THE REPUBLIC OF TURKEY
BAHÇEŞEHİR UNIVERSITY

VISION BASED INDOOR MOBILE ROBOT LOCALIZATION USING DEEP LEARNING

Master's Thesis

ARAFAT SHARIF

İSTANBUL, 2020

THE REPUBLIC OF TURKEY
BAHÇEŞEHİR UNIVERSITY

THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
MECHATROCNICS ENGINEERING

VISION BASED INDOOR MOBILE ROBOT LOCALIZATION USING DEEP LEARNING

Master's Thesis

ARAFAT SHARIF

Thesis Supervisor: Assoc. Prof. Dr. BERKE GÜR

İSTANBUL, 2020

T.C.
BAHCESEHIR UNIVERSITY
GRADUATE SCHOOL

...../...../.....

MASTER THESIS APPROVAL FORM

Program Name:	Mechatronics Engineering
Student's Name and Surname:	Arafat SHARIF
Name Of The Thesis:	Vision Based Indoor Mobile Robot Localization Using Deep Learning
Thesis Defense Date:	16 September, 2020

This thesis has been approved by the Graduate School which has fulfilled the necessary conditions as Master thesis.

Assoc. Prof. Dr. Burak KÜNTAY
Institute Director

This thesis was read by us, quality and content as a Master's thesis has been seen and accepted as sufficient.

	Title/Name	**Signature**
Thesis Advisor's	Assoc. Prof. Dr. Berke Gür	
Member's	Prof. Dr. Nafiz Arıca	
Member's	Doc¸. Dr. Ahmet Yazıcı	

ACKNOWLEDGEMENTS

I would like to express my gratitude for my family, my friends and colleges for their support, and specially Assoc. Prof. Dr. Berke Gür for his guidance and constructive criticism and Bahçeşehir University.

This work was supported by The Scientific and Technological Research Council of Turkey (TUBITAK) 1003 - Primary Subjects R&D Funding Program project no. 116E853.

İstanbul, 2020 Arafat SHARIF

ABSTRACT

VISION BASED INDOOR MOBILE ROBOT LOCALIZATION USING DEEP LEARNING

Arafat SHARIF

Mechatronics Engineering
Thesis Supervisor: Assoc. Prof. Dr. Berke Gür

September 2020, 73 Page

Of the three main questions of mobile robot navigation, 'where am I?' is the first and foremost question (the other two being 'where do I need to go?' and 'how do I get there?') that the robot must answer before performing any further action. Outdoor localization is a relatively simple task due to the availability of very accurate GPS and similar satellite-based absolute localization techniques. In the absence of satellite navigation technology, simultaneous localization and mapping (SLAM) approaches based on Bayes filtering and graph optimization using data from a multitude of on-board sensors such as sonar, IR sensors, encoders, inertial measurement units (IMU), some which are relatively costly (e.g., LIDAR, laser scanner/beacon systems) has become the state-of-art approach to indoor mobile robot localization. Although such SLAM methods deliver very high localization performance, they suffer from high CPU (due to image processing) and data storage (due to highly detailed digital maps) requirements, which strain the limited on-board computational resources and drive up the cost of hardware.

To alleviate this problem of indoor robot localization based on SLAM techniques, a vision and deep learning-based based indoor mobile robot localization method is proposed in this thesis. The input of the system is a RGB image captured by a monocular camera. The output of the system is the 2D location and orientation of the robot. The proposed method integrates different types of neural network layers to produce a system that is capable of presenting the map and producing the output based on a database that can be labeled with any localization sensor. The proposed method was tested and validated through several simulations and an experimental setup. Results indicated that the proposed method delivered a localization performance similar to the state-of-the-art RTAB-Map graph-based approach SLAM approach (positioning error $<$ 5cm, orientation error $<$ 2 degrees) with a significantly smaller computation and storage burden.

Keywords: Indoor localization, SLAM, Deep learning, Camera, Mobile Robot .

ÖZET

DERİN ÖĞRENME KULLANILARAK GÖRÜŞ TABANLI İÇ MOBİL ROBOT YERELLEŞTİRME

Arafat Sharif

Mekatronik Mühendisliği
Tez Danışmanı: Doç. Dr. Berke Gür

Eylul 2020, 73 Sayfa

Mobil robot navigasyonunun üç ana sorusundan 'neredeyim?' robotun başka bir işlem yapmadan önce yanıtlaması gereken ilk ve en önemli soru (diğer ikisi 'nereye gitmem gerekiyor?' ve 'oraya nasıl gidebilirim?'). Dış mekanda yerelleştirme, çok doğru GPS ve benzer uydu tabanlı mutlak yerelles¸tirme tekniklerinin mevcudiyeti nedeniyle nispeten basit bir iştir. Uydu navigasyon teknolojisinin yokluğunda, Bayes filtreleme ve sonar, IR sensörleri, kodlayıcılar, atalet ölçüm birimleri (IMU) gibi çok sayıda yerleşik sensörden gelen verileri kullanarak grafik optimizasyonuna dayanan es¸zamanlı yerelles¸tirme ve haritalama (SLAM) yaklaşımları, bazıları nispeten maliyetli olan (örneğin, LIDAR, lazer tarayıcı / is¸aret sistemleri) kapalı mekan mobil robot lokalizasyonuna en son teknoloji yaklaşımı haline gelmiştir. Bu tür SLAM yöntemleri çok yüksek yerelleştirme performansı sağlasa da, sınırlı yerleşik hesaplama kaynaklarını zorlayan ve donanım maliyetini artıran yüksek CPU (görüntü işleme nedeniyle) ve veri depolama (son derece ayrıntılı dijital haritalar nedeniyle) gereksinimlerinden muzdariptirler.

SLAM tekniklerine dayalı bu kapalı robot yerelleştirme sorununu hafifletmek için bu tezde vizyon ve derin öğrenme tabanlı bir kapalı mobil robot yerelleştirme yöntemi önerilmiştir. Sistemin girişi, monoküler bir kamera tarafından yakalanan bir RGB görüntüsüdür. Sistemin çıktısı, robotun 2D konumu ve yönelimidir. Önerilen yöntem, haritayı sunabilen ve çıktıyı herhangi bir yerelleştirme sensörüyle etiketlenebilen bir veri tabanına dayalı olarak üretebilen bir sistem üretmek için farklı türlerdeki sinir ağı katmanlarını entegre eder. Önerilen yöntem, birkaç simülasyon ve deneysel bir düzenek aracılığıyla test edildi ve doğrulandı. Sonuçlar, önerilen yöntemin son teknoloji RTAB-Map grafik tabanlı yaklaşım SLAM yaklaşımına benzer bir yerelleştirme performansı sağladığını gösterdi (konumlandırma hatası $<$5 cm, yönlendirme hatası $<$2 derece) önemli ölçüde daha küçük bir hesaplama ile ve depolama yükü.

Anahtar Kelimeler: İç mekan lokalizasyonu, SLAM, Derin öğrenme, Kamera, Mobil Robot.

CONTENTS

TABLES

FIGURES

ABBREVIATIONS

2D	:	Two Dimensional
3D	:	Three Dimensional
ADAM	:	Adaptive Moment Estimation
AGV	:	Automatic Guided Vehicle
AMCL	:	Adaptive Monte Carlo Localization
Adagrad	:	Adaptive Gradients
BA	:	Bundle Adjustment
BAU	:	Bahcesehir University
BRISK	:	Binary Robust Invariant Scalable Keypoints
CGI	:	Computer-Generated Imagery
cm	:	Centimeters
CNN	:	Convolutional Neural Network
CSV	:	Comma-Separated Values
CUDA	:	Compute Unified Device Architecture
DOF	:	Degrees Of Freedom
EKF	:	Extended Kalman Filter
Elu	:	Exponential linear unit
FFNN	:	Feed-Forward Neural Network
g2o	:	Graph Optimization
GPS	:	Global Positioning System
GPU	:	Graphics Processing Unit
HERO	:	Heathkit Educational RObot
IMU	:	Inertial Measurement Unit
IR	:	Infrared
KUKA	:	Keller und Knappich Augsburg
m	:	Meters
MAE	:	Mean Absolute Error

MSER	:	maximally stable extremal regions
NN	:	Neural Network
NTP	:	Network Time Protocol
ORB	:	Oriented fast and Rotated Brief
QR	:	Quick Response
R-C NN	:	Region-based Convolutional Neural Network
RAM	:	Random-Access Memory
RFID	:	Radio-Frequency Identification
RGB	:	Red-Green-Blue
RGB-D	:	Red-Green-Blue-Depth
RMSprop	:	Root Mean Squared Propagation
ROS	:	Robot Operating System
RTAB-Map	:	Real-Time Appearance-Based Mapping
Relu	:	Rectified linear unit
SFM	:	Structure From Motion
SIFT	:	scale-invariant feature transform
SLAM	:	Simultaneous Localization And Mapping
SSH	:	Secure Shell
SURF	:	Speeded Up Robust Features
Selu	:	Scaled exponential linear unit
TF	:	Transform
Tanh	:	Hyperbolic Tangent
UNIX	:	Uniplexed Information and Computing System
V-SLAM	:	Visual Simultaneous Localization And Mapping
VGG	:	Visual Geometry Group
VICON	:	Video Converter
VO	:	Visual Odometry
WiFi	:	Wireless Fidelity

SYMBOLS

Corrected first moment	:	$\hat{m}$
Corrected second moment	:	$\hat{\sigma}^2$
Decay rate	:	β
Degrees	:	$^\circ$
Desired output	:	r
Hidden neuron	:	h
Input	:	x
Input to hidden layer	:	z
Learning factor	:	η
Length	:	L
Neuron	:	j
Number	:	n
Number of neurons	:	d
Output	:	y
Partial differential	:	∂
Pitch angle	:	ψ
Position in Y-axis	:	Y
Position in X-axis	:	X
Predicted Pitch angle	:	ψ^J
Predicted Position in Y-axis	:	Y^J
Predicted position in X-axis	:	X^J
Second Weight	:	v
Smoothing term	:	ϵ
Squared meter	:	m^2
The first moment	:	m
The second moment	:	σ^2
Time	:	t

Total number of images	:	T
Weight	:	w
Width	:	W

1. INTRODUCTION

An indoor mobile robot is a robot that has the capability of moving around its surroundings. Indoor indicates that the robot operates in an indoor environment.

Mobile robots can be manually, automatically (line following or on tracks) or autonomously controlled. They may be classified by the device they use to move to legged robot, wheeled robot or on tracks.

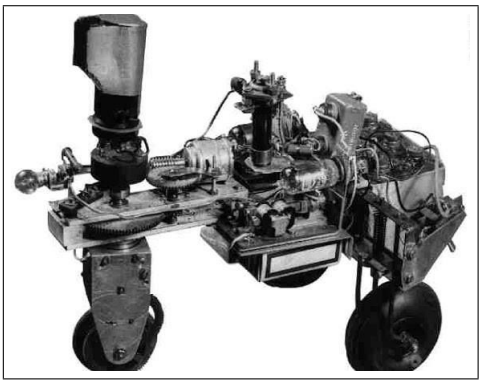
In this chapter, we display the usage of the mobile robots indoors, then we show the historical background on indoor localization for mobile robots and lastly we present the motivation and the contributions of this thesis.

1.1 THE USE OF MOBILE ROBOTS IN INDOOR ENVIRONMENTS

Generally, indoor mobile robots have many uses that vary from one field to another, the main purpose in mid last century was developing indoor mobile robots for research in educational fields.

Historically, William Grey Walter invented the first autonomous indoor mobile robot shown in figure 1.1 named Machina Speculatrix in 1948. Consisting of a light sensor, touch sensor and two motors, it had a dazzeled movement while moving towards light. After that, universities started competing to develop and reveal a new generation of indoor mobile robots.

Figure 1.1: Machina Speculatrix.



Source: https://sites.google.com/view/machinaspeculatrix/home

In the 1960s, John Hopkins University introduced the Beast, It had a physical touch sensor that would keep contact with the wall to navigate around the environment and Stanford University introduced Shakey named after its wobbly structure, it had a camera, rangefinder and a bumber sensor, it was the first robot to reason about its action.

The educational field reached a peak in mid 90s when Khepera mobile robot, in figure 1.2(a), was developed and sold to a thousand research labs and featured on the cover of nature. At the same time the Pioneer mobile robot, in figure 1.2(b) became commercially available enabling a widespread increase in robotics research and university studies over the next decade as mobile robotics became a standard part of universities curriculum.

Figure 1.2: (a)Khepera. (b) Pioneer.

(a) (b)

(a) Source: Georgia Institute of Technology. (b) Source: ActivMedia Robotics.

In 1982, RB5X a personal mobile robot with a manipulator presented in figure 1.3 was invented for domestic use. Heathkit Educational robot produced a series of several educational robots under the name of HERO series.

Figure 1.3: RB5X.

Source:RB Robotics

The interest of the public in indoor robots rose, resulting in robots that could be purchased for home use. These robots served entertainment or educational purposes such as Aibo robot that mimics a dog or as Robosapien a toy-like biomorphic toy robot.

The biggest market for indoor mobile robots for domestic uses were cleaning robots. In 1997, Cyberclean Systems developed the first fully autonomous vacuum cleaning robot that self-charged, operated elevators and vacuumed hallways with no human intervention. And in 2002, Roomba which appears in figure 1.4, a domestic autonomous mobile robot that cleans the floor with different features was commercially available.

Figure 1.4: Roomba

Source: iRobot Corporation

Indoor mobile robots played a big part in transporting equipment, drug delivery, patient services, and other nursing functions that could be easily adapted to robots. In the 1990s Joseph Engelberger, father of the industrial robotic arm, worked with colleagues to design the first commercially available autonomous mobile hospital robots, sold by Helpmate and shown in figure 1.5.

Figure 1.5: HelpMate

Source: HelpMate

Specialized equipment may be mounted on robots, allowing them to assist with surgical procedures and Customer support. The TUG robot made specifically for hospitals by Aethon became a popular means for hospitals to move medical equipments and materials. And in 2007 Speci-Minder started carrying blood and other patient samples from nurses' stations to various labs and transporting lab specimens. Security, timeliness, and reliability are key factors in evaluating this task.

Also taking advantage of its maneuverability and the ability to mount equipment, it is also used in security fields. For example, figure 1.6 presents PatrolBot a customizable autonomous service robot system that can follow people and detect doors. It does tasks such as scan buildings, create floor plans and navigate them autonomously using a laser range-finding sensor, delivery and environmental monitoring.

Figure 1.6: PatrolBot

Source: MobileRobots Inc

The ability of Maneuvering or transportation was heavily invested in by factories and warehouses, AGVs are employed in nearly every industry, they are preferred method of moving materials whether its heavy, light, normal or hazard materials.

Warehouses have installed mobile robotic systems to efficiently move materials across warehouses from production to stocking or from stocking shelves to order fulfillment zones. The labor demands for employees will be lessened as robots will be able to work alongside humans.

In 2007, Amazon Robotics, formerly Kiva Systems, manufactured mobile robotic fulfillment systems shown in figure 1.7, these automated shelving units sort themselves. OTTO motors also produces different models of autonomous mobile robots for Warehouses.

Figure 1.7: AMAZON robots

Source: Amazon Robotics

1.2 BACKGROUND ON INDOOR LOCALIZATION

As mobile robots started to be used commercially the earliest approach to solve the localization question was to have the robots attached to a rigid object that would move, like being attached to a wall or move on special tracks or wires. The robot could also be guided using magnetic or colored tape as shown in figure 1.8. These methods do not give the robot any degree of autonomy, on the contrary, they constrain it in a certain path that requires the change of the infrastructure to relocate the guided path.

Figure 1.8: Magnetic tape guidance AGV.

Source: IKV Robot

Objects or marks can also be used to help the process such as QR codes or RFID tags. The robot would read these codes and tags using a QR reader or RFID reader and re-localize the robot to the position of the tags.

Attaching physical devices and sensors to the robot to localize itself based on the robot movements led to the presence of the odometer. Encoders' Odometry is the description of the motion of a robot as position and orientation using Encoders. Encoders are sensors attached to the wheels and joints that convert motion into electric signals. The position and orientation are usually described relative to the initial frame of the robot on initialization. The downside in this sensor is that it accumulate error that causes a drift to the odometry. And the odometry would be totally wrong in some cases such as re-localizing the robot by carrying it or moving the wheels on a slippery floor.

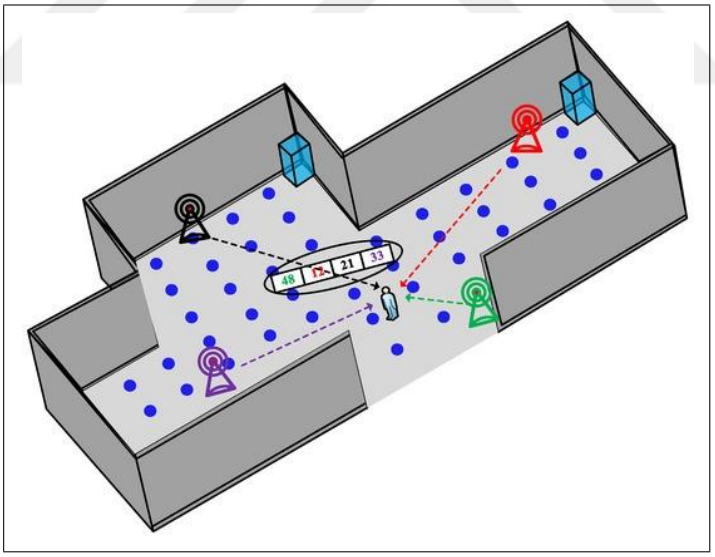
Visual Odometry is the description of the motion of a robot as position and orientation using a camera, it estimates the camera motion from using correlation to establish corre-

spondence of two consecutive images. These images can be taken by single cameras or stereo cameras. Stereo cameras perform better than the mono, however, they also accumulate error.

IMU is a sensor the calculates the acceleration and angular velocity, based on these inputs the orientation of the robot can be calculated with drift.

Another approach was using approximation techniques based on a transmitter and receiver technologies such as IR sensors that emits and detects infrared radiations and based on the reflected signal, the infrared sensor could determine the location of the objects around it. Ultrasound technique is also used by transmitting ultrasound waves and having multiple receivers in the environment that calculates the time of flight and locate the robot. Wi-Fi and Bluetooth use time of travel or the strength of the signal to calculate the position using triangulation as shown in figure 1.9. However, approximation techniques lack accuracy.

Figure 1.9: Wi-Fi indoor localization fundamentals.



Source: (Zhang et al., 2020)

As robots gained more autonomy sensors like Lidars appeared, they are used to continuously scan and update the environment, they publish the distance from the robot to every

obstacle in the environment.

And lastly, one of the most recent and accurate devices are the motion tracking systems using high definition cameras such as VICON cameras, these methods are usually used in CGI and animation movies, the disadvantage is that in order to localize a robot, the robot has to be always in the sight of the cameras. In addition, both Lidars and motion tracking systems are very expensive.

1.3 MOTIVATION

Localization is the most fundamental process that all other procedures such as navigation and path planning are based on, for that reason it gains a great research attention.

The research in indoor localization started in order to produce alternatives to GPS technology, which works outdoors with an accuracy of meters, physical sensors used in indoor localization have a noise that causes drifts in the reading of the environment. Sensor fusion and other probability based approaches were developed in order to decrease the drift. But didn't eliminate it.

The kidnapped robot problem is the issue when an autonomous robot is carried from its position to another position, or turned off and on in a different place in the map. It causes a localization failure and usually used to test the robot's ability to recover from catastrophic localization errors.

To overcome all these errors and challenges and in order to give the mobile robot the maximum degree of autonomy we use a camera as our main sensor in this thesis. Using a camera gives more potential to the robot by allowing engineers to take advantage of various platforms, frameworks and libraries to program it. That will also result in a high flexibility of the robot. Furthermore, we can perform many various tasks using a camera. Cameras are cheap, available and one camera can replace multiple sensors.

Vision based localization presented in feature detection and tracking in images used in V-SLAM systems are considered state-of-the-art indoor localization. But do we really need these outdated feature recognition methods, high computational and storage consumption and unrealistic assumption in the perception process, is it possible to learn these parts of the system just from a large number of examples?

1.4 CONTRIBUTIONS

Despite the attention that localization has in research. Yet the use of deep learning in localization is still recent and short. Influenced by the revolution of image recognition that deep learning caused, we developed a method to localize indoor mobile robots based only on the camera input. Deep learning based generates many features from images and apply an advanced analysis.

The proposed system consists of multiple layers of neural networks that process an RGB image to output the absolute location and orientation of the mobile robot.

To summarize, the contribution of this work is:

(a) The creation of a Deep learning localization model that produces an absolute location and orientation based on an image.

(b) Having a state-of-the-art performance with insignificant error that does not accumulate or fall in the kidnapped robot problem.

(c) Independence of high storage or memory consumption that allow it to work in large indoor environments unlike V-SLAM.

In the next chapter, we go through a literature review on indoor localization for mobile robots, after that a theoretical background for Deep learning and V-SLAM. The fourth chapter explains the proposed method to localize the indoor robot which was used and

tested in simulations in chapter five and in a real environment in chapter six. The thesis is concluded in chapter seven.

2. LITERATURE REVIEW

For indoor mobile robot localization, we can consider reviewing the literature from two approaches, the historical localization research and methods, and the usage of Deep Learning in localization research.

In the historical localization research and methods, we review the three fundamental SLAM paradigms, Markov Process Based Filtering including Extended Kalman Filter, Particle Filtering and Graph Based Optimization. As for the second part, we view some recent examples of the usage of Deep learning in Localization or assisting the Localization process.

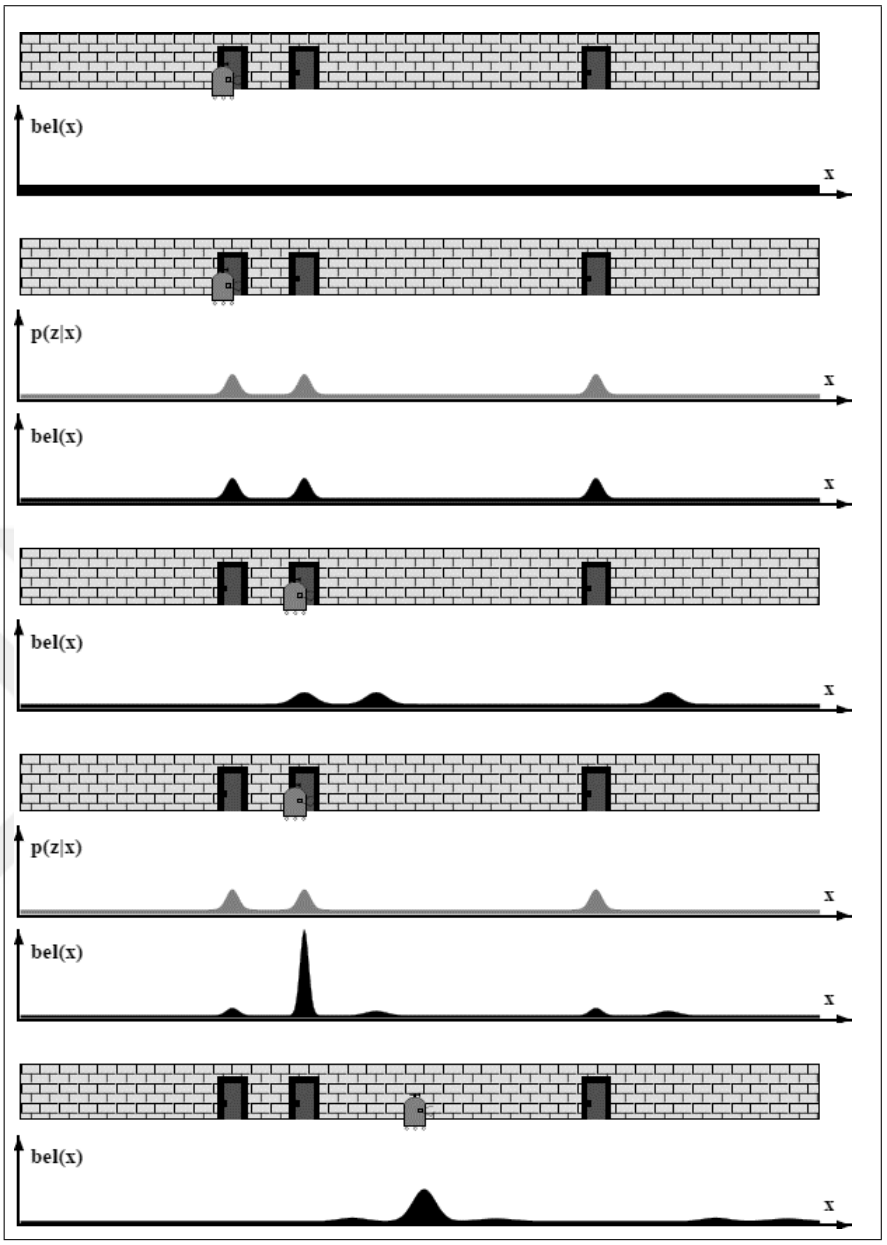
2.1 LOCALIZATION PARADIGMS LITERATURE REVIEW

Filtration based methods were the first approaches considered and have been used for a long time, these methods are computationally costly and depend on linearization and approximation.

2.1.1 Markov Process Based Filtering

Bayes filters to the localization problem is called Markov localization. Markov localization depends on two stages, the first is generating a probability where the robot would be based on previous belief and odometric input and the second is generating a probability based on the exteroceptive sensors in the environment. Combining these probabilities gives the answer. Figure 2.1 shows an example of a robot determining these probabilities.

Figure 2.1: A mobile robot during Markov Localization.

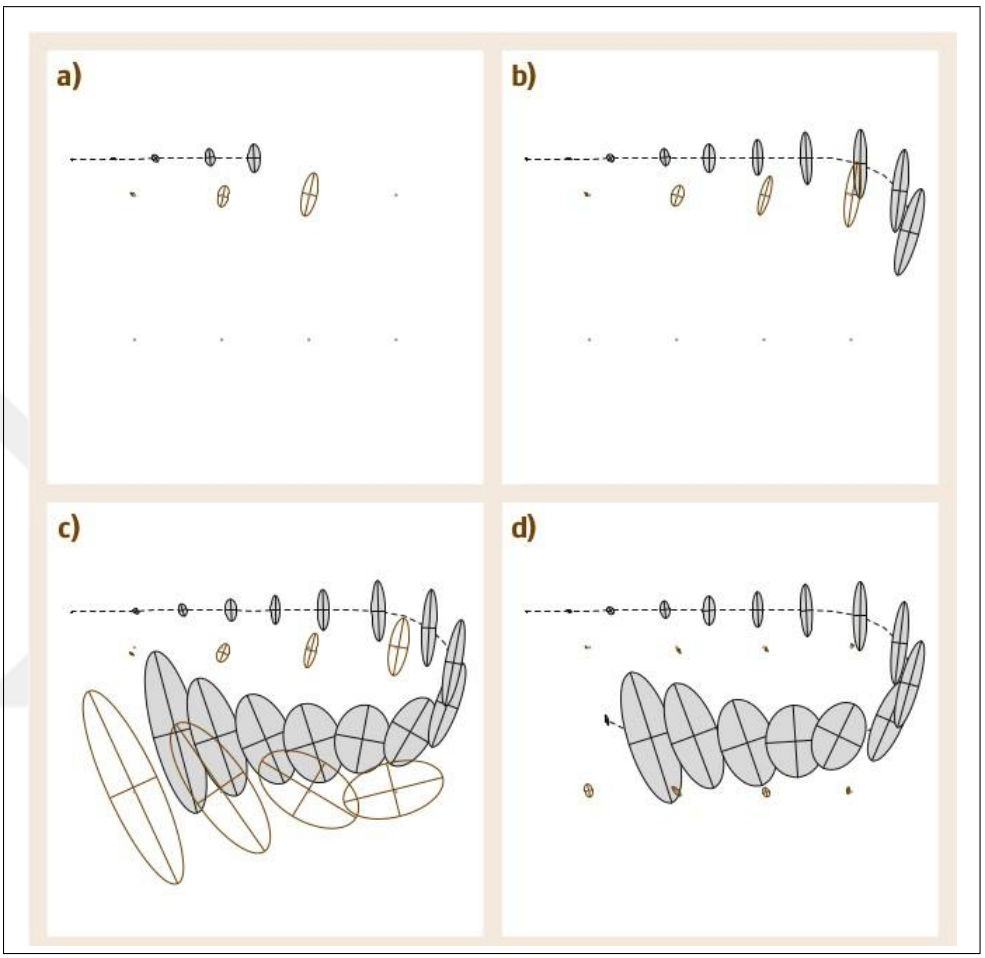


Source: (Ibrahim et al., 2011)

The Markov Localization model can represent any probability density function regarding the robot's position. However, this approach is extremely general and some authors describe it as insufficient.(Malagon-Soldara et al., 2015).

To solve that Extended Kalman Filter was introduced by Kalman, EKF is the nonlinear derivation of Kalman Filter (Kalman et al., 1960) which estimates a current state of the robot and its uncertainties and updates the estimation using a weighted average.

Figure 2.2: Robot's path and estimates using EKF.



Source: (Siciliano & Khatib, 2016).

As figure 2.2 above shows when the robot moves along the path the uncertainty in the landmarks' locations grows and so does the error until the robot observes an obstacle that was detected before.

In (Kim & Lee, 2007) an active beacon system is composed of an RFID receiver and an ultrasonic transmitter, four beacons detect the RFID signal and measure the distance from

each beacon, an Extended Kalman Filter for the nonlinear system is applied to improve the estimation of the accuracy.

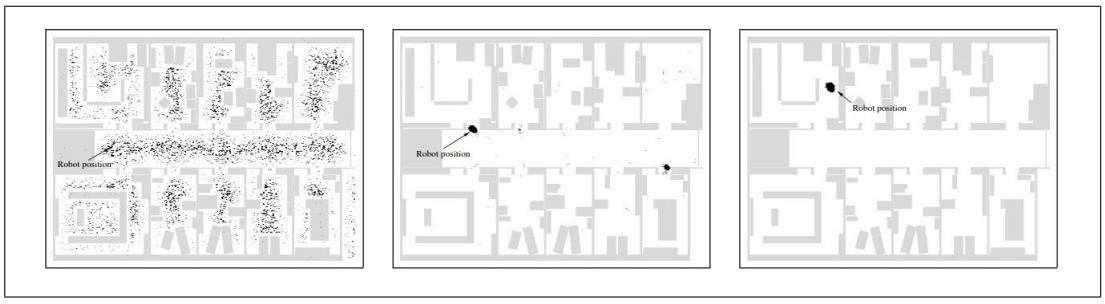
It can also give a highly accurate estimations using multiple sensors such as combining wheels encoders' readings with IR sensor readings to get the location using EKF (Oh et al., 2014).

2.1.2 Particle Filtering

Represented as Monte Carlo Localization method (Dellaert et al., 1999) where it presents the next probability function as a set of particles, each particle is a representation of a possible state of the robot and these particles survive based on the measurement of the sensors available in the environment as shown in figure 2.3. It consists of two stages, firstly prediction where every particle's probability of a set of particles that is computed in the previous iteration is calculated. And a new set is obtained based on these probabilities.

The second stage is update where the sensor measurement is taken into consideration and each sample is weighted to obtain a new set of particles based on these weights. Both stages are repeated recursively.

Figure 2.3: Monte Carlo localization process.



Source: (Dellaert et al., 1999).

Such an example would be four ultrasonic range sensors attached to a servo motor that rotates in 30 degree intervals and the sensors give the distance to the walls and obstacles that

allows to create a map and localize using Particle Filter method using a certain number of particles (Lim et al., 2015).

Localization using a Wi-Fi algorithm based on Monte Carlo Filtering can also be done, the probability is based on the strength of the Wi-Fi signal and the encoders' readings, in (Biswas & Veloso, 2010) the mean localization error was 0.7 meters.

It can also be used with a large set of sensors such as Kinetic Camera as a laser scanner to build a map and then localize the robot by distributing particles and eliminating them by decreasing their probabilities based on the obstacles detected by the laser-scan (Hamzeh & Elnagar, 2015). Bluetooth can be used in the localization of a robot by using Trilateration based on multiple devices and Monte Carlo to get a localization error of 0.427 ± 0.229 meters (Raghavan et al., 2010).

The disadvantage in this method would be the large number of particles that would be established and the time it takes to determine the particle with the highest probability, which means that the robot will be moving around the environment for a while before it determines its position.

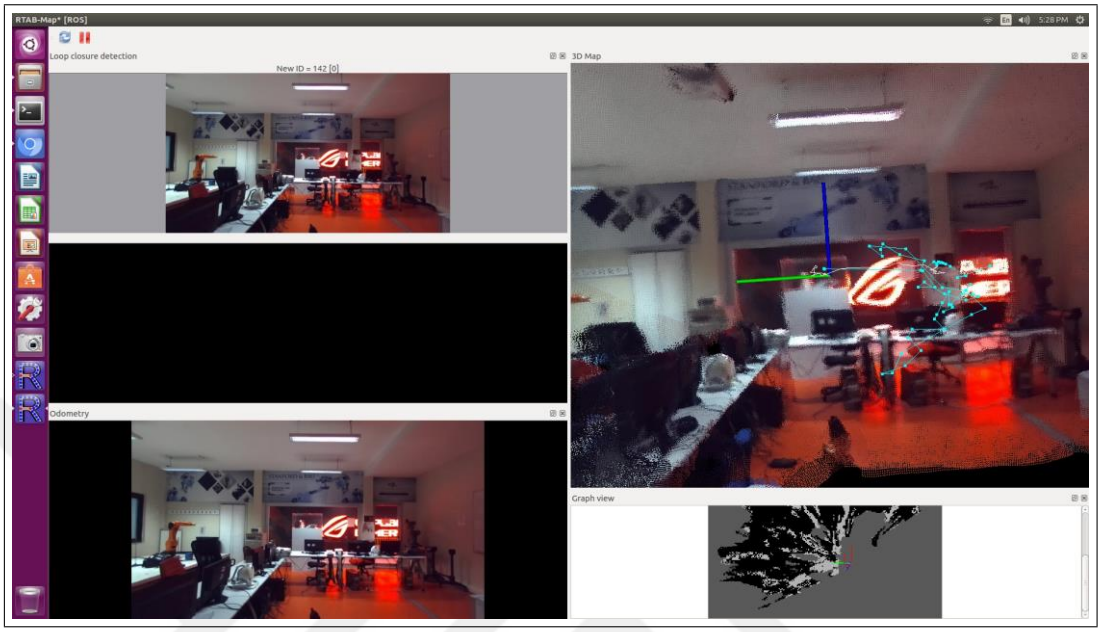
2.1.3 Graph Based Optimization

Optimization based approach is considered state-of-the-art, advantageous compared to the filter-based SLAM approaches, because non-linear optimization methods cope efficiently with the large number of feature measurements.(Strasdat et al., 2012) And avoids linearization and approximation.

The Graph Based Optimization approach uses a graph to represent the problem consisting of nodes and edges where every node represents a pose of the robot during mapping and every edge represents a spatial constraint between the nodes. The graph created is sent from the front end to the back end to be optimized. It aims to minimize the error introduced by the constraints. At the end, the map is determined by correcting the nodes.

Figure 2.4 below shows the creation of a 3D map consisting of nodes and edges in the process of RTAB-Map which is an example of the Graph Based Optimization method.

Figure 2.4: An example of Graph based approach localization process.



Overall Visual SLAM systems consist of: Sensor data which is the images from the camera, Front end including feature extraction, feature tracking and Back end including loop detection and closure and camera pose optimization.

A pose-based SLAM system computes the robot motion between the consecutive robot poses using an input such as Visual Odometry (VO), which estimates the trajectory of a camera computing the ego-motion between the consecutive frames, and then constructs a graph whose vertices are those poses, whereas the motion-related constraints are considered as edges, a loop closure introduces pose-to-pose constraints that correct the drift of the trajectory.

A drawback of this approach is that it costs high computational memory as it incorporate all the pose estimates in the calculation process, high storage space because it stores the images and maps, and it uses outdated feature extractions such as SURF and ORB.

2.2 DEEP LEARNING LOCALIZATION LITERATURE REVIEW

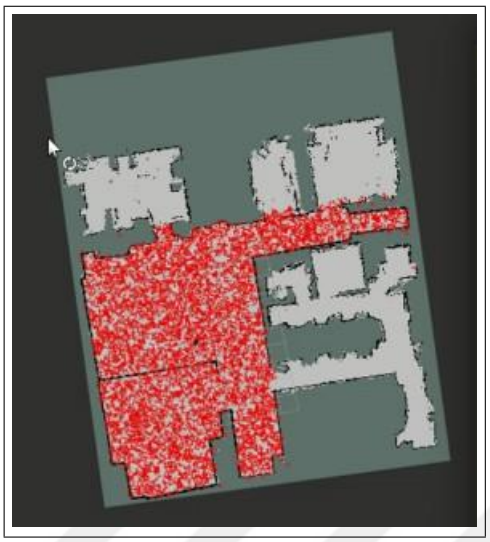
In this section we review the use of Deep Learning in Localization, Deep learning can assist the localization process by substituting one of the stages of the localization process or can replace the whole localization technique by receiving an input and providing the location and orientation.

2.2.1 Individual Components of SLAM Systems

Individual components of SLAM systems have recently been tackled with supervised deep learning methods.

Convolutional neural network or CNN is a neural network layer that filters the image and produces a feature map, CNN is highly recommended in classifying images. In assisting the Localization problem, it was used to classify a set of images taken in four different rooms so that it can label each image to its room number, then modify Adaptive Monte Carlo Localization (AMCL) by launching the particles in that room instead of the whole global map as shown in figure 2.5 so that it can convert faster (Bettaieb, 2017).

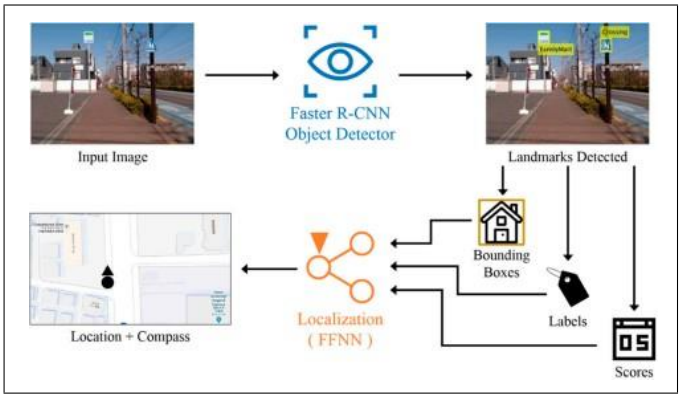
Figure 2.5: Particles launching in one room.



Source: (Bettaieb, 2017).

Deep learning revolutionized the way we tackle image detection with multiple models that are more efficient than the aged feature based methods. One of these models is Faster R-CNN (Ren et al., 2015), which is used in (Nilwong et al., 2019) to detect nine landmarks in the area of the experiment and feed them into a FFNN that returns the output as shown in figure 2.6 below.

Figure 2.6: Faster R-CNN usage in localization model.



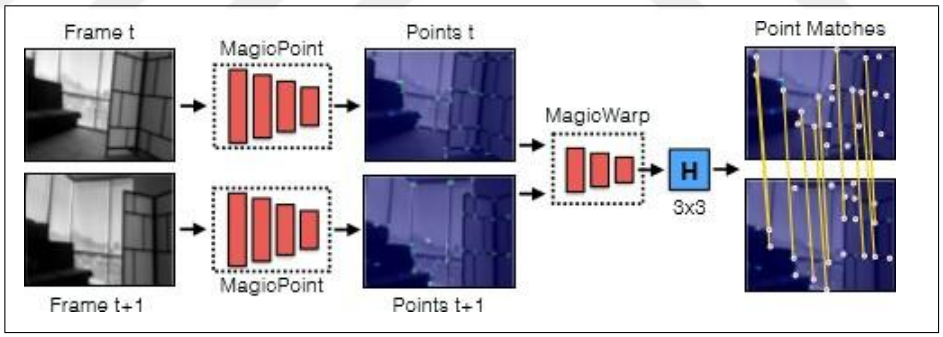
Source: (Nilwong et al., 2019).

Another method proposed in (Nilwong et al., 2019) is to apply CNN and train the whole image with 5 CNN layers. The downside in the faster R-CNN method is that we have to deal with a detection error alongside the localization error. As for the results they describe that the minimum results for both methods were in meters and their average errors were relatively high, compared to the GPS that has approximately 4.9 m error.

In (Tateno et al., 2017) CNN was used to produce a dense layer out of the mono image by training it on depth images from a Microsoft Kinect camera to replace the need of depth image in SLAM, compared to other mono SLAM it had an accuracy of 0.246 m.

In (DeTone et al., 2017) the neural network model depends on two networks, MagicPoint which consists of multiple CNN layers that takes an image and produce interest points in the images trained on ground truth corner locations, it does that for the next image. The second network MagicWarp consists of a concatenate layer to merge the output of MagicPoint, a VGG network and two dense layers produce a homography that estimates correspondence in image pairs without interest point descriptors as shown in figure 2.7.

Figure 2.7: Homography estimation model.



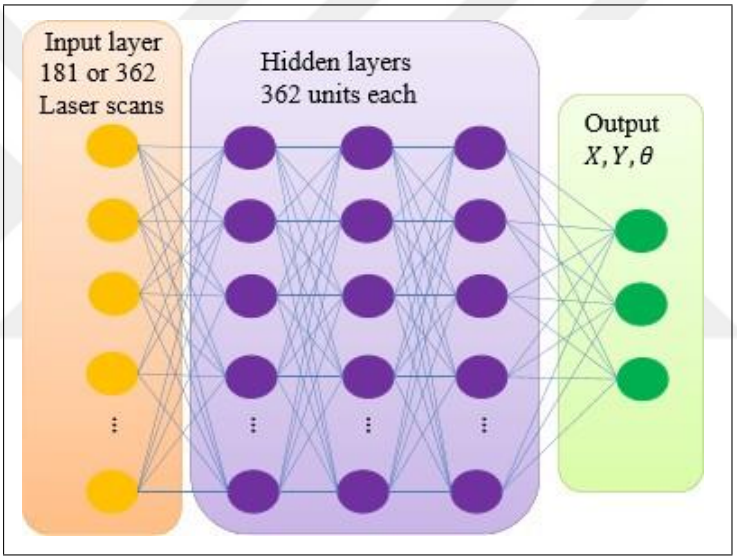
Source: (DeTone et al., 2017).

2.2.2 Na¨ıve Localization

Instead of taking the place of an Individual component of the localization systems other models can replace the full localization process by creating a model that takes a sensor's input and presents the location and orientation of a robot.

In (Tananaev et al., 2016) Na¨ıve approach was taken by creating the model shown in figure 2.8 with a laser scan input as a vector of 181 elements each representing one degree, that go into multi dense layers to output the location as regression, it was trained on encoders labels.

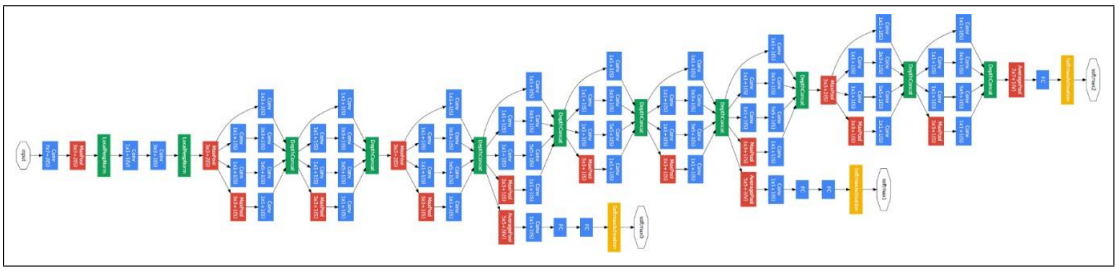
Figure 2.8: Laser scan based localization model.



Source: (Tananaev et al., 2016).

PoseNet (Kendall et al., 2015) is an outdoor camera relocalization system that receives an RGB Image and outputs the 3D location and orientation of the camera. It is based on transfer learning where they use GoogleLeNet.

Figure 2.9: The architecture of GoogleLeNet.



Source: (Szegedy et al., 2015).

GoogleLeNet (Szegedy et al., 2015) is a classification network that consists of inception layers that composed of multi CNN layer shown in figure 2.9 above with the purpose of image recognition, modifying this network by adding a dense layer before replacing the classification layer with a regression layer.

The model is trained to regress 6 DOF camera pose by taking a 224x224 RGB images and using structure from motion as training labels. Testing error was 2m and 6°accuracy for large scale outdoor scenes and 0.5m and 10 ° accuracy indoors.

3. THEORETICAL BACKGROUND

In this chapter we go through a theoretical background on visual SLAM and study some examples to provide a better understanding, then we explain the fundamental principles behind Deep learning to provide enough knowledge to go through the next chapters.

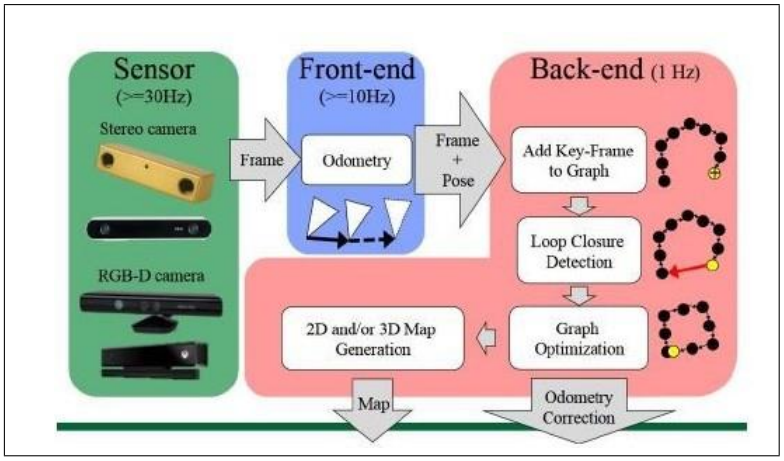
3.1 V-SLAM

Visual SLAM algorithms use sensor observations to create a map and localize in it, most used sensors are RGB-D cameras and stereo cameras, other additional sensors can be combined such as Lidars and odometers. RTAB-Map and ORB-SLAM are considered state-of-the-art V-SLAM algorithms.

RTAB-Map

RTAB-Map (Real-Time Appearance-Based Mapping) is an RGB-D graph-based SLAM approach based on an incremental appearance-based loop closure detector.

Figure 3.1: Main stages of RTAB-Map.



Source: (Labbé, 2015).

Figure 3.1 above shows the stages of RTAB-Map. Sensor measurements are provided by the main sensor such as stereo or RGB-D cameras that provide RGB images and depth information.

The purpose of the front-end stage is to process the sensor's data and the geometric constraints between the successive RGB-D extracted frames. The geometric relationships between the robot and the environment are extracted in this stage and it requires an odometry, this can be encoder + IMU or Visual odometry which is used in determining the position and orientation of a robot by analyzing the associated camera images and detecting the features using a feature detection algorithm. Visual odometry steps are:

(a) Take images at every timestamp.

(b) Detect features using a certain algorithm (e.g. SIFT, SURF, MSER, BRISK) between the images at $t = 0$ and $t = 1$ and match them.

(c) Use disparity map to calculate position of features detected in the previous step.

(d) Select a subset of points from the point cloud obtained that all matches are compatible.

(e) Estimate the position and orientation matrices from the inliers that were detected.

The resulting transformation is rendered using local bundle adjustment on features of all key frames in the Feature Map.

The purpose of the back-end stage is to focus on solving the accumulated error and on detecting the loop closure. In the previous step the visual odometry, which is based on relative measurements, will cause an accumulated error. The key-frames are selected by pre-defined parameters and added to the current frame which is stored in the Loop Detection buffer. While a new key-frame is inserted, it is compared to all previous key-frames in the Loop Detection buffer.

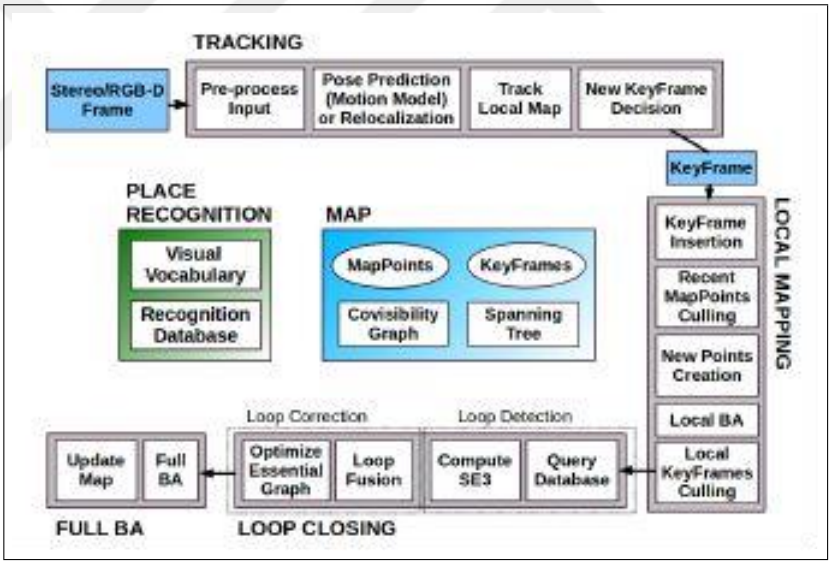
For global loop closure detection, the bag-of-words approach is used. When a loop closure hypothesis reaches a pre-defined threshold, a loop closure is detected.

Finally, when a loop is detected, the global poses are optimized by the g2o optimization algorithm. The g2o (general graph optimization) provides an open-source C++ framework for optimizing graph-based nonlinear error functions using least squares. It is applied in the Visual SLAM system to obtain a 3D global consistency map (Gurel, 2018).

ORB-SLAM2

ORB SLAM 2 (Oriented fast Rotated Brief SLAM) is a graph based optimization approach for stereo and RGB-D cameras build on monocular ORB SLAM 1.

Figure 3.2: Main stages of ORB-SLAM2.

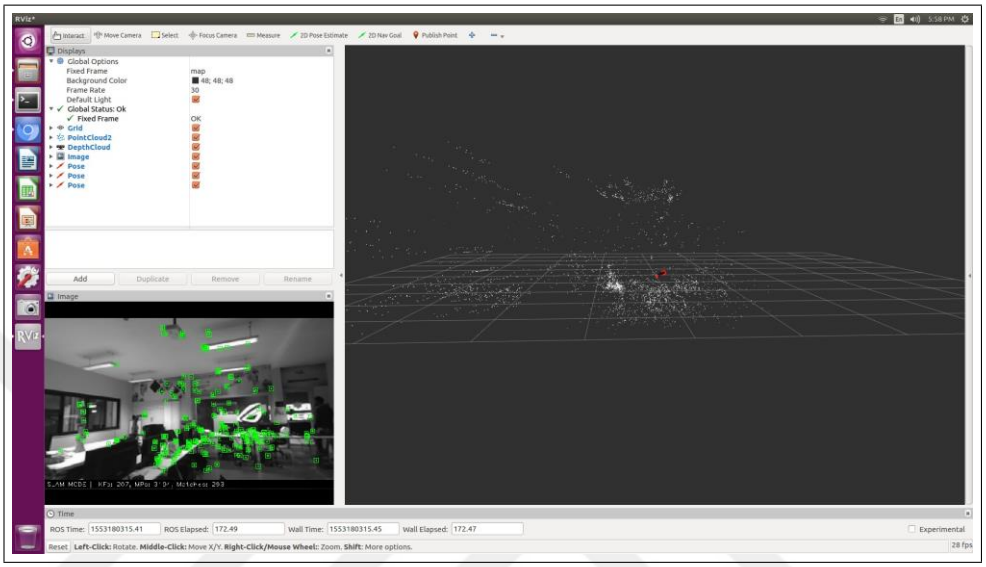


Source: (Mur-Artal & Tardos, 2017).

Figure 3.2 shows the 3 main stages of ORB-SLAM2, the first one is Tracking, the main purpose of this stage in ORB-SLAM2 is to localize the robot within every frame and decide when to insert a new keypoint or keyframe to the mapping stage at which the input images are discarded and all system operations are based on these features.

Key frames store the camera poses and ORB features shown in figure 3.3 in the frame. Each node is a keyframe and an edge between two keyframes exists if they share observations of the same map points (store the 3D positions and ORB descriptions) and that creates a Covisibility Graph.

Figure 3.3: Feature detection in ORB-SLAM2.



One of the main benefits of using stereo or RGB-D cameras is that, by having depth information from just one frame, there is no need for a specific Structure From Motion (SFM) initialization as in the monocular case.

The second stage is Local Mapping and the main purpose of this stage is to Process new keyframes and performs local BA to optimize the map points and the poses of the keyframes.

The Last stage is Loop closing, which is the act of correctly asserting that a robot has returned to a previously visited location. It is performed in two steps, firstly a loop has to be detected and validated, and secondly the loop is corrected by optimizing a pose-graph.

For stereo cameras the stereo/depth information makes scale observable and the geometric validation and pose-graph optimization no longer require dealing with scale drift that

occurs when using mono cameras.

After that, full BA optimization after the pose-graph to achieve the optimal solution that works in parallel with creating and detecting maps.

3.2 NEURAL NETWORKS

Machine learning is programming computers to optimize a performance criterion using example data or past experience (Parsons, 2010). It highly relays on large scale data that the model learns from. There are three types of learning:

(a) Supervised learning in which all samples of the data are labeled.

(b) Semi-supervised which has labeled and unlabeled data.

(c) Unsupervised learning that has no labeled data.

Machine learning tasks vary as the data and the uses are different based on the intended application, generally, models have the objective of classification where the model predicts the class of the input or regression where the model predicts a value of a given continuous valued variable.

An artificial neuron is the fundamental unit of a neural network, it represents a processing unit in the human brain, a perceptron introduced in (Rosenblatt, 1962) is the basic processing model in machine learning, it consists of an input layer, weights, net sum and an activation function that determines the output. If the purpose of the model is classification a sigmoid function is usually applied to categories the output.

Figure 3.4: A simple perceptron.

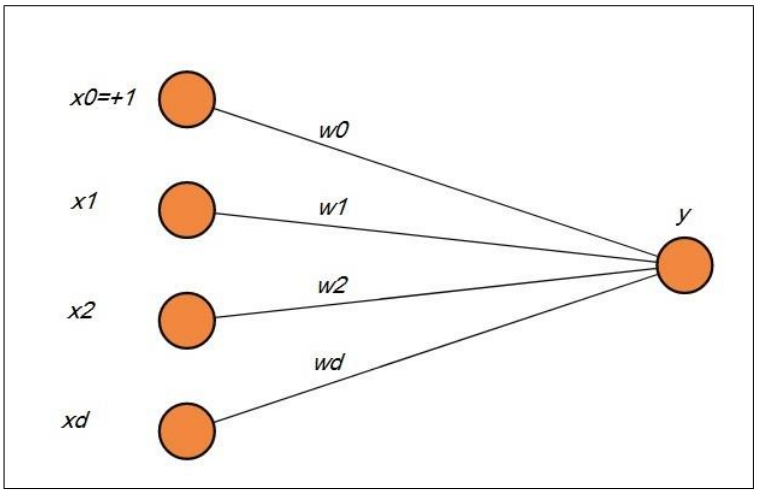


Figure 3.4 shows the architecture of a perceptron, every neuron x has a weight w that affect the output y as demonstrated in (3.1).

$$y = \sum_{j=1}^{d} w_j x_j + w_0 \tag{3.1}$$

The learning process is an iterative process that uses an optimization algorithm. The most common iterative optimizer is gradient descent, other algorithms such as Adagrad, Adam, RMSprop and others can also be used. The learning process starts with assigning random weights to each neuron, and at each step it updates the weights to minimize the error. (3.2) describes the update phase using the gradient descent optimizer, r is the desired output, y is the actual output and η is the learning factor.

$$\Delta w_{ij}^t = \eta(r_i^t - y_i^t)x_j^t \tag{3.2}$$

Adaptive Moment Estimation (Adam) computes adaptive learning rates for each parameter, we compute the first moment (the mean) and the second moment (the uncentered variance) of the gradients m_t and σ^2 respectively as follows in (3.3) and (3.4):

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \tag{3.3}$$

$$\sigma^2_t = \beta_2 \sigma^2_{t-1} + (1 - \beta_2) g_t^2 \tag{3.4}$$

Where β_1 and β_2 are the decay rates. They counteract these biases by computing bias-corrected first and second moment estimates as shown in (3.5) and (3.6):

$$\hat{m}_t = m_t / (1 - \beta_1^t) \tag{3.5}$$

$$\hat{\sigma^2}_t = \sigma^2_t / (1 - \beta_2^t) \tag{3.6}$$

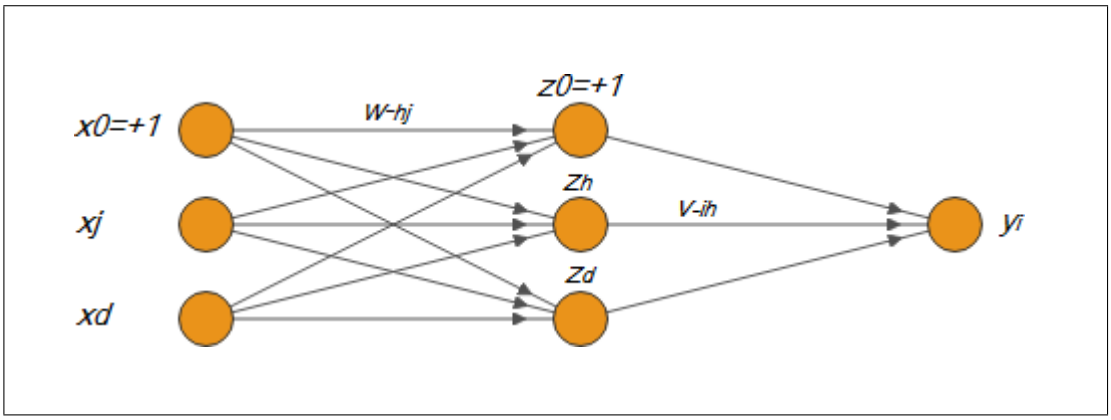
and they are used to update the weights by:

$$\Delta w_{ij}^{\,t} = \frac{\eta}{\sqrt{\hat{\sigma^2}_t + \epsilon}} \hat{m}_t \tag{3.7}$$

while ϵ in (3.7) is a smoothing term to avoid division by zero.

Deep learning is a subfield of machine learning, the model is considered to be in the field of deep learning if it has multiple hidden layers, the basic processing unit is a multi layer perceptron shown in figure 3.5.

Figure 3.5: multi-layer perceptron.



The learning process in multi-layer perceptron is described in (Rumelhart et al., 1986) as two phases process:

(a) Forward phase that computes the output as shown in (3.8).

$$y_i = \sum_{h=1}^{d} v_{ih} z_h + v_{i0} \tag{3.8}$$

(b) Back-propagation that updates both weights in every layer shown in (3.9) and (3.10) using gradient descent.

$$\Delta v_h = \sum_t (r^t - y^t) z_h^t \tag{3.9}$$

$$\Delta w_{hj} = -\eta \frac{\partial E}{\partial w_{hj}} = -\eta \sum_t \frac{\partial E}{\partial y^t} \frac{\partial y^t}{\partial z_h^t} \frac{\partial z_h^t}{\partial w_{hj}} = \eta \sum_t (r^t - y^t) v_h z^t (1 - z^t) x^t \tag{3.10}$$

In the next chapter, we will dive in some deep learning layers and specifics explaining the proposed architecture for this thesis.

4. METHODOLOGY

In this chapter we propose a method to localize an indoor mobile robot using a Deep Learning approach based on camera input in order to get a direct 2D location and orientation of the robot.

Deep learning is a part of machine learning where the created model is structured of multiple layers of neural networks to simulate the architecture of a brain. Deep learning approaches are based on a model with a unique architecture, we use the form of supervised learning, which is training a model on multiple inputs given their consecutive outputs.

Multiple data are required to train the neural network model. In our case, images and their exact location and orientation are required to be collected.

4.1 DATA COLLECTION

For data collection two ROS nodes were created, both RGB and Depth images are recorded with their timestamps with the frequency of the camera, the timestamp is described as seconds and nanoseconds in the simulation or as Unix time in real implementation both provided by ROS, and each RGB image is saved in a certain path with a unique label that is also saved in a CSV file with its timestamp, and the same goes for the depth images.

The encoders' data are also recorded using ROSBAG tool that saves the odometry topic -which has the timestamp and the robot's position and orientation- into a bag file.

A launch file was created to run the RGB image node, the Depth image node and the ROSBAG node to record the RGB images, Depth images and the encoders' data while the robot is moving in the environment.

The bag file containing the encoders' data is then converted to a CSV file using ros-

bag_to_csv package, the Odometry file is then modified by transforming the orientation data from quaternion to Euler and then from radians to degrees in the range of (-180°,180°).

The timestamps of the images and the Odometric data are then synchronized and a document of the timestamps with the images' filenames, robot's location (X,Y) and orientation (Ψ) is created.

4.2 MODEL ARCHITECTURE

The most common approach when building a neural network is to start with a rough guess based on prior experience or based on networks used for similar problems and then change the parameters in order to try some variations, and check the performance carefully before picking the final model.

As mentioned before Deep learning has multiple layers that can be used in the creation of the model, we explore two of these layers that we need to use.

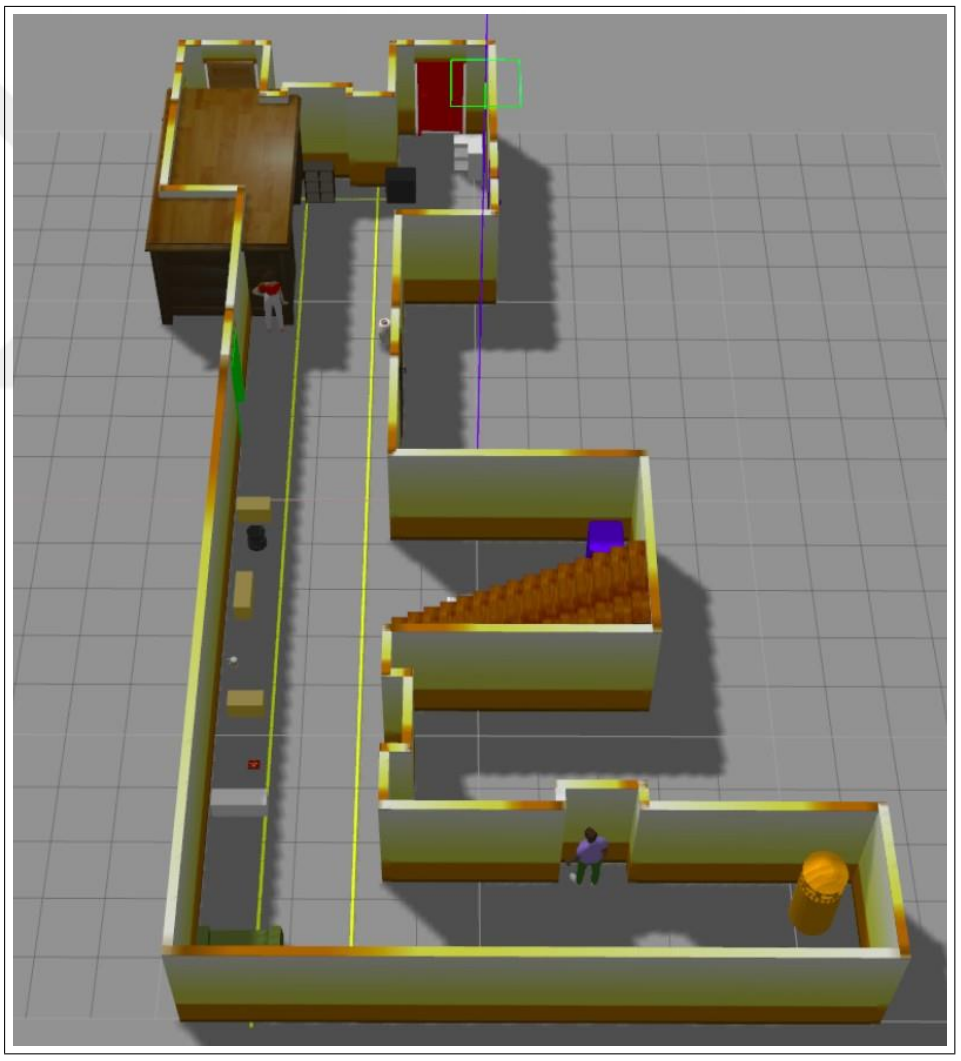
1. CNN layer: Convolutional Neural Network is a network that studies an image and creates a feature map by multiplying the image by different numbers of filters with different sizes known as kernels. CNN layer is usually followed by max pooling, which reduces the dimensions of the matrix by representing every (n, n) pixels by the maximum pixel, and so allowing for assumptions to be made about features contained in the sub-regions of the image.
2. FFNN layer: Feed Forward Neural Network or Dense layer is a layer that has a different number of neurons that simulates a brain neuron with a constant weight for each neuron.

In order to create the model that provides a high localization accuracy, we try designing different models by changing or tuning the controllable parameters and analyze them.

And for that purpose all the training and validation was done using TensorFlow-GPU 1.9 on ASUS X550VX with a 4GB NVIDIA GEFORCE GTX 950M, the simulation was in Gazebo 7 on ROS Kinetic.

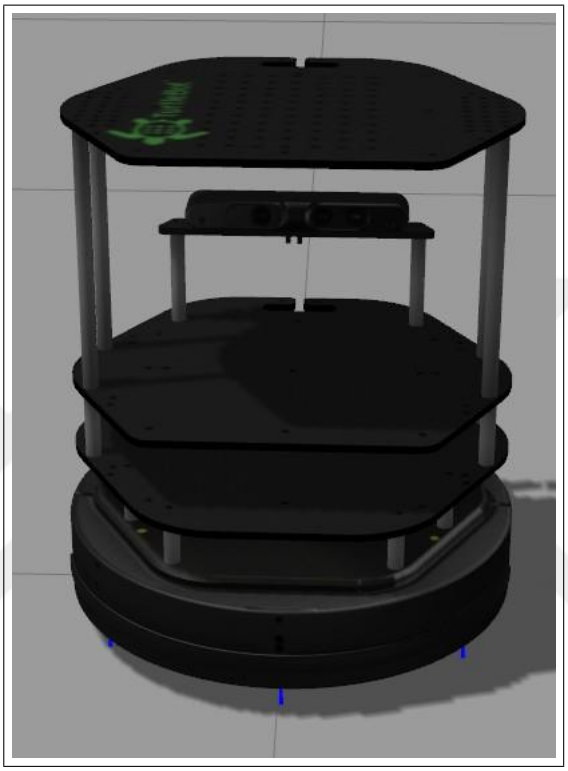
Gazebo is a simulation software that is supported by ROS, Gazebo offers the ability to accurately and efficiently simulate populations of robots in complex indoor environments. An indoor environment was created in Gazebo to simulate an office environment with an area of (18 x 1.5)m^2 as shown in figure 4.1 for the robot to navigate in with different obstacles along the way.

Figure 4.1: Office environment simulation model.



In the simulations TurtleBot2 shown in figure 4.2 an Open-source robot development kit was used, it's a differential drive robot with the support for ROS, which allows us to use it in the simulations and create realistic scenarios. Attached to its base is a Kinect Camera that provides the RGB and Depth images.

Figure 4.2: TurtleBot2 in simulation.



The office environment simulation was used to create the model, design and tune the parameters and options of the model. The robot moved in the environment for approximately 45 min recording both the RGB, Depth and Odometry data from the encoders. A total of 50947 RGB and 50947 depth images with their corresponding 2D location and orientation were obtained.

The data are then shuffled to ensure that each data point creates an independent effect on the model, to increase the efficiency of the training process. Taking non-sequential images and labels will lead to a better result than sequential data to reduce variance, less

overfitting and remain general. And then divided into a training set (90%) to train the neural network model and a validation set (10%) to test the accuracy of the model.

Before creating the model, based on the input of the system we have 3 different layouts or models:

1. RGB images as an input.
2. Depth image as input.
3. Combined input of both RGB and Depth images.

4.2.1 RGB Based Model

The images captured by the camera in the simulation have a resolution of (640X480) (W x L), each image is normalized and converted to (320 x 240) for memory's sake.

The features of the images are then detected using multi layers of CNN to produce a feature map of the image. To analyze how many CNN layers we need, how many filters do we require and the size of the filters. A one epoch training process with a batch size of 8 was applied to various models with different CNN layers using ADAM (Adaptive Moment Estimation) optimizer and the MAE (Mean Absolute Error) described in (4.1) as the loss function to the three outputs. The performances are evaluated by the validation loss with a condition of having a yaw angle error less than 10° and the minimum euclidean error possible.

The mean absolute error for the loss function given T the total number of images is described as

$$MAE = \frac{\sum_{n=1}^{T} (|X - X^{J}| + |Y - Y^{J}| + |\Psi - \Psi^{J}|)}{3 * T} \tag{4.1}$$

Generally, mobile robot localization error is described as the mean Euclidean error in meters (4.2) and mean yaw angle error in degrees (4.3)

$$Mean\ Euclidean\ error = \frac{\sum_{n=1}^{T} \sqrt{(X - X^J)^2 + (Y - Y^J)^2}}{T} \tag{4.2}$$

$$Yaw\ error = \frac{\sum_{n=1}^{T} \Delta\Psi}{T} \tag{4.3}$$

$$\Delta\Psi = \begin{cases} ||\Psi - \Psi^J| - 360| & \text{if } \Psi * \Psi^J < 0 \ \&\ \Delta Y > 180 \\ |\Psi - \Psi^J| & \text{otherwise} \end{cases} \tag{4.4}$$

Figure 4.3: Robot's orientation.

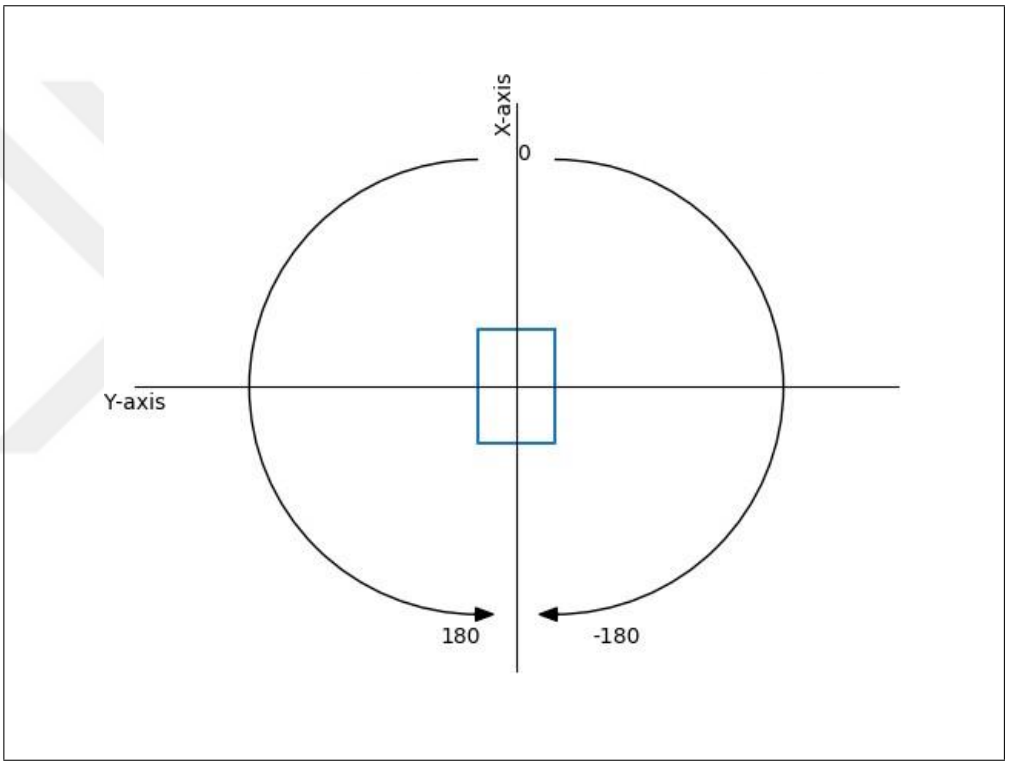


Figure 4.3 presents the orientation of the robot between angles (-180°,180°) and so the calculation of the yaw angle error is referred to as in (4.4) to avoid any miss calculations.

X, Y and Ψ are the ground truth location and orientation and X^J, Y^J and Ψ^J are the predicted location and orientation.

We start our assumptions with CNN layers that have 64 kernels with the size of 3*3 followed by a fixed number of dense layers with fixed parameters to analyze the CNN layers and variables.

Table 4.1 The effect of CNN layers quantity on accuracy

Number of CNN layers	Training error	Validation error
2	$1.0m$, 7.7°	$1.0m$, 7.8°
3	$0.88m$, 8.9°	$0.89m$, 8.9°
4	$1.0m$, 9.1°	$1.0m$, 9.1°
5	$0.87m$, 9.4°	$0.87m$, 9.7°
6	$1.2m$, 21°	$1.2m$, 20°

According to table 4.1 above the best validation loss is the model that has 3 CNN layers.

Table 4.2 The effect of Kernel size in each CNN on accuracy.

Kernel Size	Training error	Validation error
$3 * 3$	$0.88m$, 8.9°	$0.89m$, 8.9°
$4 * 4$	$0.84m$, 11°	$0.82m$, 10°
$5 * 5$	$0.85m$, 9.8°	$0.86m$, 9.9°
$6 * 6$	$1.6m$, 17°	$1.6m$, 17°

According to table 4.2 above the best validation loss is the model that has a kernel size of 3*3.

Table 4.3 The effect of filters quantity in each CNN on accuracy.

Number of the filters	Training error	Validation error
64	$0.88m$, 8.9°	$0.89m$, 8.9°
32	$1.3m$, 8.0°	$1.3m$, 7.8°

And according to table 4.3 the best validation loss is the model that has a number of the filter size of 64.

After establishing the variables of the CNN layers we can build our model on the basis that we have extracted the best feature map of the images.

Next, we need to provide a link between the feature maps and the output of our model, we use FFNN or dense layers to provide an understanding of the map to the model, and

each dense layer is constructed of a number of neurons that set weights to memorize the map.

Since the output of the model is three neurons representing X^J,Y^J and Ψ^J, the proposed architecture of the dense layers has the shape of an encoder, in an encoder each layer has number of neurons less than the previous layer, it aims to leverage this ability of neural networks to learn efficient representations or to map raw inputs to representations.

We explore different analyses in the range that our memory allows us to.

Table 4.4 The effect of dense layers quantity on accuracy.

Dense Layers	Neurons	Training error	Validation error
0	(0)	$2.1m$, 23°	$2.1m$, 22°
1	(64)	$0.70m$, 14°	$0.70m$, 14°
2	(128,64)	$0.88m$, 8.9°	$0.89m$, 8.9°
3	(265,128,64)	$0.99m$, 7.0°	$1.0m$, 7.2°

According to table 4.4 above the best validation loss is the model that has 2 dense layers.

Another analysis is the number of the neurons in each dense layer

Table 4.5 The effect of neurons quantity in each dense layer on accuracy.

Number of the neurons	Training error	Validation error
64, 64	$1.3m$, 11°	$1.3m$, 11°
128, 128	$1.0m$, 7.9°	$1.1m$, 8.2°
128, 64	$0.88m$, 8.9°	$0.89m$, 8.9°

According to table 4.5 the best validation loss is the model with a first dense layer of 128 neurons and a second dense layer of 64 neurons.

We also test expand and contract technique where we add a dense layer with neurons more than the previous one and then add a dense layer with less neurons. But according to table 4.6 it did not enhance the accuracy.

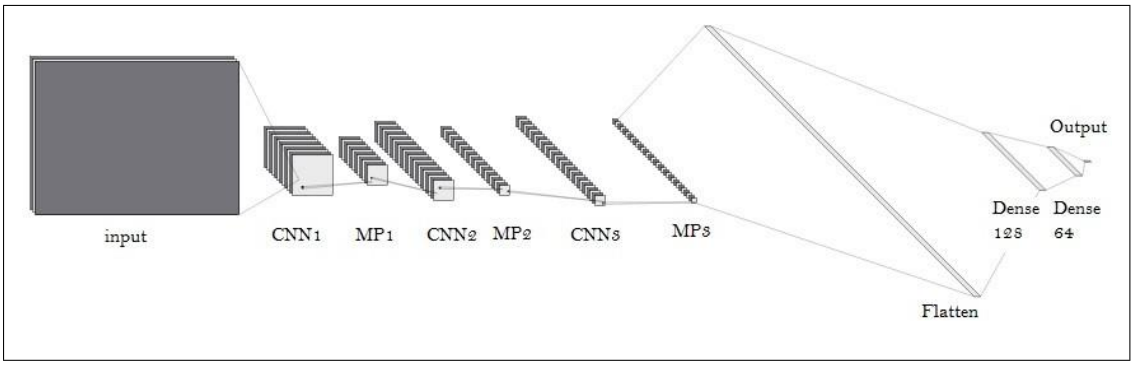
Table 4.6 Expand and contract effect on accuracy.

Number of neurons	Training error	Validation error
128, 64	$0.88m$, 8.9°	$0.89m$, 8.9°
64, 128, 64	$1.0m$, 7.9°	$1.0m$, 8.5°

Dense layers use an activation function to train these layers and choose the weights of the neurons, regression functions as Relu, Tanh, Selu, Elu and Linear were examined and the most suitable functions are Relu function for CNN layers, and as this is a regression model linear function is used for the output layer.

We construct our model based on the previous results as shown in figure 4.4:

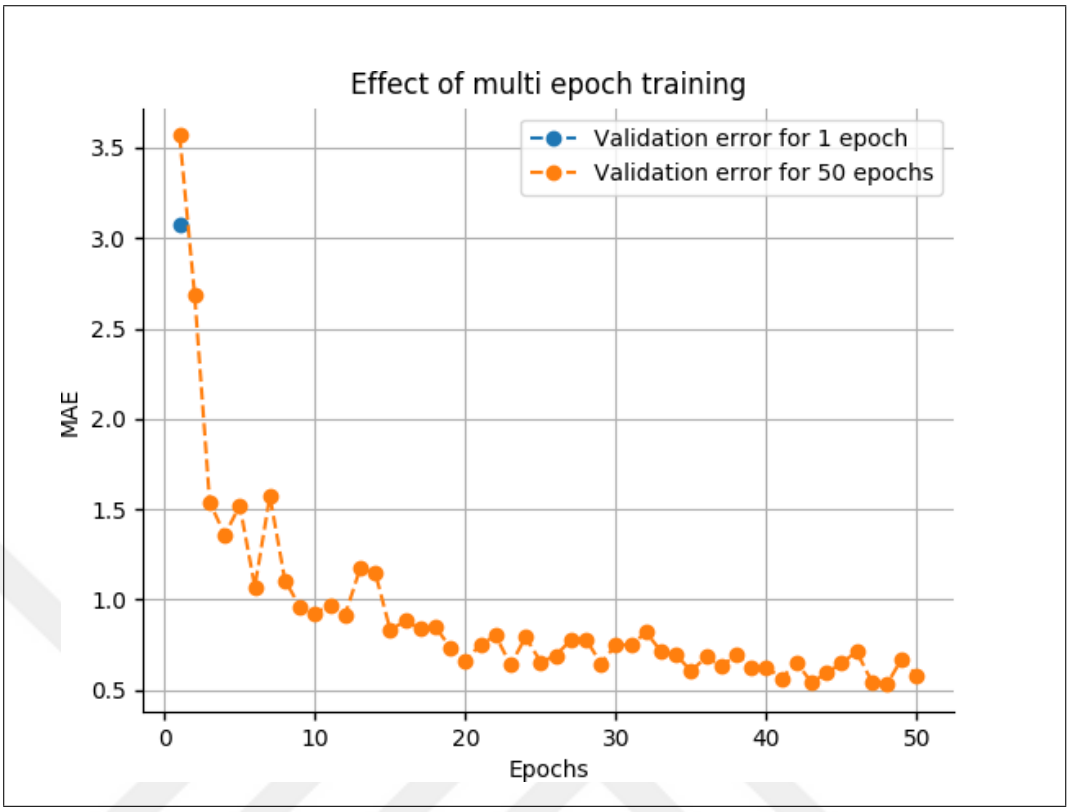
Figure 4.4: RGB based model.



When using a small dataset, we use a small number of epochs in order to not cause over fitting, but in my dataset we need to pass the full dataset multiple times to the same network, keep in mind that we are using ADAM which is an iterative process so updating the weights with single pass or one epoch is not enough.

At the same time we don't want it to over fit to the point that the accuracy of the training data will keep improving and the validation accuracy will decrease, so I use a checkpoint to save the model with the best validation accuracy, and use early stopping to stop the training process when the validation accuracy does not improve for a certain number of epochs.

Figure 4.5: Validation loss from 1 epoch vs 50 epochs of training.



As figure 4.5 above and table 4.7 show, the loss in 1 epoch is much more than the loss of 50 epochs of training.

Table 4.7 Epochs effect on accuracy.

Epochs	Training error	Validation error
1	$0.88m$, 8.9°	$0.89m$, 8.9°
50	$0.28m$, 1.1°	$0.28m$, 1.4°

After many attempts of training the model, it was noticed that the ADAM is trying to change the weights in order to reduce the yaw angle error where its loss weight in the loss function (MAE) is more than the weight of X and Y because they are in meters, so I changed the distance units from m to cm for better accuracy and it enhanced the results.

We can observe that when the sensitivity is in cm the model learns faster than in m and

yields to a better result according to table 4.8.

Table 4.8 Sensitivity effect on accuracy.

Sensitivity	Training error	Validation error
m	$0.28m$, 1.1°	$0.28m$, 1.4°
cm	$0.074m$, 2.1°	$0.076m$, 2.4°

4.2.2 Depth Images Model

The same loss function and approach were taken in developing this model, in addition to that, the model was derived on the same map using only the depth images.

Table 4.9 Effect of CNN layers quantity on accuracy.

CNN Layers	Training error	Validation error
2	$1.1m$, 24°	$1.1m$, 25°
3	$1.2m$, 25°	$1.2m$, 25°
4	$1.2m$, 26°	$1.2m$, 26°
5	$1.4m$, 25°	$1.3m$, 25°

Table 4.10 The effect of Kernel size in each CNN on accuracy.

Kernel Size	Training error	Validation error
$2 * 2$	$0.99m$, 25°	$0.99m$, 26°
$3 * 3$	$1.1m$, 24°	$1.1m$, 25°
$4 * 4$	$1.4m$, 26°	$1.4m$, 26°
$5 * 5$	$1.4m$, 26°	$1.4m$, 26°
$6 * 6$	$1.1m$, 28°	$1.1m$, 28°

Table 4.11 The effect of filters quantity in each CNN on accuracy.

Number of the filters	Training error	Validation error
64, 64	$0.99m$, 25°	$0.99m$, 26°
32, 32	$1.2m$, 22°	$1.2m$, 23°
64, 32	$1.1m$, 32°	$1.1m$, 24°

Tables 4.9, 4.10 and 4.11 above show that having a model with 2 CNN layers with a kernel size of 2*2 and 64 filters grant us the least errors.

Table 4.12 The effect of dense layers quantity on accuracy.

Dense Layers	Neurons	Training error	Validation error
0	(0)	$3.3m$, 41°	$3.3m$, 41°
1	(64)	$1.2m$, 32°	$1.3m$, 33°
2	(128,64)	$0.99m$, 25°	$0.99m$, 26°
3	(256,128,64)	$0.88m$, 24°	$0.90m$, 25°
4	(512,256,128,64)	$1.0m$, 24°	$1.0m$, 25°

Table 4.12 shows that adding three dense layers after the CNN layers yield to a better accuracy.

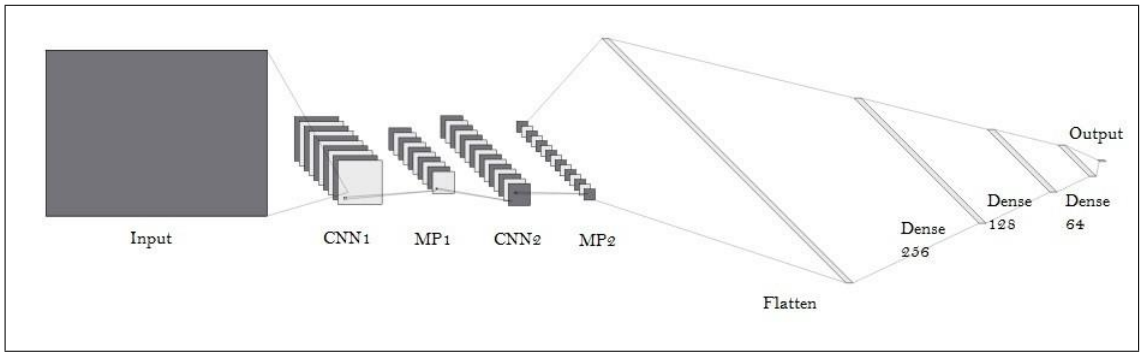
Table 4.13 Epochs effect on accuracy.

Epochs	Training error	Validation error
1	$0.88m$, 24°	$0.90m$, 25°
50	$0.53m$, 8.1°	$0.55m$, 9.1°

After training the model for 50 epochs the error is decreased as presented in table 4.13.

And so following our previous results the best depth model is shown in figure 4.6:

Figure 4.6: Depth based model.



Changing the sensitivity from m to cm yields to the results in table 4.14

Table 4.14 Sensitivity effect on accuracy.

Sensitivity	Training error	Validation error
m	0.53*m*, 8.1°	0.55*m*, 9.1°
cm	0.25*m*, 4.9°	0.25*m*, 4.9°

4.2.3 Combined Input of Both RGB and Depth Images Model

In this model we explore the idea of having both RGB and Depth images as inputs to the model at the same time in order to have more features and information about the environment and test the effect of that on the output.

Tested two approaches:

A- Using a combination of both models derived previously, we produce our first model.

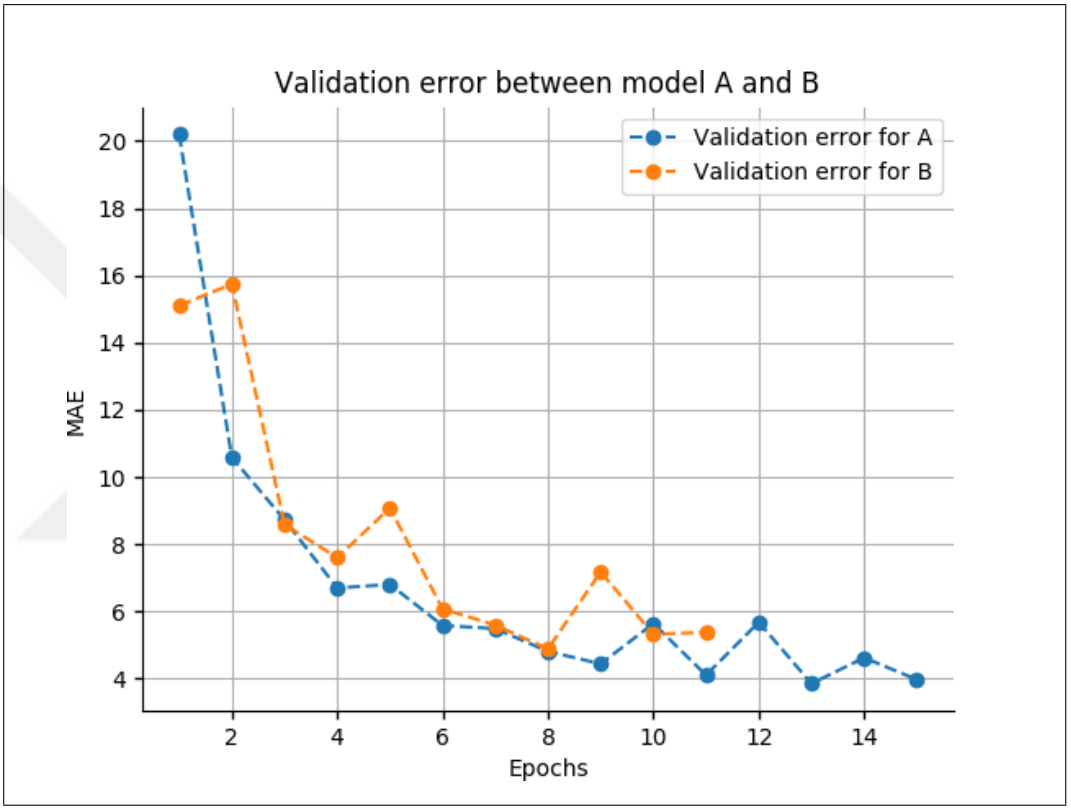
B- Analyze modifying the model by adding a dense layer of 64 neurons after the combination of both models.

Table 4.15 Combined models accuracies.

Model	Training error	Validation error
A	$0.069m$, 2.4°	$0.073m$, 2.4°
B	$0.095m$, 3.5°	$0.097m$, 3.6°

The results in table 4.15 above were derived using cm as distance labels and using 15 epochs of training and shown in figure 4.7.

Figure 4.7: Validation loss during multi epoch between model A and model B.



4.2.4 Comparing The Best Models Based on The Input

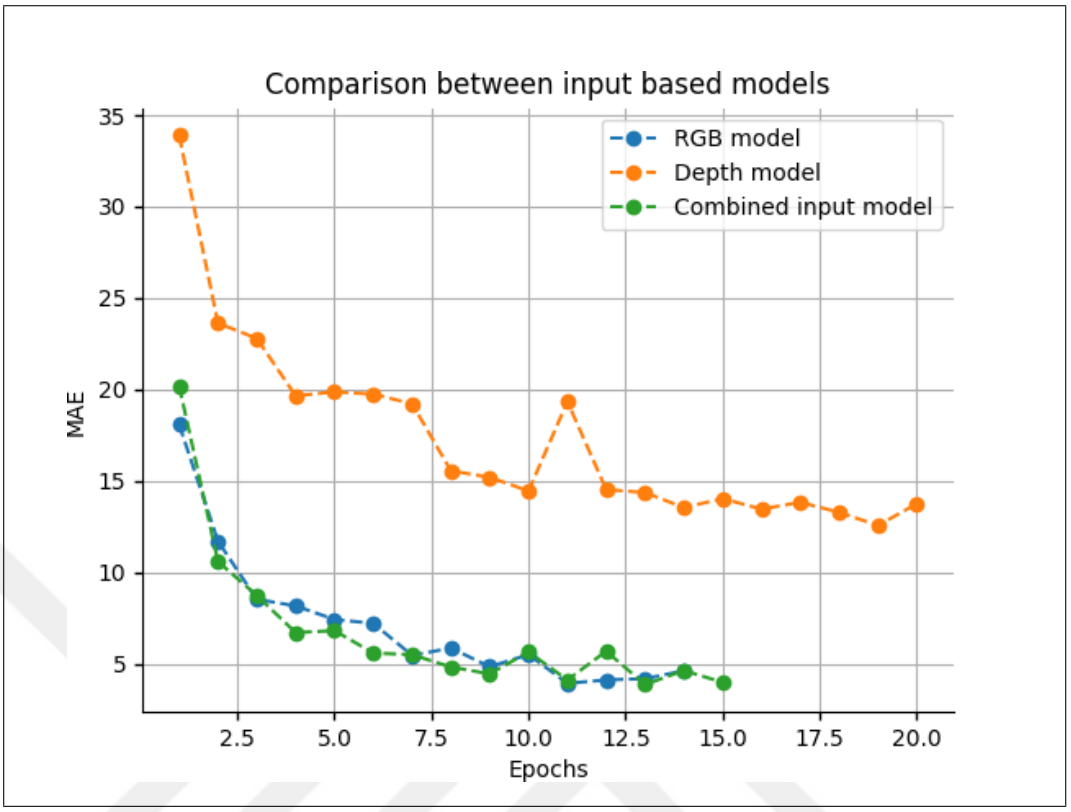
Table 4.16 Comparing the best models.

Model	Training error	Validation error
RGB	$0.074m$, 2.1°	$0.076m$, 2.4°
Depth	$0.25m$, 4.9°	$0.25m$, 4.9°
RGB + Depth	$0.069m$, 2.4°	$0.073m$, 2.4°

As shown in table 4.16 and figure 4.8 RGB and RGB+Depth models have the advantage on the Depth image and that's because stereo cameras do not publish a depth image unless an obstacle is detected in a certain range for that camera, which means that if the camera is not detecting any object or the camera is too close to a wall or an object, the camera will publish a blank black image, this issue affects the localization in both the training process and the deployment.

As for the RGB+Depth input based model it shows a similar accuracy to the RGB input based model, but the first consumes more computational power and hence takes more time and space than the second model and thus choosing the RGB input based model as our proposed system saves storage, time and preferably in deployment. In addition, it allows us to use mono camera or any kind of cameras.

Figure 4.8: Validation loss during multi epochs comparison between models.



4.3 CONCLUSION

Based on the trainings and tests described in this chapter, we propose a Deep Learning model that inputs an RGB image and outputs the location and orientation of a robot.

Our system trains Convolutional Neural Networks and Feed Forward Neural Networks to regress the 2D position and orientation of a robot from a single RGB image provided by a camera attached to the robot.

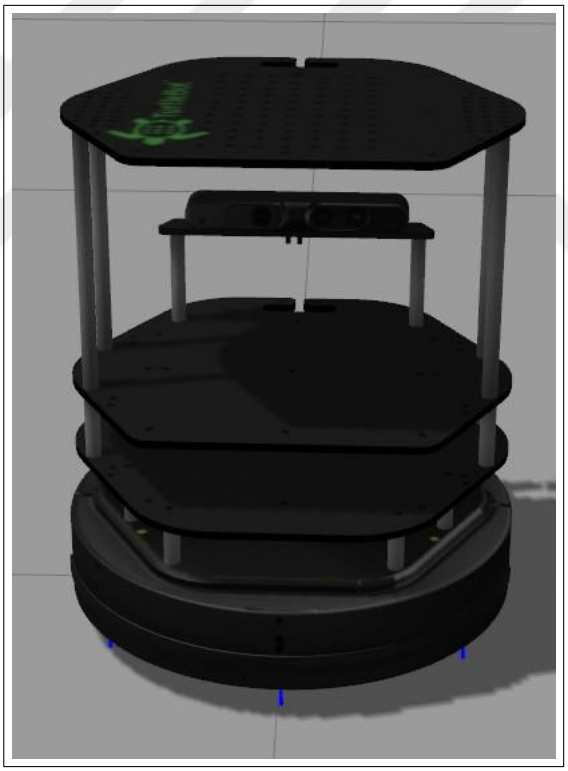
In addition the inaccuracy of this system is insignificant and does not accumulate over time, it does not consume much storage or require after-calculation optimization.

5. SIMULATIONS

In this chapter we present the Implementation of our localization method in simulations. For each environment, we introduce the layout of the simulation, implement both our model and RTAB-Map and display the results.

Using Gazebo 7 simulation program, we test our system in two different environments using TurtleBot2 shown in figure 5.1 which is an open source differential drive robot with a Kinect camera attached to its base, operating in ROS Kinetic on ASUS X550VX with a 4GB NVIDIA GEFORCE GTX 950M.

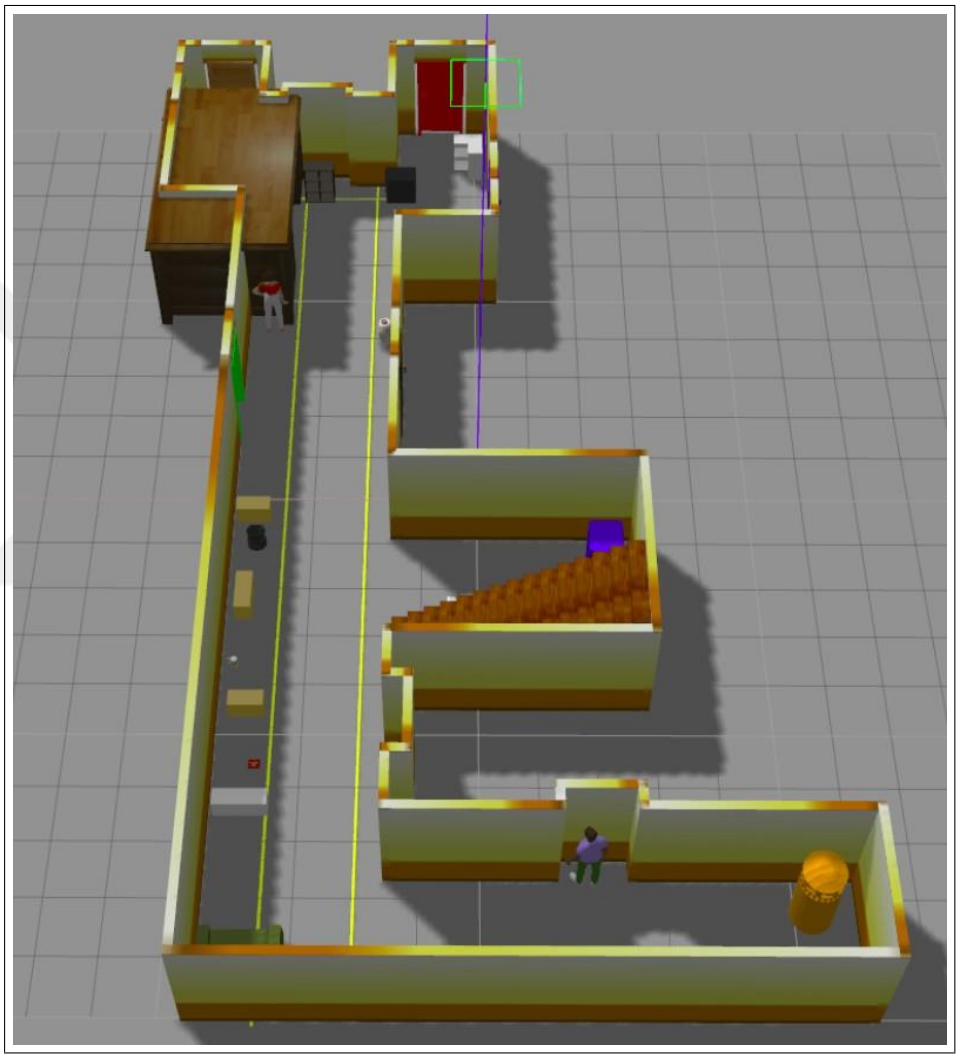
Figure 5.1: TurtleBot2 in simulation.



5.1 OFFICE ENVIRONMENT

The first map presented in figure 5.2 simulates an indoor office environment with an area of (18X1.5)m^2 for the robot to navigate in; it contains various obstacles that generate many features.

Figure 5.2: Office environment simulation model.



5.1.1 Our Model

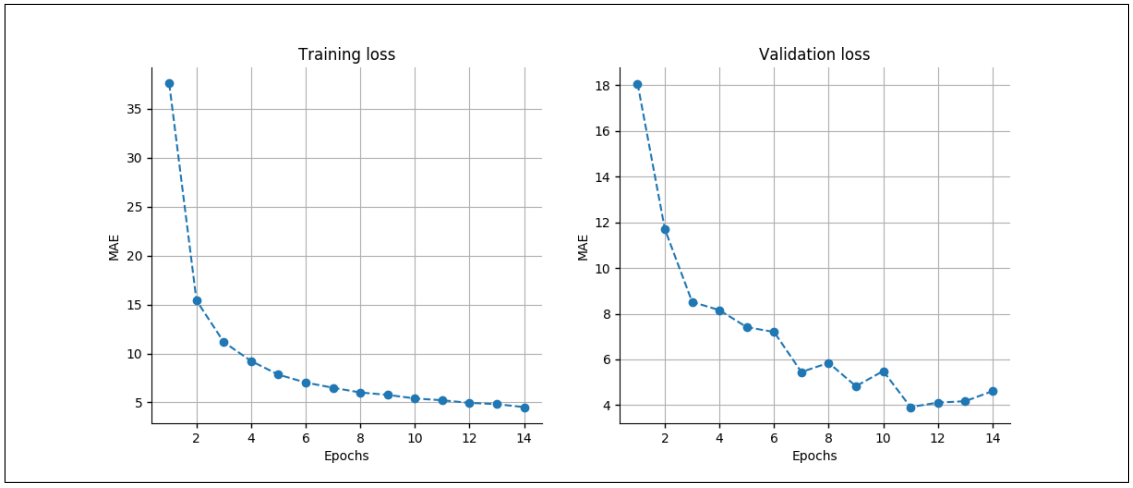
Our Deep learning based model consists of two stages: the first is data collection which produces a document containing each image's filename and the robot's location and orientation in that image, the second is training the model by feeding that document to the neural network.

Our Deep learning based model is trained for both simulation on TensorFlow1.9 using Keras libraries on an ASUS X550VX with a 4GB NVIDIA GEFORCE GTX 950M.

The TurtleBot moved in the office environment for approximately 45 minutes recording RGB, Depth and Odometry data. A total of 50947 RGB images with their corresponding 2D location and orientation were obtained.

Images captured by the Kinetic camera had a resolution of (640,480) for (W x L) which was converted with a factor of 0.5 to (320,240). The system was trained with a batch size of 8 for 15 epochs on 90% of the RGB images. The other 10% were used for validation.

Figure 5.3: Training and validation loss for the office environment model.



In figure 5.3 we can observe that the model activated early stopping at epoch 14 because the validation loss did not improve for 3 consecutive epochs and the model was saved at epoch 11. Table 5.1 shows the results of our model.

Table 5.1 Validation results of our model in the office environment.

Environment	Validation error	Standard deviation
Office simulation	$0.076m$, 2.4°	$0.11m$, 9.4°

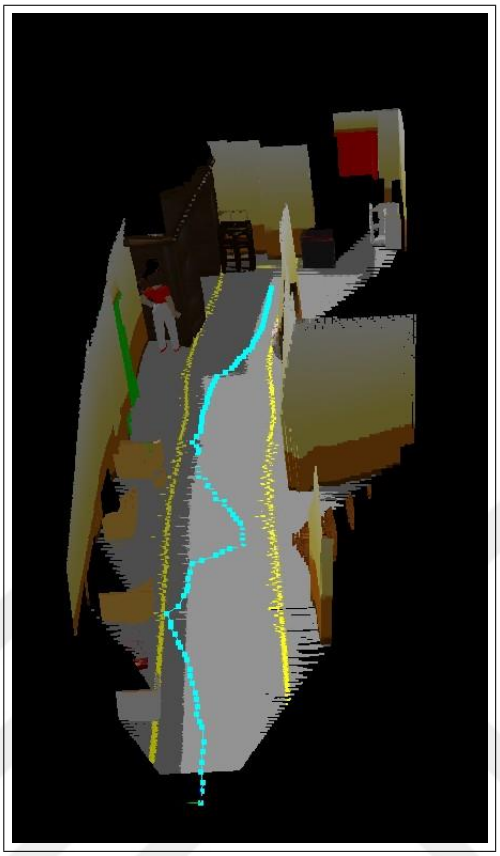
Recording data process is an essential part of our method, on the office simulation map we tested the effect of data recording, recording the data for a short time gives 1655 images where the locations on the map are only visited once and will give an accuracy of (0.20 m, 7.8°). In contrast, recording for a long time in the same map and having more images of the same locations will yield for a better result (0.076 m, 2.4°).

5.1.2 RTAB-Map

Using rtabmap-ros package on Kinect camera RTAB-Map provides Visual Odometry for the camera to map the environment and localize in it. RTAB-Map is a two stage process: the first is mapping and the second is localization.

For the office environment the map created, shown in figure 5.4, did not achieve loop closure since the path was straight forward but the drift of the visual odometry provided by RTAB-Map caused a drift in mapping as observed in the following figure and when the robot rotates 180 degrees RTAB-Map loses the visual odometry which causes the robot to map over a mapped area which diminishes the mapping process.

Figure 5.4: Office environment in RTAB-Map.



After mapping we initialize localization mode to localize the robot and move the robot around the environment and record both the wheels' odometry data and the RTAB-Map odometry. The results are shown in table 5.2.

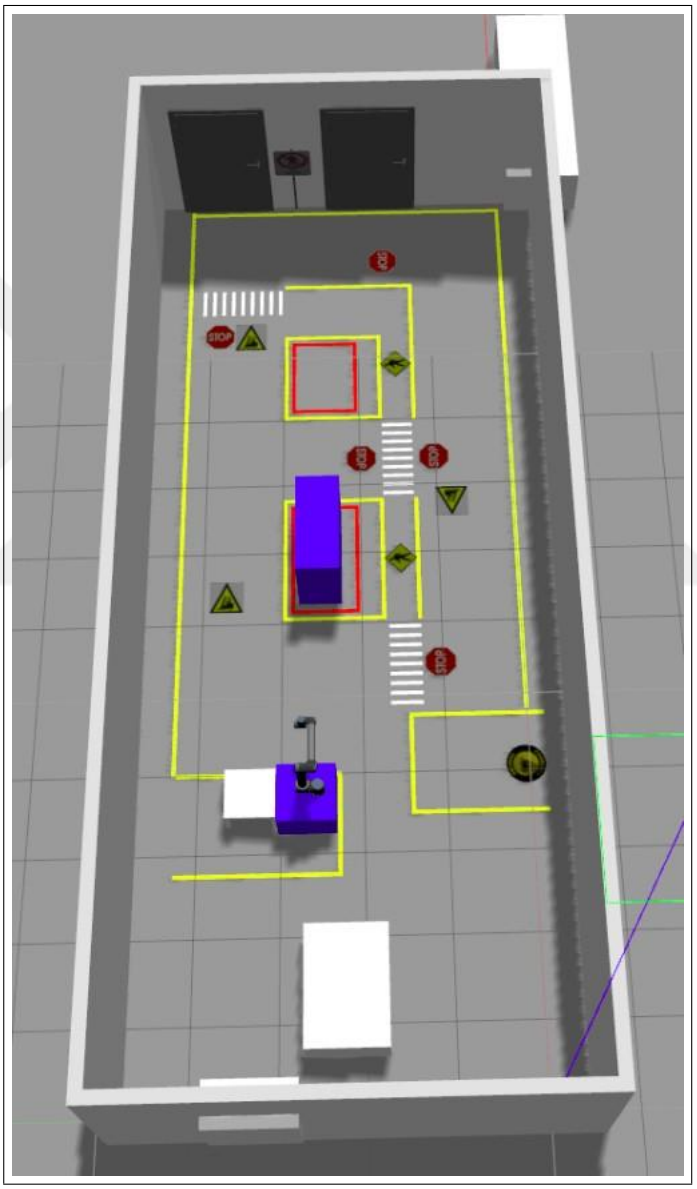
Table 5.2 RTAB-Map results for the Office environment.

Environment	Mean Error	Standard deviation
Office simulation	$1.0m$, 6.5°	$0.5m$, 3.9°

5.2 LAB ENVIRONMENT

The second map presented in figure 5.5 simulates a lab environment of a small factory with an area of (12X5)m^2, this map contains less features and has a bigger area for the robot to navigate in than the previous map.

Figure 5.5: Lab environment simulation model.

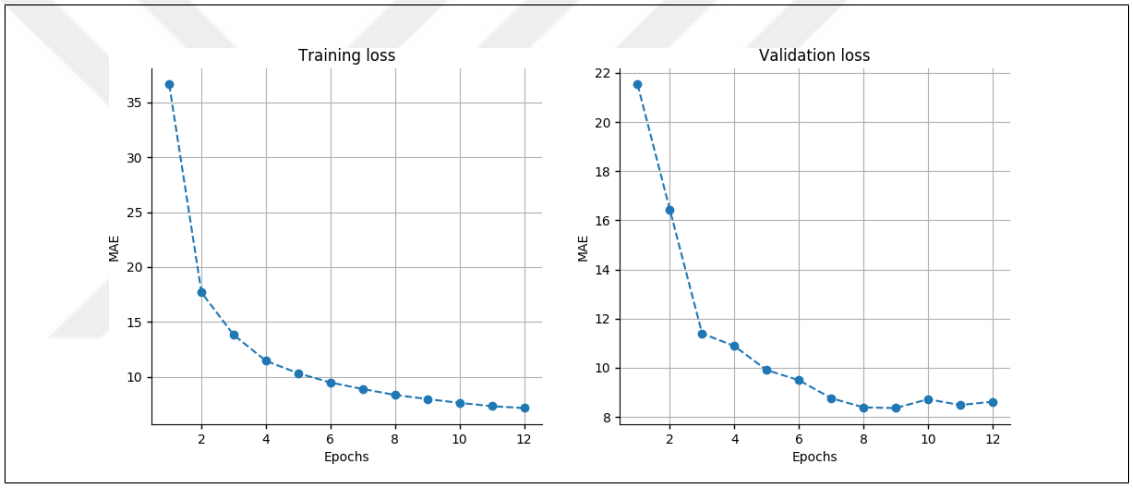


5.2.1 Our Model

The TurtleBot moved in the lab environment for approximately 60 minutes recording RGB, Depth and Odometry data. A total of 89032 RGB images with their corresponding 2D location and orientation were obtained.

Images captured by the kinetic camera had a resolution of (640,480) which was converted with a factor of 0.5 to (320,240). The system was trained with a batch size of 8 for 15 epochs on 90% of the RGB images with early stopping option. The other 10% were used for validation.

Figure 5.6: Training and validation loss for the lab environment model.



In figure 5.6 we can observe that the model activated early stopping at epoch 12 since the accuracy did not improve after epoch 9 which was saved. Table 5.3 shows the results of our model.

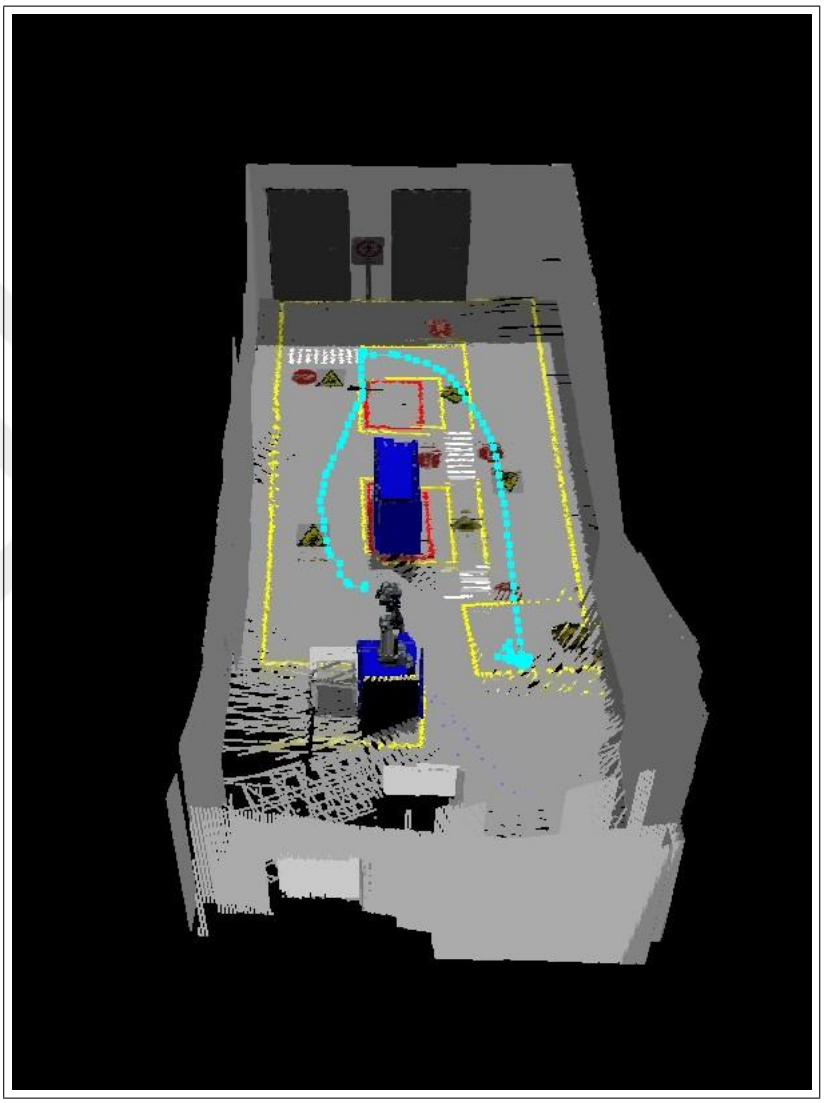
Table 5.3 Validation results for the lab environment.

Environment	Validation error	Standard deviation
Lab simulation	$0.17m$, 3.7°	$0.23m$, 10°

5.2.2 RTAB-Map

In this environment RTAB-Map mapped the environment as shown in figure 5.7, it achieved loop closure with no drift in mapping.

Figure 5.7: Lab environment in RTAB-Map.



However, in localization mode RTAB-Map suffered the lack of features, especially in the walls, which caused the robot to falsely localize the robot many times. The mean error and standard variation is shown in table 5.4.

Table 5.4 RTAB-Map results for the lab environment.

Environment	Mean Error	Standard deviation
Lab simulation	$2.2m$, 2.4°	$2.6m$, 3.6°

6. EXPERIMENTAL RESULTS

In this chapter, we present the experimental localization results. Firstly, we introduce the layout and arrangements of the test concluded in a real-world experiment. Secondly, we show our model's results, and lastly we display the performance of RTAB-Map.

6.1 HARDWARE

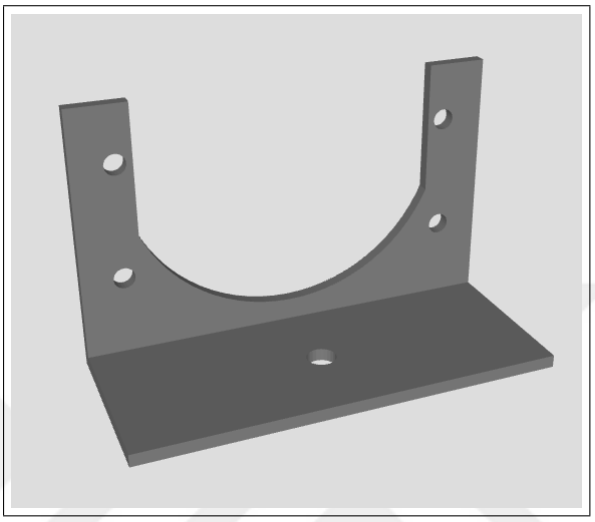
Testing was done on a KUKA youBot, it consists of an omnidirectional mobile platform on which a 5-axis robot arm with a gripper is installed. A model of KUKA youBot is shown in figure 6.1.

Figure 6.1: KUKA youBot.

Source: youbot-store.com

Zed camera was used to capture the images by linking it to the youBot's gripper with a 3D printed link shown in figure 6.2 and a ROS node was created to adjust the manipulator's joints to position it as shown in figure 6.3 in order to give the camera a good view of the environment.

Figure 6.2: 3D printed link.



A Jetson TX2 Development Kit is also connected to the base of the youBot running on JetPack3.3 connected to the computer of the youBot (Mini-ITX) in order to run the Zed camera, use platforms such as CUDA and take advantage of its processing power such as RAM memory and GPU that is way higher than the youBot's capabilities.

Figure 6.3: Jetson TX2 with KUKA youBot and zed camera.

In figure 6.3 the Jetson TX2 is using ROS kinetic which is also connected by an Ethernet cable to the youBot's ROS Kinetic via SSH connection that gives the ability for the Jetson to act as a ROS master.

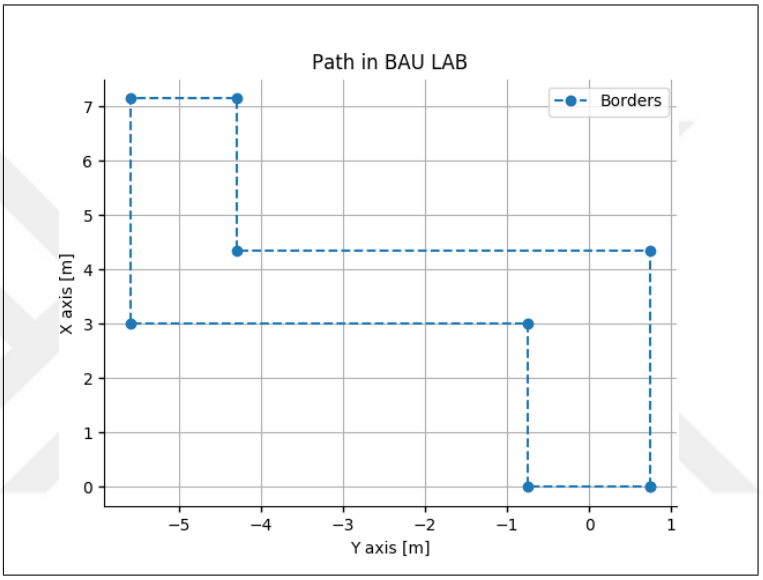
The time is synchronized between the YouBot's computer and the Jetson using an NTP server deployed in the Jetson and an NTP client deployed in the youBot.

6.2 OUR MODEL

Our Deep learning based model is trained for the experiment on TensorFlow1.9 using Keras libraries on an ASUS X550VX with a 4GB NVIDIA GEFORCE GTX 950M.

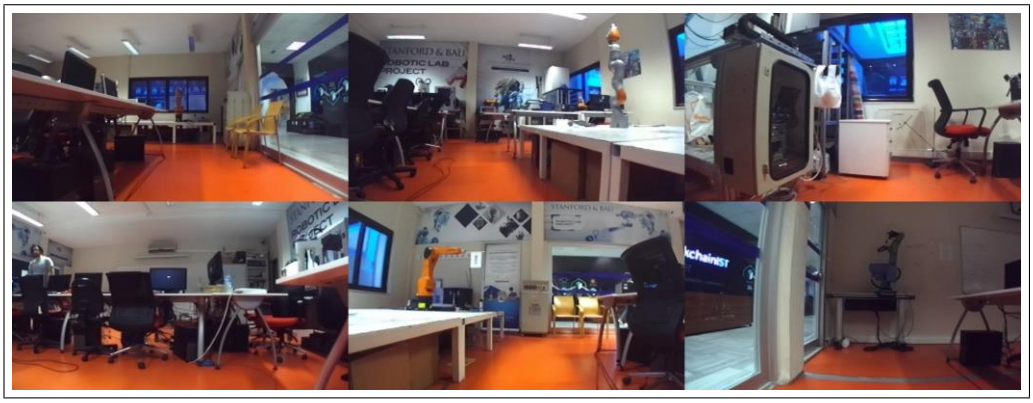
The testing was done in the BAU Robotics Lab; a path of nearly (1.5X11)m^2 was designed for the youBot to navigate in. Figure 6.4 describes the dimensions of the path.

Figure 6.4: Path in BAU lab.



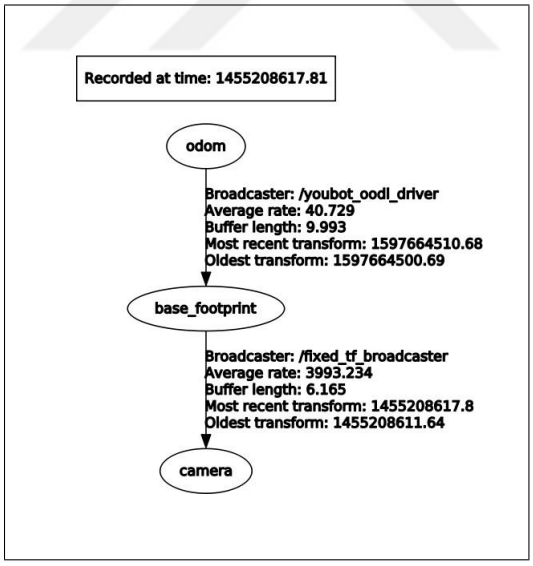
The YouBot moved in the environment for a total of 1 hour and a total of 51134 RGB images were recorded, each image with the corresponding encoders' odometry and the visual odometry provided by the Zed camera with their timestamps. Figure 6.5 presents samples of the collected RGB images.

Figure 6.5: Samples of the dataset.



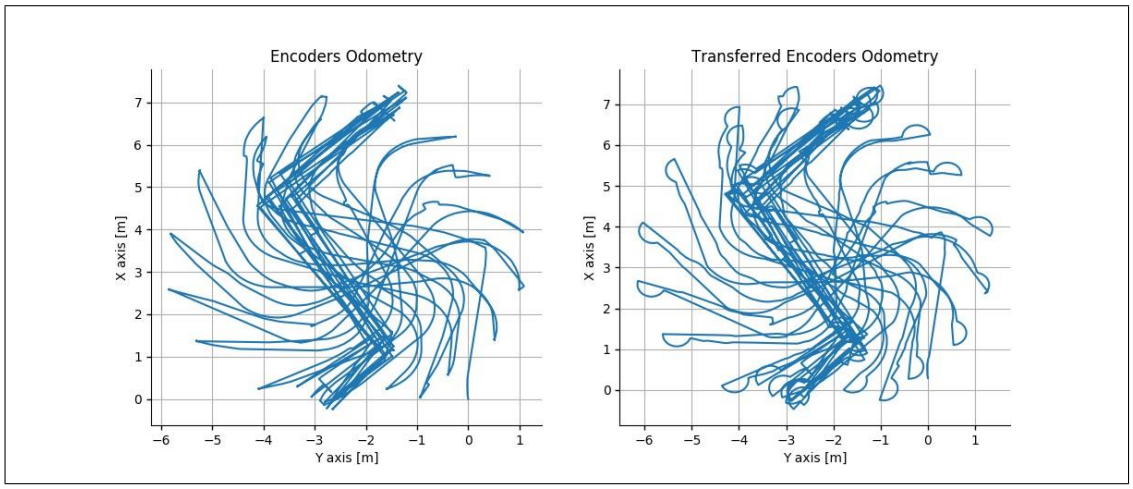
A link transformer shown in figure 6.6 was created and added to the youBot's tf broadcaster and a listener was created to transform the position and orientation of the encoders' odometry which is located in the center of the youBot's base to the camera's position which is located on the gripper.

Figure 6.6: ROS TF frame tree.



We analyze both encoders' odometry and Zed camera's odometry and compare them to the designed path to choose the ground truth to label the recorded images with.

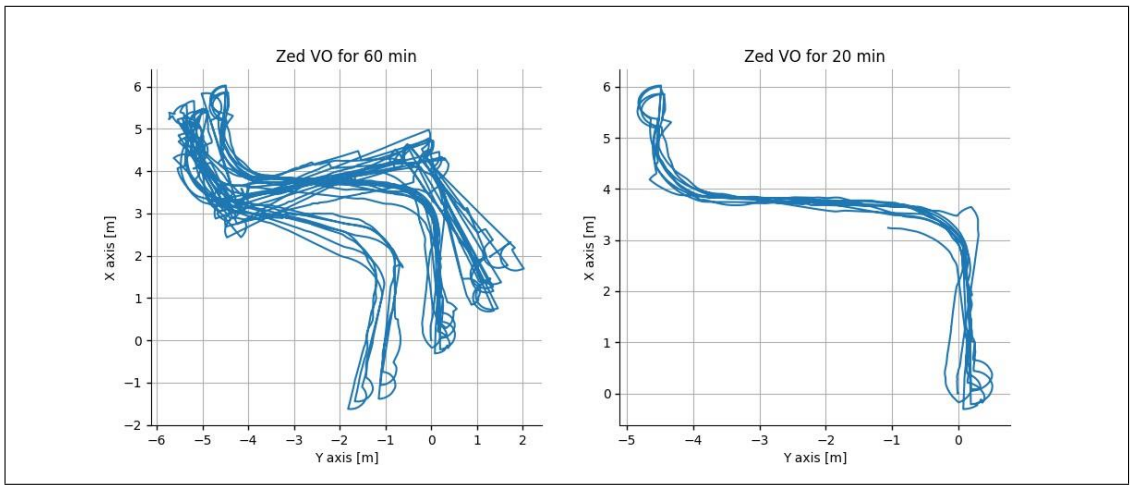
Figure 6.7: Encoders odometry data.



As observed in figure 6.7 above the encoders' odometry has a big error that accumulates over time rapidly and therefore is not eligible to consider as a ground truth.

As for the Visual odometry provided by the Zed camera, it starts to drift after nearly 20 minutes of recording data. As figure 6.8 shows the Zed odometry in the first 20 minutes is very accurate, unlike the last 40 minutes which has an error.

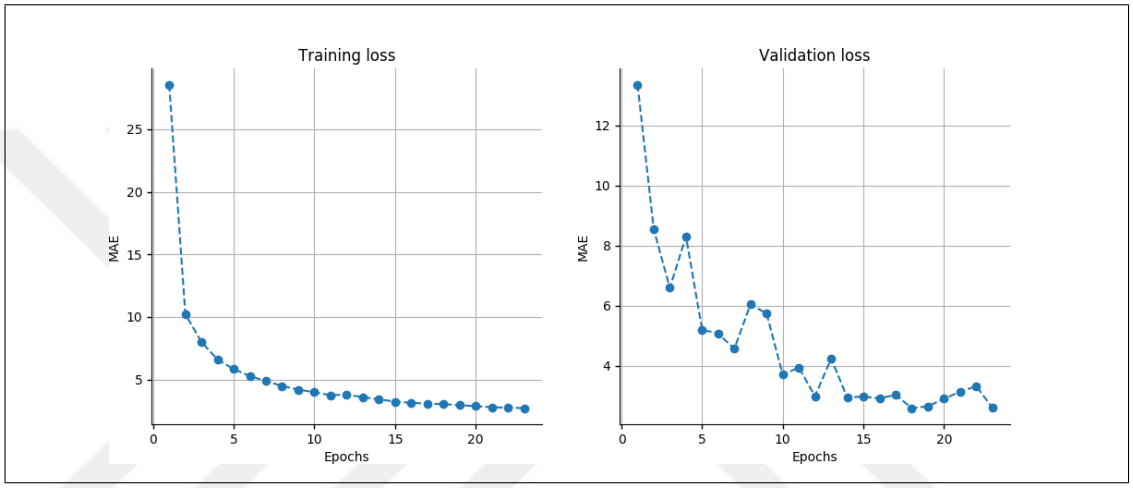
Figure 6.8: Zed camera visual odometry.



This leads us to discard the data recorded in the last 40 minutes. Therefore, we only use 34% of the data and a total of 17228 RGB images were considered in which 90% were trained with their labels, the other 10% are for validation.

The images captured by the Zed camera have a resolution of (672,376) in (W ,L), the images are scaled by a factor of 0.5 to (336,188) and a total of 25 epochs with an early stopping and checkpoint training was activated with a batch size of 7.

Figure 6.9: Training process.



The model was trained for roughly 5 hours and an early stopping was activated to save the best model at 18 epochs which had the best validation accuracy and to avoid over fitting. Figure 6.9 describes the training and validation process. The results are presented in table 6.1.

Table 6.1 Validation results for the experiment.

Environment	Validation error	Standard deviation
BAU Lab	$0.049m$, 1.4°	$0.036m$, 4.3°

6.3 RTAB-Map

Using rtabmap-ros package on the Zed camera, RTAB-Map failed to map the environment in the first attempt because it lost the odometry, but it successfully mapped the lab in the second attempt as shown in figure 6.10.

Figure 6.10: RTAB-Map of BAU LAB.

In localization the robot moved in the path for nearly 19m presented in figure 6.11 and both the zed odometry and RTAB-Map odometry were recorded.

Figure 6.11: Path of the robot.

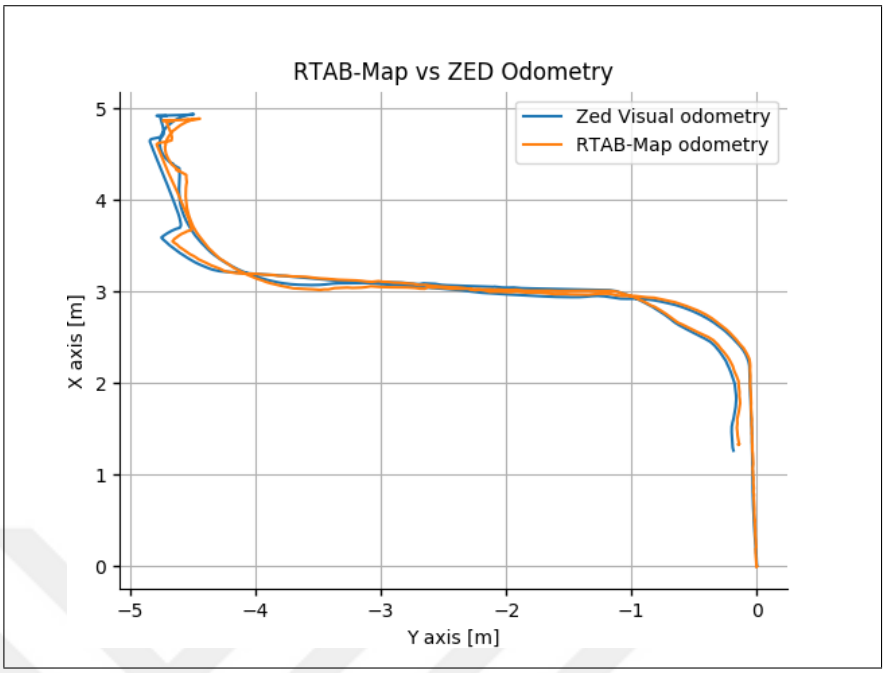


Table 6.2 below shows the results of RTAB-Map.

Table 6.2 RTAB-Map results for the experiment.

Environment	Mean Error	Standard deviation
BAU Lab	$0.057m, 0.96^{\circ}$	$0.028m, 0.38^{\circ}$

7. CONCLUSION

In this thesis, a vision-based indoor localization system is developed to predict the location of a robot based on the RGB inputted image. The system is deep learning based which consists of multiple layers of CNN and FFNN. It highly depends on a training process that consists of thousands of labeled images, a large environment requires a bigger dataset.

The system was developed in a simulation environment using Gazebo and turtlebot2 and tested in another simulation environment. For validation, it was also tested in a real experimental environment, on the hardware level, we successfully integrated KUKA youBot with Jetson TX2 and a zed camera which was our vision sensor.

Our method is able to create a model that understands or represents the surrounding environment successfully. And determines the location and orientation of the mobile robot based on the available vision sensor with insignificant error.

After comparing our model with RTAB-Map, we can establish that the system performs as the V-SLAM state-of-the-art method.

REFERENCES

Books

Ibrahim, M., Omar, S., & Mostafa, H., 2011. *Extended Kalman Filter Simultaneous Localization and Mapping (Graduation Project)*.

Parsons, S., 2010. Introduction to machine learning, second editon by ethem alpaydin. *Knowledge Engineering Review - KER* **25**.

Rosenblatt, F., 1962. *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms*.

Siciliano, B. & Khatib, O., 2016. Springer handbook of robotics. In *Springer Handbooks*. pp. 1153–1175.

Periodicals

Bettaieb, L. A., 2017. A deep learning approach to coarse robot localization.

Biswas, J. & Veloso, M. M., 2010. Wifi localization and navigation for autonomous indoor mobile robots. *2010 IEEE International Conference on Robotics and Automation* , pp. 4379–4384.

Dellaert, F., Fox, D., Burgard, W., & Thrun, S., 1999. Monte carlo localization for mobile robots. *Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No.99CH36288C)* **2**, pp. 1322–1328 vol.2.

DeTone, D., Malisiewicz, T., & Rabinovich, A., 2017. Toward geometric deep slam. *ArXiv* .

Gurel, C. S., 2018. Real-time 2d and 3d slam using rtab-map, gmapping, and cartographer packages.

Hamzeh, O. & Elnagar, A., 2015. Localization and navigation of autonomous indoor mobile robots. vol. 2. p. 228–233.

Kalman, R. et al., 1960. A new approach to linear filtering and prediction problems. *Journal of basic Engineering* **82**, pp. 35–45.

Kendall, A., Grimes, M. K., & Cipolla, R., 2015. Posenet: A convolutional network for real-time 6-dof camera relocalization. *2015 IEEE International Conference on Computer Vision (ICCV)* , pp. 2938–2946.

Kim, S.-B. & Lee, J.-M., 2007. Precise indoor localization system for a mobile robot using auto calibration algorithm. *The Journal of Korea Robotics Society* **2(1)**, pp. 40–47.

Labbé, M., 2015. Simultaneous localization and mapping (slam) with rtab-map. *Universite´ de Sherbrooke* .

Lim, J., Lee, S. J., Tewolde, G. S., & Kwon, J., 2015. Indoor localization and navigation for a mobile robot equipped with rotating ultrasonic sensors using a smartphone as the robot's brain. *International Journal of Handheld Computing Research* **7**, pp. 621–625.

Malagon-Soldara, S. M., Toledano-Ayala, M., Soto-Zarazúa, G. M., Carrillo-Serrano, R. V., & Rivas-Araiza, E. A., 2015. Mobile robot localization: A review of probabilistic map-based techniques. In *ICRA 2015*, vol. 4. pp. 73–81.

Mur-Artal, R. & Tardos, J. D., 2017. Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras. *IEEE Transactions on Robotics* **33(5)**, p. 1255–1262.

Nilwong, S., Hossain, D., Kaneko, S.-i., & Capi, G., 2019. Deep learning-based landmark detection for mobile robot outdoor localization. *Machines* **7**, pp. 2–25.

Oh, J. H., Kim, D., & Lee, B. H., 2014. An indoor localization system for mobile robots using an active infrared positioning sensor. *Journal of Industrial and Intelligent Information* **2(1)**, pp. 35–38.

Raghavan, A. N., Ananthapadmanaban, H., Sivamurugan, M. S., & Ravindran, B., 2010. Accurate mobile robot localization in indoor environments using bluetooth. *2010 IEEE International Conference on Robotics and Automation* , pp. 4391–4396.

Ren, S., He, K., Girshick, R. B., & Sun, J., 2015. Faster r-cnn: Towards real-time object detection with region proposal networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence* **39**, pp. 1137–1149.

Rumelhart, D., Hinton, G. E., & Williams, R. J., 1986. Learning representations by back-propagating errors. *Nature* **323**, pp. 533–536.

Strasdat, H., Montiel, J. M. M., & Davison, A. J., 2012. Visual slam: Why filter? *Image Vis. Comput.* **30**, pp. 65–77.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., & Rabinovich, A., 2015. Going deeper with convolutions. *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* , pp. 1–9.

Tananaev, V., Tananaev, D., & Kutkina, O., 2016. Localization with deep learning .

Tateno, K., Tombari, F., Laina, I., & Navab, N., 2017. Cnn-slam: Real-time dense monocular slam with learned depth prediction. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* , pp. 6565–6574.

Zhang, H.-L., Gao, S., Xiao, Meng, L., & Li, L., 2020. Cost-effective wearable indoor localization and motion analysis via the integration of uwb and imu. *Sensors* **20**, p. 344.