and height of the filters that the training function uses while scanning along with the images. In this research, the filter size is 3-by-3. The reason to chose the 3 by 3 is that the single neuron can be left, right, upper, down, upper left, upper right, lower left, lower right, as a total of 8 neighbor information, and usually it is the smallest filter size.
 - The number of features map: This parameter defines the number of features, 32 features selected for this experiment. Each feature map is being trained to capture a particular aspect of the input. The number of feature maps is a function of how many different features need to be captured and understood by CNN to produce the desired output. This parameter of the CNN, tested by 8, 16, 32, and 64. The best result obtained when the number of features map is 32.

- Stride number: This is the number of steps, each time the filter frame should move while sweeping the image. The stride number is one.
- Normalization layer: as discussed in chapter 3, the normalization layer is just used for normalizing the negative numbers to speed up the training process.
- MaxPooling layer: As stated in chapter 3, MaxPooling reduced the image size of the feature image by omitting the redundant spatial information to reduce the computational cost.
 - The size of the MaxPooling frame is 2 by 2
 - The stride of the MaxPooling frame is 1
- Fully connected layers: Max-pooling layers are followed by one or more fully connected layers. As described in chapter 3, the fully connected layer is the flattened version of the output of the hidden layer. This layer combines all the features learned by the previous layers across the image to identify the larger patterns. The last fully-connected layer combines the features to classify the images. Therefore, the output size parameter in the last fully connected layer is equal to the number of classes in the target data. In this example, the output size is six, corresponding to the six classes.
- Softmax Layer; The softmax activation function normalizes the output of the fully connected layer. The output of the layer consists of positive numbers that sum to one, which can then be used as classification probabilities by the classification layer.
- Classification Layer: The final layer is the classification layer. This layer uses the probabilities returned by the softmax activation function for each input to assign to one of the mutually exclusive classes.

The next section describes the training process and how to prepare the data set for this training.

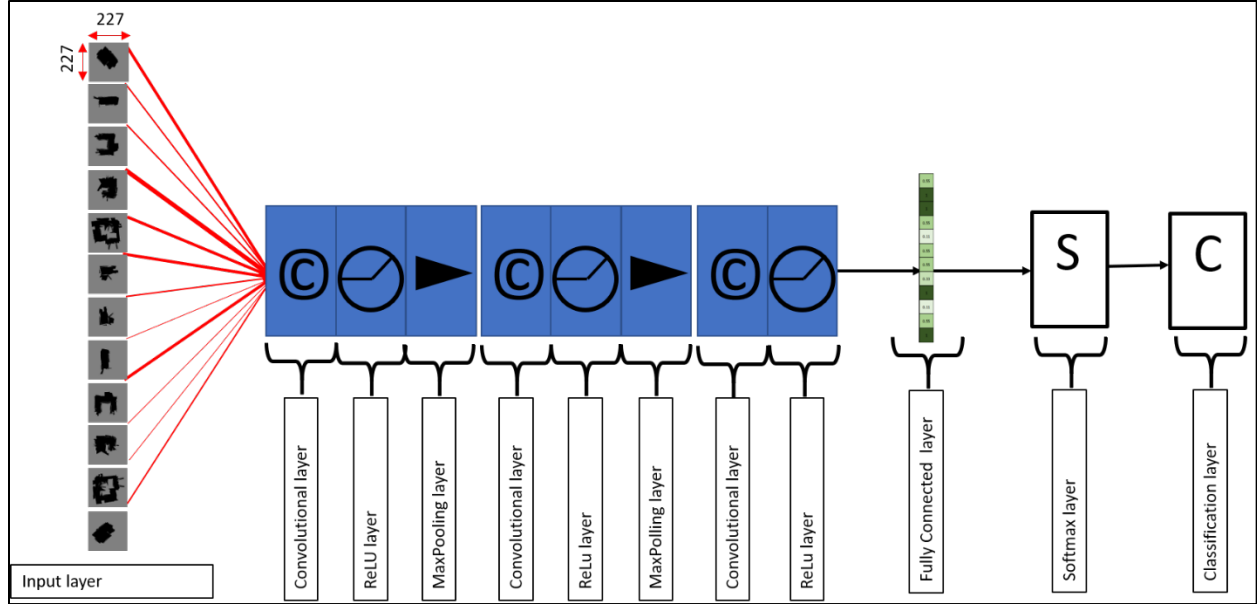


Figure 5.2 Layers construction.

5.2 Training the Network

Train the network using the CNN architecture defined by layers and the training data. For this training, the program uses a GPU if one is available. Otherwise, it uses a CPU. The parameters discussed in the previous section, prepare the MATLAB program which implementing the program that performs CNN. The program in MATLAB called “RoomCategorization.m” can be found in Appendix C.2. As stated before, the raw data from [26] is used to make the 2D image of the map and normalize it into the 227 x 227-pixel images. The different rooms in each category and a few samples of images in each category built are shown in figure 5.3.







		Bathroom	Bedroom	Toilet
Kitchen	21			
Bedroom	28			
Bathroom	35			
Toilet	12			
Corridor	6			
Livingroom	21			
		Corridor	Kitchen	Livingroom
				

Figure 5.3 Number of different rooms and sample of each room.

These images, plus the images obtained in section 4.2.5 mapping an apartment, are feed to the “RoomCategorization.m” MATLAB program for the training the CNN. As discussed in chapter 3, each pixel of the images feeds into the neuron, followed by convoluting layers. The features at the beginning of the training selected randomly, and they will change during the training. ReLU and MaxPooling layers will follow the output of the convoluted layer.

After the neural network trained by these images, the validation process is started. The validation and the result of it will be discussed in the next section.

5.3 Simulation Results

The goal of the validation process is to find the probability of each image classification in the correct category. For instance, at the end of the training, the “RoomCategorization.m” MATLAB program reads every image in different categories and labels them to different classes. As shown in figure 5.4, the “RoomCategorization.m” MATLAB program listed all the room's image in the bedroom category. Then each of them classified by the trained network, and the output is labeled bellow each image. In a perfect circumstance in which the network trained ideally, all of the images in figure 5.4 should be labeled as a bedroom. One of the reasons that the network is not trained ideally is the number of datasets. With the bigger data set, it can train a better neural network.

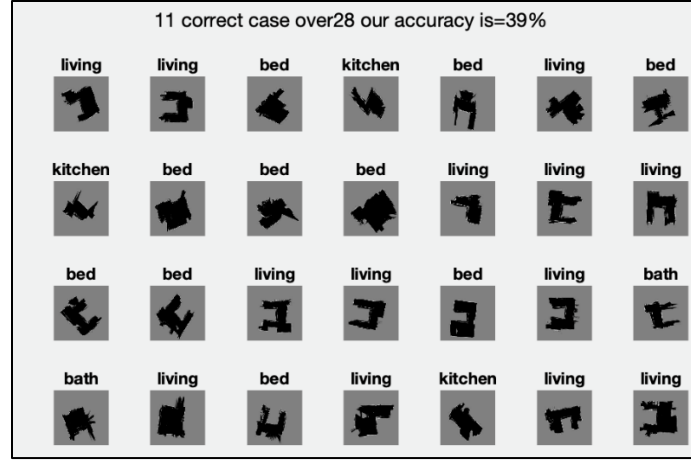


Figure 5.4 Classification results for the bedroom.

Figure 5.4 shows each one of the images in the bedroom category classified with the trained network, and the result shows out of 28 rooms, only 11 of them classified correctly, which means 39% accuracy. The same process had been done, and the probability rate obtained for each. Table 5.2 shows the results for each category.

Table 5.2 Accuracy for each room classification

Category	Probability	Category	Probability
Kitchen	14%	Toilet	92%
Bedroom	52%	Corridor	100%
Bathroom	39%	Livingroom	52%

As shown in Table 5.2, the corridor has the highest accuracy, and the kitchen has the lowest. The reason could come from the corridor's spatial shape, which is the long rectangular room, and it can be detected easier. The results of this work are compared to Peter Ursic's work [20], and the comparison is shown in table 5.3 using the same dataset. Peter Ursic, in [20][25][38], uses the Hierarchical spatial model approach to classify the rooms.

Table 5.3 Comparison between the results of our work with [21]

	Our Work Result	Result of [20]
Kitchen	14%	52%
Bedroom	52%	42%
Bathroom	39%	34%
Toilet	92%	91%
Corridor	100%	83%
Livingroom	81%	52%

As it shows in table 5.3, our approach has a better performance in all rooms, excluding Kitchen. The comparison in table 5.3 shows our approach with CNN is slightly better than the [21]. One of the downsides of the [21] approaches, is the complexity to implement the approach.

Another validation that has been done in this research was to classifies each of the rooms that maps were obtained experimentally in chapter 4. The results are shown in the form of a matrix [40] in table 5.4. The matrix in Table 5.4 shows that the network can only not classify the toilet correctly.

Table 5.4 Confusion matrix for the apartment rooms that obtained in section 4.2.5.

	Bath room	Bed Room	Corridor	Kitchen	Living Room	Toilet
Bath room	0.9851	0	0	0	0	0
Bed Room	0	1	0	0	0	0
Corridor	0.0149	0	0.99	0	0	1
Kitchen	0	0	0.01	1	0.2349	0
Living Room	0	0	0	0	0.7651	0
Toilet	0	0	0	0	0	0

As discussed in chapter 3, the best neural network is the one that trained with a large amount of data set. For instance, there is a pre-trained network called Alexnet [41], which trained by 1.2 million images for classifying 1000 different objects. It means there are 1200 images per object on average. But in our work, the total of the training data set is about 130. One of the

problems of having such a short data set is the difficulty of collecting data. For instance, Peter Ursic [21] collected his data set from 24 different apartments.

One approach for solving this problem will be discussed in the next chapter in the conclusion and future work.

6. CONCLUSIONS AND FUTURE WORK

The goal of this research was to show the ability of the convolutional neural network to classify the room with the map of the environment obtained by SLAM (gmapping algorithm). In chapter four, simulations and experiments validated the map accuracy using Gmapping, and in chapter 5, simulations showed all the layout obtained from the apartment mapped in chapter 4 was classified correctly, excluding the toilet using CNN. One of the applications for developing a room categorization algorithm is for assisted robots to distinguish between different rooms for better service.

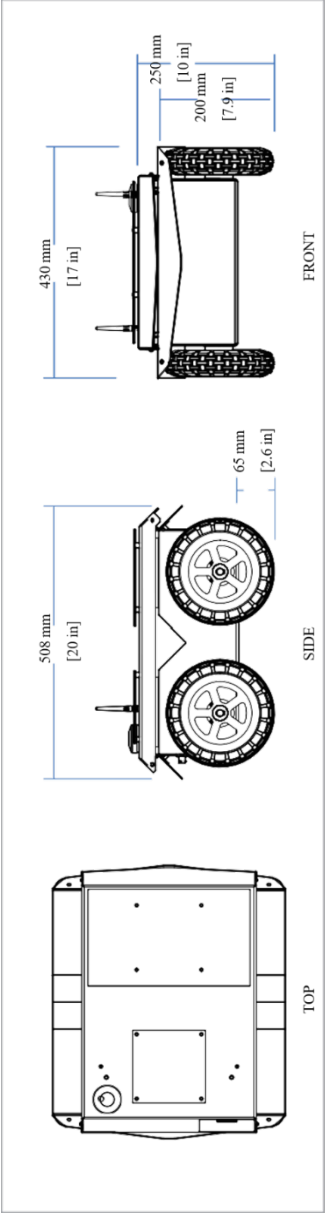
Since one of the big challenges in this work was the dataset, the future work should focus on increasing the amount of training dataset. Since collecting, scanning of the data from home to home is a challenging task, one of the approaches to obtain such a data set is to make a data set in a simulation environment. In chapter 4, simulating the environment in Gazebo is described briefly, and more data sets can be obtained by designing a different layout in Gazebo and add the map inside that simulated environment to the dataset.

Another suggestion is to implement CNN inside the Jackal computer. All the work in chapter 5 had been done on a host computer using MATLAB. In order to have a robot that moved around and classify the rooms during motion, the robot should be able to perform the steps described in chapters 4 and 5. This means; first, the robot should be able to make the map of the environment and localize itself, and then it should detect when the map of the environment is completed (loop closing in SLAM). Afterward, the CNN is trained inside the robot computer and feed the obtained map to the CNN for room categorization in real-time. The room categorization information is then utilized to perform tasks in the home environment by an assisted robot.

APPENDIX A. ROBOT SPECIFICATIONS

This appendix provides the Jackal Robot specification and the sensors used.

- Mechanical



SIZE AND WEIGHT	
EXTERNAL DIMENSIONS (L x W x H)	508 x 430 x 250 mm (20 x 17 x 10 in)
INTERNAL STORAGE DIMENSIONS	250 x 100 x 85 mm (10 x 4 x 3 in)
WEIGHT	17 kg (37 lbs)
GROUND CLEARANCE	65 mm (2.6 in)
SPEED AND PERFORMANCE	
MAX. PAYLOAD	20 kg (44 lbs)
ALL-TERRAIN PAYLOAD	10 kg (22 lbs)
MAX. SPEED	2.0 m/s (6.6 ft/s)
DRIVE POWER	500 W

- Electrical

BATTERY AND POWER SYSTEM	
BATTERY CHEMISTRY	Lithium-Ion
CAPACITY	270 Watt-hours
CHARGE TIME	4 hours
RUN TIME	Heavy usage: 2 hours Basic Usage: 8 hours

- Interface and communication

INTERFACING AND COMMUNICATION	
CONTROL MODES	Kinematic Commands — velocity, angular velocity Open Loop Motor Driver Commands — voltage Wheel Velocity Commands
FEEDBACK	Battery and motor current Integrated GPS receiver Wheel velocity and travel Integrated gyroscope and accelerometer
COMMUNICATION	Ethernet, USB 3.0, RS232. (IEEE 1394 available)
DRIVERS AND APIs	Packaged with ROS Indigo
INTEGRATED ACCESSORIES (included)	Wireless Game controller, GPS, IMU, On-Board Computer, WIFI Adapter, Accessory Mounting Plates

- Computer

COMPUTER	
CPU	Celeron J1800 WIFI Adapter Dual-core, 2.4GHz
HDD	32 GB Hard Drive
RAM	2 GB RAM
Interface	USB 3.0, RS232. (IEEE 1394 available)

- LiDAR Sensor

LiDAR LMS111	
Scan Angle	270°
Scanning frequency	25 Hz ~ 50 Hz
Angular resolution	0.25 ~ 0.5
Distance measuring range (operating range)	0.5 m ~ 20 m
Laser class	Laser class 1 according to IEC 60 8251:2014
Height	162 mm
Width	102 mm
Depth	106 mm
Power supply	10.8 V ~ 30 V
Power consumption Sensor	8 W ~ 10 W

APPENDIX B. ROBOTIC OPERATING SYSTEM SOFTWARE PACKAGES AND SOFTWARE TOOLS USED

1. ROS [42]: The Robot Operating System (ROS) is an open-source framework for developing robot software. It is consisting of tools, libraries, and conventions that aim to simplify the task of creating complex and robust robot behavior across a wide variety of robotic platforms. Some of the tools used for this research are RVIZ and Gazebo. Gmapping is one of the libraries for performing SLAM on the ROS platform. The ROS platform has different versions. The version used in this research is Kinetic. ROS is installed on Linux operating system. The Linux version used in this research is Ubuntu 16.04. For installing ROS Kinetic in Ubuntu 16.04, refer to [43].
2. Gazebo is a 3D simulator, while ROS serves as the interface for the robot. Combining both provides a powerful robot simulator environment. With Gazebo, you are able to create a 3D environment with robots, obstacles, and many other objects. The gazebo also uses a physical engine for illumination, gravity, inertia, etc. for making the world in Gazebo, following steps are performed.

2.1. Install all the requires tools and library for Jackal Robot

```
Sudo apt-get install ros-kinetic-jackal-simulator ros-kinetic-jackal-desktop ros-kinetic-jackal-navigation
```

2.2. load the gazebo and build the floor plan in the layout builder.

```
roslaunch gazebo gazebo
```

2.3. Download the 3d model from 3dwherhouse.com and save the model in the below folder.

Make sure the folder content has <filename.sdf> and <model.config>. If the sdf file is not the same name as the folder, change it and also go to model.config and change the name in line <sdf version="1.6">filename.sdf</sdf>

```
~/catkin_ws/src/two_wheels_gazebo/models
```

2.4. copy the file in ~/.gazebo/models

```
cp -r ~/catkin_ws/src/two_wheels_gazebo/models/<filename> ~/.gazebo/models
```

2.5. go to to the “world” file. The “world” file has all the component which appears in the enviroment

```
gedit ~/catkin_ws/src/two_wheels_gazebo//worlds/room.world
```

2.6. and add this

```
<include>
  <uri>model://<the name of the model></uri>
  <pose>0 0 0 0 -0 0</pose>
</include>
```

3. Jackal Description file can be found in [44], which is the GitHub file that contains all the files and parameter needs for making the Jackal model in the simulation. All required files will be uploaded into the system from section 2.1, at the time of installing the Jackal library. More information on how to simulate Jackal can be found in [45].
4. RVIZ is one of the powerful software tools uses on the ROS platform for 3D and 2D visualization the mobile robot information, such as log sensor information from robot’s sensors. RVIZ is a tool that can visualize how the robot is seeing the environment and how it’s performing. RVIZ can be used for debugging the robot application software.
5. Gmapping is the software library implemented in C++ on the ROS platform to perform SLAM. Gmapping is very user friendly and can interact with different ROS tools such as RVIZ and Gazebo. The Gmapping package software also downloads at the time of step 2.1.
6. In this thesis, the environment in Gazebo constructed in step 2. The robot model also obtained from the software package that installs in step 2.1. Two different rooms shape used in this research. For making the environment in figure 4.1(a), the filename in step 2.6 should be “Sqaure 3x4” and for constructing the figure, 4.1(b) should be “lcoridor”
 - 6.1. For navigating the robot in a constructed environment in simulation, these steps should be performed.
 - 6.1.1. Launch the gazebo with the constructed world


```
roslaunch my_jackal_gazebo jackal_world.launch config:=front_laser
```

6.1.2. Launch Gmapping package

```
roslaunch my_jackal_tools gmapping.launch
```

6.1.3. Launch the RVIZ for visualizing the Robot sensors. Use the navigation wheel around the robot to move around the robot in the Gazebo environment and make the map.

```
roslaunch jackal_viz view_robot.launch config:=gmapping
```

6.2. For navigating the robot in the real world, these steps should be performed.

6.2.1. Turn on the robot and connect the jackal computer to the Host computer. For connecting to the jackal computer, refer to the Jackal manual provided by Clearpath.

6.2.2. Launch the Gmapping package on the Jackal computer.

```
roslaunch my_jackal_tools gmapping.launch
```

6.2.3. Launch the RVIZ for visualizing the Robot sensors in the Host computer.

```
roslaunch jackal_viz view_robot.launch config:=gmapping
```

6.2.4.

6.3. After the desire map obtained, this command should be performed to save the map in the PGM file format. The filename mymap can be any desired name.

```
roslaunch map_server map_saver -f mymap
```

APPENDIX C. MATLAB SIMULATION PROGRAMS

For this research, two MATLAB programs were developed. One program for making the 2D map from the raw data, and the other one to perform the CNN.

C.1 Making the 2D map from raw data

MapBuilder_Dataset.m

```
%% Initialazation
% Clear all the variable before start the program
for icnt=131:132
clear A Angel_Steps Angels data fileID i Lidar_Start_Angel numScans Pos
Ranges Room_Category Scan_test sizeA;
%% Reading the file from dataset folder
% the data set should copy in the location like below. (Inside the matlab
% workspace or chang the address to the desire location. becarful that dont
% use spce in middle of the address names like "My Project")
STR_Root='/Users/imanyazdansepas/Desktop/MyStuff/Learning/university/SLAM_The
sis/Jackal_Private_PNW/Data_collected/DataColected/Room_Categorization/drdata
set/';
%STR_Root='/Users/imanyazdansepas/Desktop/MyStuff/Learning/university/SLAM_Th
esis/Jackal_Private_PNW/Data_collected/DataColected/Room_Categorization/myIma
ges';

%% Choos the folder
% chose one of the category from the below list:

%STR_IMG_FolderName='bedrooms_Img';
%STR_FolderName='bedrooms';

%STR_IMG_FolderName='bathrooms_Img';
%STR_FolderName='bathrooms'

%STR_IMG_FolderName='corridors_Img';
%STR_FolderName='corridors'

%STR_IMG_FolderName='kitchens_Img';
%STR_FolderName='kitchens'

%STR_IMG_FolderName='livingRooms_Img';
%STR_FolderName='livingRooms'

STR_IMG_FolderName='toilets_Img';
STR_Raw_FolderName='toilets'

%% Choos the file
% chose the name of the dataset
%STR_FileName='laserScansAndOdom134.txt';
NumberFile= int2str(icnt);
STR_FileName='laserScansAndOdom';
strf=strcat(STR_FileName,NumberFile, '.txt')
```

```

strf_IMG=strcat(STR_FileName,NumberFile, '.jpg');
%% Plugging data
% in this section the folder and the file is plug in to matlab commands.
% folder variable read the directory.
% filename construct the path of the file and
% fopen syntax open and read the data inside the files.
% and the Lookup_table construct to hold each category
folder=dir(STR_Root);

filename=[STR_Root '/' STR_Raw_FolderName '/' strf];
IMG_folder=[STR_Root '/' STR_IMG_FolderName];
fileID = fopen(filename,'r');
formatSpec = '%f';
Lookup_table=["Living","Coridoor","Bath","Kitchen","", "BedRoom","Toilet"];
%% Construct the matrices
% make the matrices from whole text file
% Row: the number of the scans and pose in each single position each data
% set has adiffrent rows number
% Column: 692= 682(number of the scans)+ room type+odometryx+Odometryy+...
% after that it transpose the matrices and retriive start angel, steps,
pos,...
sizeA = [692 Inf];
A = fscanf(fileID,formatSpec,sizeA);
data=A';
Lidar_start_Angel=data(2,6);
Angel_Steps=data(2,7);
Pos=data(1:end,3:5);
Angels=zeros(682,1);
Room_category=data(1,2);
numScans = numel(data(:,1))+1
%% make the Lidar cells
% construct the angle steps matrices
for i=1:682
Angels(i,:)=Lidar_start_Angel+Angel_Steps*i;
end
% construct the Lidar scan cells
for i=2:numScans
    Ranges=data(i-1,11:end);
    sacn_room=lidarScan(Ranges,Angels);
    Scan_test(1,i-1)={sacn_room};
end
%% make the map and generate the jpg file of the map
occGrid = buildMap(Scan_test,Pos,100,2);
mat = occupancyMatrix(occGrid);
%mat(mat == 128) = 255;
imwrite(mat, fullfile(IMG_folder, strf_IMG));
origsize=imread([IMG_folder '/' strf_IMG]);
im = imresize(origsize,[227 227]);
imwrite(im, fullfile(IMG_folder, strf_IMG));
%delete ([IMG_folder '/' strf_IMG])
% figure
% show(occGrid)
% title('Occupancy Map of '+Lookup_table(Room_category+1)+' ':'+STR_FileName)
% hold on
end

```

C.2 Room Categorization Program using CNN

RoomCategorization.m

```
% Copyright 2017 The MathWorks, Inc.
%% Load and Explore 2D layout Data
%TrainFolder='/Users/imanyazdansepas/Desktop/MyStuff/Learning/university/SLAM_Thesis/Jackal_Private_PNW/Data_collected/DataCollected/Room_Categorization/myImages_Extended3';
TrainFolder='/Users/imanyazdansepas/Desktop/MyStuff/Learning/university/SLAM_Thesis/Jackal_Private_PNW/Data_collected/DataCollected/Room_Categorization/myImages5/'; % for Windows
%TrainFolder='/Users/imanyazdansepas/Desktop/MyStuff/Learning/university/SLAM_Thesis/Jackal_Private_PNW/Data_collected/DataCollected/Room_Categorization/myImages_Rotated';
TestFolder='/Users/imanyazdansepas/Desktop/MyStuff/Learning/university/SLAM_Thesis/Jackal_Private_PNW/Data_collected/DataCollected/Room_Categorization/myImages'; %for mac

%TestFolder='C:\Users\imany\OneDrive\Desktop\Jackal_Private_PNW\Data_collected\DataCollected\Room_Categorization\myImages'; %for windows
%% Choose the category
% testSubFolder='living';
% testSubFolder='kitchen';
% testSubFolder='corrid';
% testSubFolder='bath';
% testSubFolder='bed';
% testSubFolder='toilets';
allImages = imageDatastore(TrainFolder, 'IncludeSubfolders', true, 'LabelSource', 'foldernames');
%% for tessting. it usfule during developing
labelCount = countEachLabel(allImages)
img = readimage(allImages,1);
size(img);
%% Specify Training and Validation Sets
% Divide the data into training and validation data sets, so that each category in the training set contains 180 images, and the validation set contains the remaining images from each label. splitEachLabel splits the datastore digitData into two new datastores, trainDigitData and valDigitData.
%numTrainFiles = 180;
numTrainFiles = 30;
[imdsTrain,imdsValidation] = splitEachLabel(allImages,numTrainFiles,'randomize');
%% Define the convolutional neural network architecture.
% Image Input Layer An imageInputLayer is where you specify the image size, which, in this case, is 28-by-28-by-1. These numbers correspond to the height, width, and the channel size. The digit data consists of grayscale images, so the channel size (color channel) is 1. For a color image, the channel size is 3, corresponding to the RGB values. You do not need to shuffle the data because trainNetwork, by default, shuffles the data at the beginning of training. trainNetwork can also automatically shuffle the data at the beginning of every epoch during training.
layers = [
    imageInputLayer([227 227 1]);
    % Convolutional Layer In the convolutional layer, the first argument is filterSize, which is the height and width of the filters the training
```

function uses while scanning along the images. In this example, the number 3 indicates that the filter size is 3-by-3. You can specify different sizes for the height and width of the filter. The second argument is the number of filters, `numFilters`, which is the number of neurons that connect to the same region of the input. This parameter determines the number of feature maps. Use the 'Padding' name-value pair to add padding to the input feature map. For a convolutional layer with a default stride of 1, 'same' padding ensures that the spatial output size is the same as the input size. You can also define the stride and learning rates for this layer using name-value pair arguments of `convolution2dLayer`.

% Batch Normalization Layer Batch normalization layers normalize the activations and gradients propagating through a network, making network training an easier optimization problem. Use batch normalization layers between convolutional layers and nonlinearities, such as ReLU layers, to speed up network training and reduce the sensitivity to network initialization. Use `batchNormalizationLayer` to create a batch normalization layer.

% ReLU Layer The batch normalization layer is followed by a nonlinear activation function. The most common activation function is the rectified linear unit (ReLU). Use `reluLayer` to create a ReLU layer.

```
convolution2dLayer(3,16,'Padding','same')
batchNormalizationLayer
reluLayer
```

% Max Pooling Layer Convolutional layers (with activation functions) are sometimes followed by a down-sampling operation that reduces the spatial size of the feature map and removes redundant spatial information. Down-sampling makes it possible to increase the number of filters in deeper convolutional layers without increasing the required amount of computation per layer. One way of down-sampling is using a max pooling, which you create using `maxPooling2dLayer`. The max pooling layer returns the maximum values of rectangular regions of inputs, specified by the first argument, `poolSize`. In this example, the size of the rectangular region is [2,2]. The 'Stride' name-value pair argument specifies the step size that the training function takes as it scans along the input.

```
maxPooling2dLayer(2,'Stride',2)
```

```
convolution2dLayer(3,32,'Padding','same')
batchNormalizationLayer
reluLayer
```

```
maxPooling2dLayer(2,'Stride',2)
```

```
convolution2dLayer(3,64,'Padding','same')
batchNormalizationLayer
reluLayer
```

% Fully Connected Layer The convolutional and down-sampling layers are followed by one or more fully connected layers. As its name suggests, a fully connected layer is a layer in which the neurons connect to all the neurons in the preceding layer. This layer combines all the features learned by the previous layers across the image to identify the larger patterns. The last fully connected layer combines the features to classify the images.

Therefore, the `OutputSize` parameter in the last fully connected layer is equal to the number of classes in the target data. In this example, the

output size is 10, corresponding to the 10 classes. Use `fullyConnectedLayer` to create a fully connected layer.

% Softmax Layer The softmax activation function normalizes the output of the fully connected layer. The output of the softmax layer consists of positive numbers that sum to one, which can then be used as classification probabilities by the classification layer. Create a softmax layer using the `softmaxLayer` function after the last fully connected layer.

% Classification Layer The final layer is the classification layer. This layer uses the probabilities returned by the softmax activation function for each input to assign the input to one of the mutually exclusive classes and compute the loss. To create a classification layer, use `classificationLayer`.

```

    fullyConnectedLayer(5)
    softmaxLayer
    classificationLayer];
%% Specify Training Options
% After defining the network structure, specify the training options.
% Train the network using stochastic gradient descent with momentum (SGDM)
% with an initial learning rate of 0.01. Set the maximum number of epochs
% to 4. An epoch is a full training cycle on the entire training data set.
% Monitor the network accuracy during training by specifying validation
% data and validation frequency. Shuffle the data every epoch. The software
% trains the network on the training data and calculates the accuracy on
% the validation data at regular intervals during training.
% The validation data is not used to update the network weights.
% Turn on the training progress plot, and turn off the command window output.
options = trainingOptions('sgdm', ...
    'InitialLearnRate',0.01, ...
    'MaxEpochs',20, ...
    'Shuffle','every-epoch', ...
    'ValidationData',imdsValidation, ...
    'ValidationFrequency',30, ...
    'Verbose',false, ...
    'Plots','training-progress');
%% Train Network Using Training Data
net = trainNetwork(imdsTrain, layers, options);
%% validating the network
CntPerc=0;
LoopCnt=0;
% for icnt=1:22 %for Living room
% for icnt=62:83 %for Kitchen
    for icnt=20:27 %for corridor
% for icnt=27:62 %for bath
% for icnt=101:130 %for bed
% for icnt=130:142 %for toilet
        NumberFile= int2str(icnt);
        STR_FileName='laserScansAndOdom';
        strf_IMG=strcat(STR_FileName,NumberFile,'.jpg');
        imgs = imageDatastore([TestFolder '/' testSubFolder '/' strf_IMG],
'IncludeSubfolders', true, 'LabelSource', 'foldernames');
        [pred,scores] = classify(net,imgs);
        % i=icnt;%for livingroom
        % i=icnt-61;%for Kitchen
        i=icnt-19;%for corridor
        % i=icnt-26;%for bath
        % i=icnt-100;%for bed
        % i=icnt-129; % for toilet
    end
end
end

```

```

subplot(6,7,i);
LoopCnt=LoopCnt+1;
%% showing the bar graph of each category
% highscores = scores > 0.01
% bar(scores(highscores))
% categorynames = net.Layers(end).ClassNames;
% xticklabels(categorynames(highscores))
%% showing the picture with the category that estimated
[M,I] = max(scores);
categorynames = net.Layers(end).ClassNames;
k=categorynames(I);
imshow(imgs.Files{1});
title(k);

if strcmp(k,testSubFolder)==1
    CntPerc=CntPerc+1;
end
end
strsubplot=strcat(int2str(CntPerc),' ', testSubFolder , ' ', 'correct case
over',' ', int2str(LoopCnt),' ', ' our accuracy is= ', ' ',
int2str((CntPerc/LoopCnt)*100), '%');
sgtitle(strsubplot);

```

REFERENCES

- [1] R. C. Smith and P. Cheeseman, “On the Representation and Estimation of Spatial Uncertainty,” *Int. J. Rob. Res.*, vol. 5, no. 4, pp. 56–68, 1986, doi: 10.1177/027836498600500404.
- [2] U. of O. J.J. Leonard (NEC Research Institute), H.F. Durrant-Whyte (Department of Engineering Science, “Simultaneous Map Building and Localization for Mobile Robots :,” *International Workshop on Intelligent Robots and Systems IROS*, vol. 3, no. November. pp. 1442–1447, 1991.
- [3] G. Dissanayake, S. B. Williams, H. Durrant-Whyte, and T. Bailey, “Map management for efficient simultaneous localization and mapping (SLAM),” *Auton. Robots*, 2002, doi: 10.1023/A:1015217631658.
- [4] N. Ayache and O. D. Faugeras, “Building, Registrating, and Fusing Noisy Visual Maps,” *Int. J. Rob. Res.*, 1988, doi: 10.1177/027836498800700605.
- [5] R. Chatila and J. P. Laumond, “Position referencing and consistent world modeling for mobile robots,” in *Proceedings - IEEE International Conference on Robotics and Automation*, 1985, doi: 10.1109/ROBOT.1985.1087373.
- [6] J. L. Crowley, “World modeling and position estimation for a mobile robot using ultrasonic ranging,” 1989, doi: 10.1109/robot.1989.100062.
- [7] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit, “FastSLAM: A factored solution to the simultaneous localization and mapping problem,” in *Proceedings of the National Conference on Artificial Intelligence*, 2002.
- [8] S. Thrun, W. Burgard, and D. Fox, “Real-time algorithm for mobile robot mapping with applications to multi-robot and 3D mapping,” in *Proceedings - IEEE International Conference on Robotics and Automation*, 2000, doi: 10.1109/robot.2000.844077.
- [9] K. P. Murphy, “Bayesian map learning in dynamic environments,” *Adv. Neural Inf. Process. Syst.*, pp. 1015–1021, 2000.
- [10] A. Doucett, N. Freitas, K. P. Murphy, and S. Russent, “Rao-Blackwellised particle filter based track-before-detect algorithm,” *IET Signal Process.*, vol. 2, no. 2, pp. 169–176, 2008, doi: 10.1049/iet-spr:20070075.

- [11] G. Grisetti, C. Stachniss, and W. Burgard, “Improved techniques for grid mapping with Rao-Blackwellized particle filters,” *IEEE Trans. Robot.*, 2007, doi: 10.1109/TRO.2006.889486.
- [12] “OpenSLAM.org.” <https://openslam-org.github.io/gmapping.html> (accessed Apr. 05, 2020).
- [13] A. Elfes, “Occupancy Grids: A Stochastic Spatial Representation for Active Robot Perception,” 2013, [Online]. Available: <http://arxiv.org/abs/1304.1098>.
- [14] H. Choset and K. Nagatani, “Topological simultaneous localization and mapping (SLAM): Toward exact localization without explicit localization,” *IEEE Trans. Robot. Autom.*, vol. 17, no. 2, pp. 125–137, 2001, doi: 10.1109/70.928558.
- [15] S. Thrun, “Learning metric-topological maps for indoor mobile robot navigation,” *Artif. Intell.*, vol. 99, no. 1, pp. 21–71, 1998, doi: 10.1016/S0004-3702(97)00078-7.
- [16] Ó. Martínez Mozos, R. Triebel, P. Jensfelt, A. Rottmann, and W. Burgard, “Supervised semantic labeling of places using information extracted from sensor data,” *Rob. Auton. Syst.*, vol. 55, no. 5, pp. 391–402, 2007, doi: 10.1016/j.robot.2006.12.003.
- [17] “AdaBoost -Wikipedia.” <https://en.wikipedia.org/w/index.php?title=AdaBoost&oldid=934066286> (accessed Apr. 05, 2020).
- [18] Ó. Martínez Mozos, R. Triebel, P. Jensfelt, A. Rottmann, and W. Burgard, “Supervised semantic labeling of places using information extracted from sensor data,” *Rob. Auton. Syst.*, 2007, doi: 10.1016/j.robot.2006.12.003.
- [19] A. Swadzba and S. Wachsmuth, “Categorizing perceptions of indoor rooms using 3D features,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 5342 LNCS, pp. 734–744, 2008, doi: 10.1007/978-3-540-89689-0_77.
- [20] J. Wu, H. I. Christensen, and J. M. Rehg, “Visual place categorization: Problem, dataset, and algorithm,” *2009 IEEE/RSJ Int. Conf. Intell. Robot. Syst. IROS 2009*, pp. 4763–4770, 2009, doi: 10.1109/IROS.2009.5354164.
- [21] P. Ursic, A. Leonardis, D. Skocaj, and M. Kristan, “Hierarchical spatial model for 2D range data based room categorization,” *Proc. - IEEE Int. Conf. Robot. Autom.*, vol. 2016-June, pp. 4514–4521, 2016, doi: 10.1109/ICRA.2016.7487650.

- [22] J. Zhang and S. Singh, “Visual-lidar odometry and mapping: Low-drift, robust, and fast,” in *Proceedings - IEEE International Conference on Robotics and Automation*, 2015, doi: 10.1109/ICRA.2015.7139486.
- [23] M. Velas, M. Spanel, and A. Herout, “Collar Line Segments for fast odometry estimation from Velodyne point clouds,” in *Proceedings - IEEE International Conference on Robotics and Automation*, 2016, doi: 10.1109/ICRA.2016.7487648.
- [24] F. Kallasi and D. L. Rizzini, “Efficient loop closure based on FALKO lidar features for online robot localization and mapping,” *IEEE Int. Conf. Intell. Robot. Syst.*, vol. 2016-Novem, pp. 1206–1213, 2016, doi: 10.1109/IROS.2016.7759202.
- [25] R. Goeddel and E. Olson, “Learning semantic place labels from occupancy grids using CNNs,” *IEEE Int. Conf. Intell. Robot. Syst.*, vol. 2016-Novem, pp. 3999–4004, 2016, doi: 10.1109/IROS.2016.7759589.
- [26] P. Uršič, D. Tabernik, M. Boben, D. Skočaj, A. Leonardis, and M. Kristan, “Room categorization based on a hierarchical representation of space,” *Int. J. Adv. Robot. Syst.*, vol. 10, 2013, doi: 10.5772/55534.
- [27] R. Goeddel and E. Olson, “Learning semantic place labels from occupancy grids using CNNs,” in *IEEE International Conference on Intelligent Robots and Systems*, 2016, doi: 10.1109/IROS.2016.7759589.
- [28] G. Dissanayake, H. Durrant-whyte, and T. Bailey, “A (slam),” no. April 2000, 2006.
- [29] K. P. Murphy, “Bayesian map learning in dynamic environments,” in *Advances in Neural Information Processing Systems*, 2000.
- [30] “PPT - Probabilistic Robotics PowerPoint Presentation, free download - ID:3360074.” <https://www.slideserve.com/cira/probabilistic-robotics> (accessed May 03, 2020).
- [31] W. S. McCulloch and W. Pitts, “A logical calculus of the ideas immanent in nervous activity,” *Bull. Math. Biophys.*, 1943, doi: 10.1007/BF02478259.
- [32] “Artificial neural network - Wikipedia.” https://en.wikipedia.org/wiki/Artificial_neural_network (accessed Apr. 05, 2020).
- [33] F. ROSENBLATT, “Simulation Experiments,” *Encycl. Meas. Stat.*, 1960, doi: 10.4135/9781412952644.n411.
- [34] michael A. Nielsen, “Neural Networks and Deep Learning,” in *Machine Learning*, 2015.

- [35] “What is Perceptron | Simplilearn.” <https://www.simplilearn.com/what-is-perceptron-tutorial> (accessed Apr. 05, 2020).
- [36] “Everything you need to know about Neural Networks and Backpropagation — Machine Learning Easy and Fun.” <https://towardsdatascience.com/everything-you-need-to-know-about-neural-networks-and-backpropagation-machine-learning-made-easy-e5285bc2be3a> (accessed Apr. 05, 2020).
- [37] “A Simple Explanation of the Softmax Function - victorzhou.com.” <https://victorzhou.com/blog/softmax/> (accessed Apr. 06, 2020).
- [38] “Robot Operating System - Wikipedia.” https://en.wikipedia.org/wiki/Robot_Operating_System (accessed Apr. 12, 2020).
- [39] P. Uršič, M. Kristan, D. Skocaj, and A. Leonardis, “Room classification using a hierarchical representation of space,” *IEEE Int. Conf. Intell. Robot. Syst.*, pp. 1371–1378, 2012, doi: 10.1109/IROS.2012.6385546.
- [40] “Confusion matrix - Wikipedia.” https://en.wikipedia.org/wiki/Confusion_matrix (accessed Apr. 17, 2020).
- [41] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet classification with deep convolutional neural networks,” *Commun. ACM*, 2017, doi: 10.1145/3065386.
- [42] “ROS.org | About ROS.” <https://www.ros.org/about-ros/> (accessed Apr. 25, 2020).
- [43] “kinetic/Installation/Ubuntu - ROS Wiki.” <http://wiki.ros.org/kinetic/Installation/Ubuntu> (accessed Apr. 25, 2020).
- [44] “jackal/jackal.urdf.xacro at melodic-devel · jackal/jackal.” https://github.com/jackal/jackal/blob/melodic-devel/jackal_description/urdf/jackal.urdf.xacro (accessed Apr. 25, 2020).
- [45] “urdf/Tutorials/Using Xacro to Clean Up a URDF File - ROS Wiki.” [http://wiki.ros.org/urdf/Tutorials/Using Xacro to Clean Up a URDF File](http://wiki.ros.org/urdf/Tutorials/Using%20Xacro%20to%20Clean%20Up%20a%20URDF%20File) (accessed Apr. 25, 2020).