THE UNIVERSITY OF JORDAN

School of Engineering

Department of Mechatronics Engineering

Bachelor of Science in Mechatronics Engineering

Senior Design Graduation Project Report

Design of novel positioning system for internet of things (IoT)

Report by

Ramez Qasim (0167595)

Jamal Sa'd (0162986)

Supervisor

Dr. Hussam Khasawneh

Date

26/05/2021

DECLARATION STATEMENT

We, the undersigned students, confirm that the work submitted in this project report is entirely our own and has not been copied from any other source. Any material that has been used from other sources has been properly cited and acknowledged in the report.

We are fully aware that any copying or improper citation of references/sources used in this report will be considered plagiarism, which is a clear violation of the Code of Ethics of University of Jordan.

In addition, we have read and understood the legal consequences of committing any violation of the University's Code of Ethics.

	Student Name	Student ID	Signature	Date
1	Ramez Shawqi Qasim	0167595		26/05/2021
2	Jamal Saad	0162986		26/05/2021

ABSTRACT

Outdoor positioning has made great leaps in the past decade or so through the Global Positioning System (GPS) and the introduction of mobile internet. With the prevalence of Internet of Things (IoT) devices, the need for more capable positioning systems that work in complex indoor environments is becoming a necessity. These systems must withstand real world challenges such as attenuation, multipath effect, noise, etc., and predict position accurately.

This three-part work was concerned with designing a novel indoor positioning system using the minimum equipment installation cost and equipment cost possible while achieving good accuracies. This was done through fusing Received Signal Strength Indicator (RSSI) measurements from reference Wireless Fidelity (Wi-Fi) beacons with measurements from an Inertial Measurement Unit (IMU) in a robot. It started by simulating IMU and Wi-Fi RSSI measurements for a robot moving inside a 32 \times 32 \times 7 meters warehouse floor. Then, a Deep Learning (DL) model that uses Long Short Term Memory (LSTM) cells was built to find the robot's position within said environment. Finally, the placement of those reference beacons was optimized such that the accuracy of the predicted position was maximized.

Results showed a Root Mean Square Error (RMSE) in the predicted position of 0.4821 meters and a Mean Absolute Error (MAE) of 0.4786 meters using the developed LSTM model. The optimized placement of the beacons resulted in a 27.71% decrease in the RMSE and a 47.75% decrease in the MAE of the aforementioned DL model.

ACKNOWLEDGEMENTS

We are thankful for having supportive families, professors and friends, this work would not have been done without them. We are particularly thankful to our supervisor professor Hussam Khasawneh.

Finally, we would like to thank the tree with coordinates [32.01317483, 35.873916114] under which we contemplated life and future.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

GLOSSARY

ABBREVIATION	DESCRIPTION
ACO	Ant Colony Optimization
AdaGrad	Adaptive Gradient Algorithm
Adam	Adaptive Moment Estimation
AI	Artificial Intelligence
ANN	Artificial Neural Network
AP	Access Point
BD	Big Data
CNN	Convolutional Neural Network
CPU	Central Processing Unit
DL	Deep Learning
DLOA	Deep Learning Optimization Algorithm
DOF	Degrees of Freedom
EMF	Electromagnetic Field
FSPL	Free-space Path Loss
GA	Gradient Ascent
GD	Gradient Descent
GLONASS	Russia's Global Navigation Satellite System
GNSS	Global Navigation Satellite System
GPS	Global Positioning System
GPU	Graphics Processing Unit

IARC	International Agency on Research on Cancer
IMU	Inertial Measurement Unit
LORAN	Long Range Navigation
LOS	Line-of-Sight
LSTM	Long Short-term Memory
MAE	Mean Absolute Error
MEMS	Micro-electromechanical Systems
ML	Machine Learning
MST	Minimum Spanning Tree
NN	Neural Network
PDF	Probability Density Function
PMS	Probability Mass Function
ReLU	Rectified Linear Unit
RF	Radio Frequency
RFID	Radio Frequency Identification
RMSE	Root Mean Square Error
RMSprop	Root Mean Square Propagation
RNN	Recurrent Neural Networks
RS	Random Search
RSSI	Received Signal Strength Indicator
RTT	Round Trip Time
SA	Simulated Annealing
SGD	Stochastic Gradient Descent
SGDM	Stochastic Gradient Descent with Momentum

ToF	Time of Flight
TSP	Travelling Salesman Problem
UBI	Universal Basic Income
Wi-Fi	Wireless Fidelity
λ^*	Wavelength
D_r	Receiving Antenna Directivity
D_t	Transmitting Antenna Directivity
$\widetilde{P}$	Received Signal Power in the New Log Scale
P_r	Received Power
P_t	Transmitted Power
R_i	Receiver Image
R_x	Receiver
T_x	Transmitter
U^{xy}	Achieved Accuracy at (x, y)
$\widetilde{d}$	Distance in the New Log Scale
f_{xy}	PDF of the Robot's Location
g_{xy}	PMS of the Robot's Location
l_d	Lookahead Distance
$\widetilde{U}$	Smoothened Expected Accuracy of Placement of Beacons
θ^*	Heading Angle
α^*	Beacon x-coordinate Vector
β^*	Beacon y-coordinate Vector

∇	Gradient Operator
λ	Regularization Parameter
F	Feasible Set
N	Number of Beacons
P	Received Signal Power
c	Speed of Light
d	Distance
s	Standard Deviation
H	Hessian Matrix
W	Weight Matrix
g	Gradient Vector
h	Prediction Vector
$\mathcal{U}$	Expected Accuracy of Placement of Beacons
γ	Momentum Parameter
δ	Curvature
η	Number of Examples
θ	Roll Angle
μ	Mean
ρ	Reflection Plane
σ	Dropout Probability
ψ	Yaw Angle
ϕ	Pitch Angle

Chapter 1 INTRODUCTION

1.1 *Project Introduction*

The technologies of positioning, localization, and navigation have been extensively studied and profitably widely employed in several applications. More specifically, indoor positioning technology has become steadily more crucial as the new technology of chip-level Micro-electromechanical Systems (MEMS) evolves. Furthermore, the considerable development in positioning Big Data (BD) and Machine Learning (ML) technologies and the social potential and the growth of broad public interest in it has made indoor positioning more critical.

It is predicted that the worldwide indoor positioning, localization, and navigation market cap will hit more than $28 billion by 2024, with a steep growth annual rate of more than 38% [1]. Consumer giants such as Tesla, Apple, Nvidia, and Google have raised their recognition of indoor positioning technology. This recognition by major players in the high-tech field is mainly because of the highly emerging autonomous driving vehicle applications that need indoor positioning capabilities. The main three issues to be addressed by both traditional and unmanned vehicles are: the problem of positioning, localization, and navigation, the problem of environmental perception, and the issue of making decisions. An unmanned vehicle needs to position itself accurately to some level within the surrounding environment, which is the indoor environment in this context, and make decisions to take potential actions. Hence, it is necessary that the indoor positioning problem be solved so that location services, as well as completely autonomous driving services, can be delivered. Some degree of accuracy has to be achieved for the indoor positioning systems so that they are practical enough. For example, in some construction and industry applications of indoor positioning systems, a millimeter-level accuracy is required [2].

Regarding the sensors used in indoor positioning systems, specialized sensors are typically installed in the environment, such as environmental monitoring and awareness sensors. However, it would be cheaper and more feasible and widely applicable if an already-existing infrastructure is used. A ubiquitous existing-infrastructure that is available in many indoor environments is the Wireless Fidelity (Wi-Fi) networks that are used to deliver internet connection within an indoor area. Research has been conducted to employ this infrastructure for the purpose of indoor positioning [3]. The minimal cost of installing this infrastructure of

Wi-Fi sensors for the indoor environment is in the order of $\$0.1/m^2$. Wi-Fi, alongside Bluetooth, is the number-one used indoor positioning technology for consumer electronics. The typically used criterion to determine the position in this indoor positioning application is Received Signal Strength Indicator (RSSI) [4], where the usual positioning accuracy is in the order of 1 meter. Furthermore, more high-accuracy measurements have been extracted. Some of these new measurements can be used for indoor positioning applications that require an order of 0.1 meter accuracy, or even accuracy at the order of 0.01 meter [5], [6].

In order to be able to meet the future requirements of both Internet of Things (IoT) and accurate localization, some additional features can be added to the latest Wi-Fi technologies. Some of these recently added features include long-range low-power consumption in Wi-Fi HaLow, which has been released in order to improve the range of the signal, while Wi-Fi Round Trip Time (RTT) has been released in order to improve the precision of positioning when Wi-Fi is applied in positioning systems.

In addition to that, Inertial Measurement Units (IMUs) can extract the states of the motion by employing the measurements of both the linear and angular-rate specific force from accelerometers and gyroscopes [7]. These IMUs are traditionally utilized in many applications that include military, mobile surveying, and aerospace. A couple of decades earlier, cheap IMUs that are based on the MEMS technology were used in the positioning of land vehicles [8]. Furthermore, the use of MEMS-based IMUs has become a standard feature for smartphones since the release of iPhone 4. This has driven the emergence of new applications such as pedestrian indoor positioning. Various degrees of inertial sensors have different costs and performances. Hence, it is important to choose a proper IMU type corresponding to the requirements of the application.

The IMU is known to be able to provide autonomous indoor positioning solutions. This means that it does not need to receive any external signal or to interact with outside environments. This powerful self-contained quality of this type of sensor makes it the candidate for ensuring indoor positioning reliability and continuity while the performance of other types of sensors deteriorate by environmental causes. However, it is important to consider the main hindrance of these IMU sensors, their main source of error in positioning systems, which is the significant accumulation of IMU sensor errors, the thing that causes the measurements to drift widely.

1.2 *Problem Definition*

With the increasing demand for indoor navigation solutions, and the vitality of its impact on future technologies, there are a lot of gaps and pain points that can be alleviated and improved. Namely, improving positioning accuracy and minimizing the number of beacons while maintaining maximum positioning accuracy.

The ability to properly detect the position of the operating robot is the most crucial asset to navigate and automate its functions properly. However, IMU sensor readings suffer from increasing inaccuracy the more the distance travelled, as the error gradually accumulates. Therefore, the use of other range technologies such as Bluetooth beacons and WIFI beacons is needed.

As mentioned in 1.1, technologies such as Bluetooth and WIFI beacons have their own drawbacks as well, including signal blocking and signal distortion. Therefore, neither range technologies nor IMU sensors individually suffice to achieve a satisfactory positioning accuracy.

Another drawback of WIFI and Bluetooth beacons is the overhead costs of deploying the appropriate number of beacons and distributing them over the working environment, too many beacons increase overhead and maintenance costs, and pose a threat of excess electromagnetic field exposure.

1.3 *Aims and Objectives.*

1.3.1 Aims

- Optimize the number of WIFI / Bluetooth beacons deployed within a working environment.
- Optimize the distribution of said beacons across the said environment to achieve best positioning accuracy.

1.3.2 Objectives

- Simulate a factory environment, where WIFI beacons are deployed, and a robot with IMU sensors moves within the environment.
- Generate sensors data and WIFI signals data from the simulated environment.

- Design a deep learning model that can accurately infer the position of the robot given sensors' readings.
- Develop an optimization algorithm to optimize the location of beacons.
- Demonstrate an increased positioning accuracy after optimizing the location of beacons.

1.4 Project Plan

Graduation Project (1) Tasks Description

Task No.	Task Description	Semester Weeks																Students Involved			
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	1	2	3	4
1	Brainstorming																				
2	Literature Review																				
3	Reviewing available sensors and technologies																				
4	Reviewing of ML techniques																				
5	Familiarization with Kalman Filters and sensor fusion technologies																				
6	Specification of simulation and development platforms based on our needs																				
7	Review and trial of existing simulation environments																				
8	Studying optimization techniques																				

Graduation Project (2) Tasks Description

Task No.	Task Description	Semester Weeks																Students Involved			
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	1	2	3	4
1	Develop Simulation environment (1)																				
2	Develop Simulation Environment (2)																				
3	Develop Simulation Environment (3)																				
4	Data Analysis and Visualization																				
5	Design ML models to determine the position																				
6	Optimization Algorithm																				
7	Documentation and presentation																				

1.5 *Report Organization*

After introducing the general motives of this project and the desired objectives to be achieved, chapter two discusses the design options to achieve those objectives and the constraints that guide our choices. Chapter three divides the project into three complementary fields and discusses the background of each along with the already existing solutions, then the procedures and methodologies implemented in the design and analysis are explained in chapter four. Chapter five presents the results and further discusses them, and chapter 6 outlays project conclusions and recommended future work.

Chapter 2 BACKGROUND AND LITERATURE REVIEW

2.1 *Positioning Fundamentals*

Throughout history, different methods have been used for positioning. Dead reckoning, the method of estimating the position of a ship depending on its previously determined point of departure, has helped sailors in completing thousands of trips across oceans. Using maps, compasses, and knotted ropes, ships navigated east and west and generated wealth through trade and colonization. However, dead reckoning is prone to cumulative errors. These errors have led to unfortunate accidents throughout history.

With the rise of radio applications in the early 20th century, radio wave signals have been utilized to create navigation solutions. Long Range Navigation (LORAN) was introduced in the 1940s and allowed ships to determine their position through triangulation from LORAN shore-based stations. Later in the 1970s, the satellite-based system, Global Positioning System (GPS), was introduced in the United States and allowed ships, vehicles, aircrafts and humans to determine their position anywhere on earth. After GPS, Russia introduced its Global Navigation Satellite System (GLONASS), Europe introduced GALILEO, and China introduced BeiDou [9]. All satellite-based navigation systems are considered Global Navigation Satellite Systems (GNSSs). A figure depicting a GNSS is shown in Figure 1.

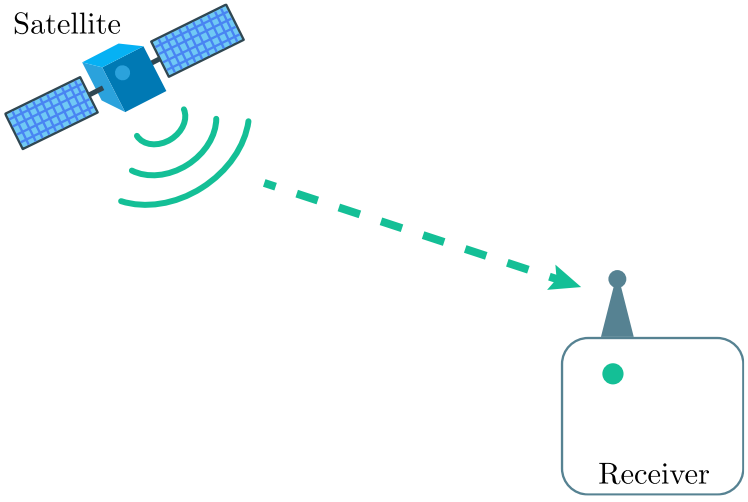

Figure 1: A GNSS [10].

In robotic applications, the pose describes the combination of position and orientation of the moving body. Degrees of Freedom (DOF) of a mechanical system is the number of independent variables that describe its state. DOF is important in the analysis of systems of bodies in aerospace engineering, drones, and robotics. The pose of a body could be described by variables equal to the DOF. The navigation problem could either be two dimensional (3-DOF) or three dimensional (6-DOF). For two dimensional problems, the pose problem contains three unknowns to be solved for, expressed as

$$\text{Pose} = \begin{bmatrix} x \\ y \\ \theta^* \end{bmatrix}, \tag{1}$$

where x is the position in the x-axis, y is the position in the y-axis, and θ is the heading angle. For three dimensional problems, the pose problem contains six unknowns to be solved for, expressed as

$$\text{Pose} = \begin{bmatrix} x \\ y \\ z \\ \phi \\ \psi \\ \theta \end{bmatrix}, \tag{2}$$

where x is the position in the x-axis, y is the position in the y-axis, z is the position in the z-axis, ϕ is the pitch angle, ψ is the yaw angle, and θ is the roll angle. The six unknowns are depicted in Figure 2.

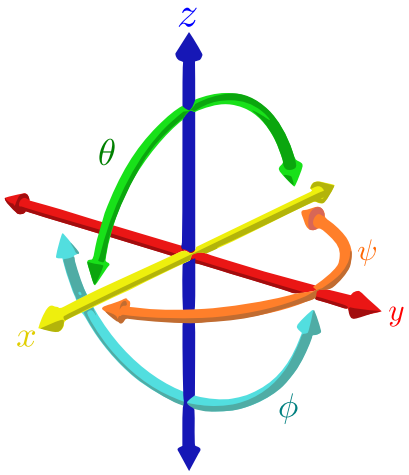

Figure 2: The 6-DOF in three dimensional problems [11].

Trilateration, which is shown in Figure 3, is the process of computing the position of a target using distances from stations of known locations used in GNSSs. Because satellite signals are blocked or distorted in indoor areas, different signal sources are needed for trilateration in those areas. The need for indoor positioning systems is growing dramatically as more robotic systems are functioning indoors. For example, robots are used in warehouses to pick up and move products efficiently. Signal sources such as Wi-Fi beacons, Bluetooth beacons, and Radio Frequency Identification (RFID) tags in conjunction with positioning methods could be utilized to provide a localization platform for indoor environments.

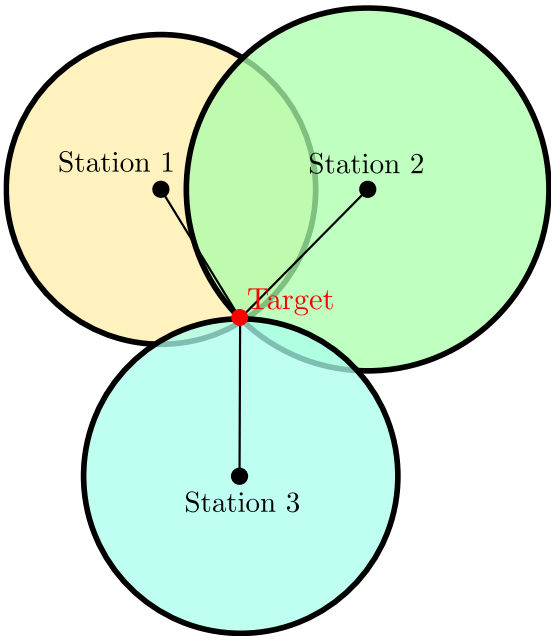

Figure 3: Depiction of trilateration [12].

Wi-Fi beacons are widely used in indoor localization, asset tracking, and robot navigation. Wi-Fi positioning has an advantage over other wireless methods, as Wi-Fi beacons are already used in indoor areas for communication and do not require a significant change in infrastructure. Received Signal Strength Indication (RSSI) and Time of Flight (ToF) are used to determine the distance between the target and fixed Wi-Fi beacons. With the Wi-Fi beacons position already known, trilateration is used to compute the position. An RSSI map of a Wi-Fi beacon in an indoor environment is shown in Figure 4.

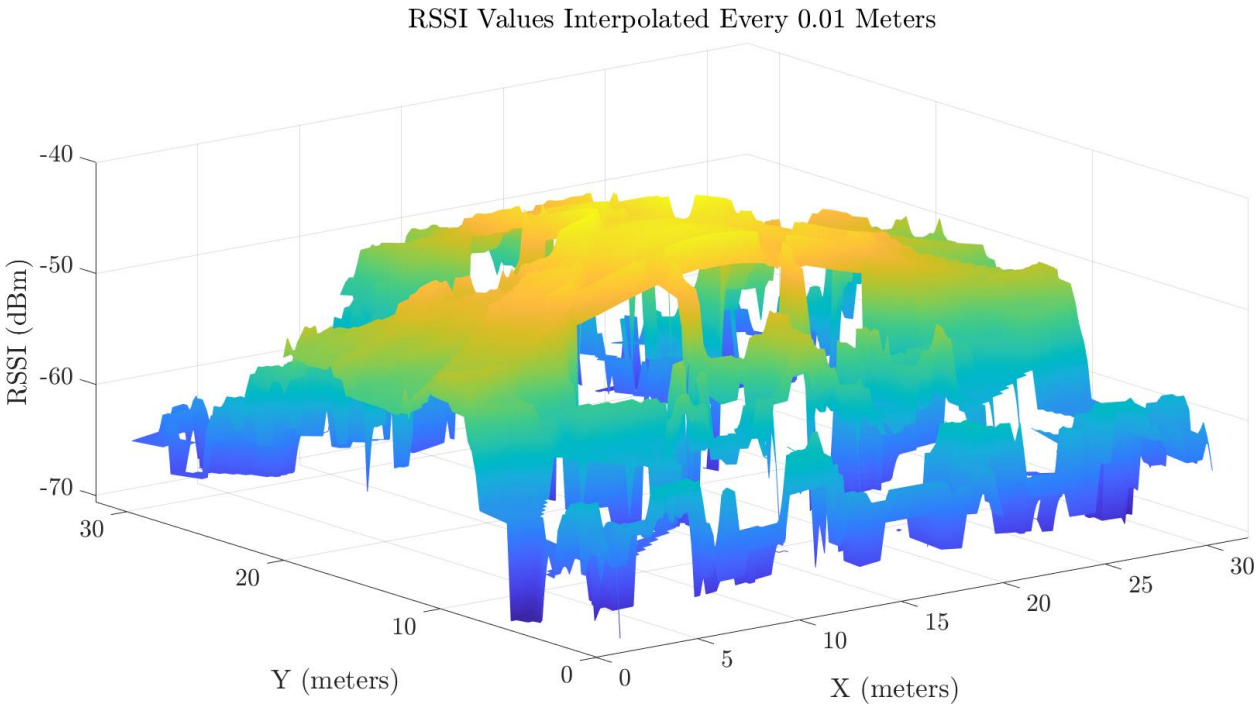

Figure 4: RSSI map of a beacon in an indoor environment.

The ToF positioning approach computes the range between the target and Wi-Fi beacons using timestamps of a sent message and its acknowledgement message. Assuming the Radio Frequency (RF) signals travel at the speed of light, the distance between the target and Wi-Fi beacons could be computed by multiplying ToF by the speed of light c. An example of the ToF approach in laser range-finding is shown in Figure 5. While this approach could provide higher accuracy than when the RSSI is used, it requires wo-way communication between the target and Wi-Fi beacons.

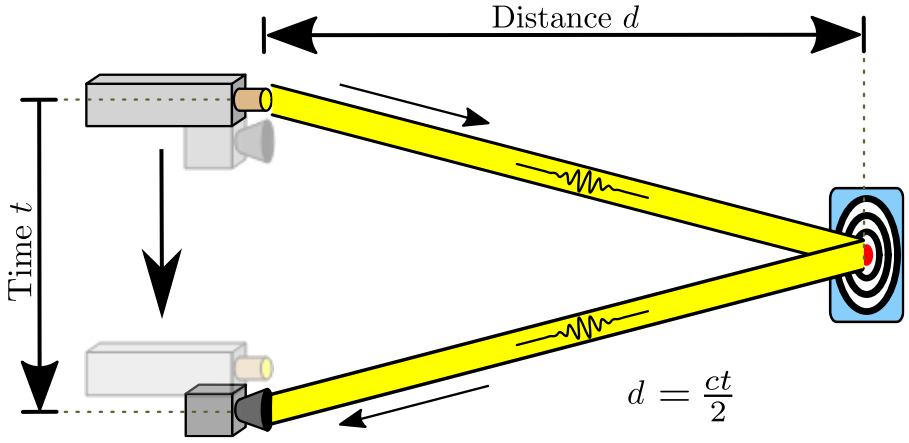

Figure 5: The ToF approach applied to laser range-finding [13].

FSPL is defined as "the loss between two isotropic radiators in free space, expressed as a power ratio". The power loss is directly proportional to the square of the distance between the transmitter and the receiver and inversely proportional to the square of the wavelength of radio waves [14]. Figure 6 shows how the same amount of power spreads over an area that is proportional to the square of the distance.

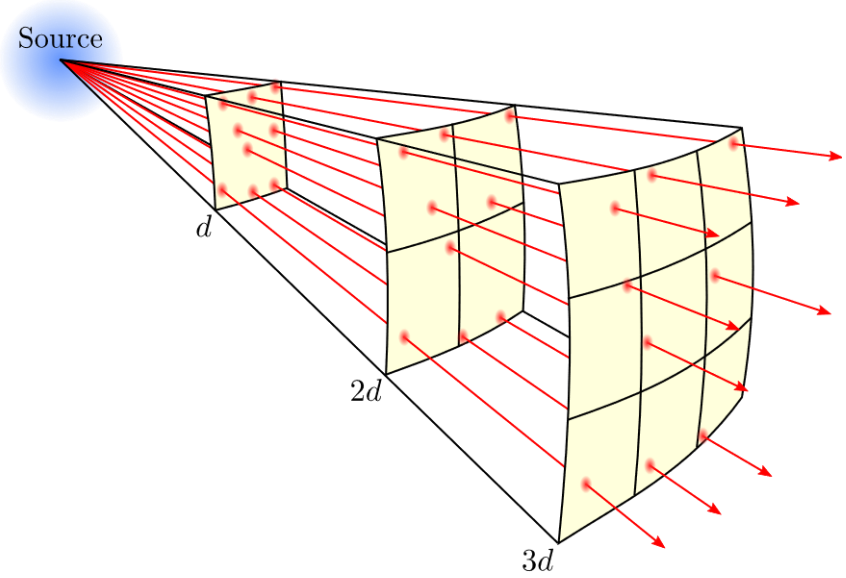

Figure 6: The inverse square law's relationship to FSPL [15].

The FSPL is defined mathematically as

$$\frac{P_r}{P_t} = D_t D_r \left(\frac{\lambda^*}{4\pi d}\right)^2 \quad (3)$$

where P_r is the received power, P_t is the transmitted power, D_t is the transmitting antenna directivity, D_r is the receiving antenna directivity, λ^* is the wavelength of the radio wave, and d is the distance between the transmitting and receiving antennas.

The FSPL model can be fairly accurate in modelling RSSI information when a direct Line-of-Sight (LOS) is found, but this is not always the case. Signals could be reflected, absorbed, refracted, diffracted, or scattered by objects in the environment. A better model to use is the multipath propagation model, which takes into account the multiple paths that the transmitted radio signals take due to reflections [16]. A depiction of the multipath propagation is shown in Figure 7.

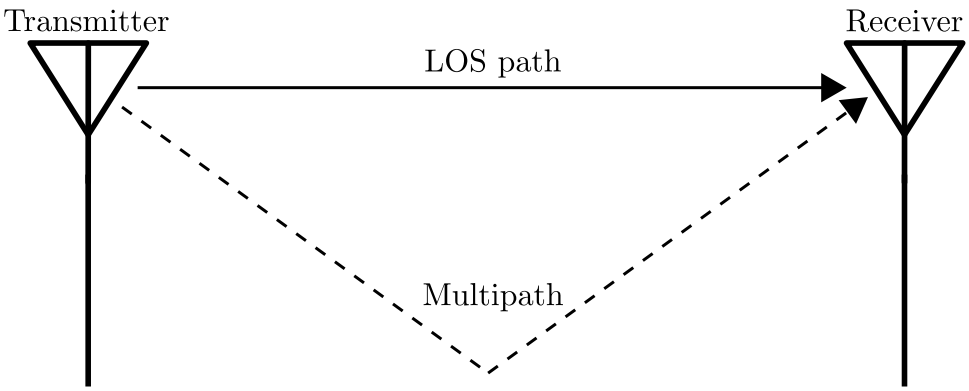

Figure 7: The multipath effect.

According to [17], experimental results have shown that the presence of one person between Wi-Fi Access Point (AP) and a mobile device reduced the RSSI by 5 dBm, which translates into a positioning error of more than 2 meters.

Ray tracing models reflected, scattered, and diffracted signals when the geometry and dielectric properties of the environment are known. It works by simulating the propagation of a discrete number of narrow beams, known as rays, through the environment. The complexity of computation increases with an increase in the number of simulated rays. More propagated rays result in a closer approximation of the continuous nature of the propagation of waves [18]. More on ray tracing in 4.1.

An (IMU) is a combination of embedded electronic sensors that measure force, angular rate, and orientation of bodies based on a combination of accelerometers, gyroscopes, and magnetometers. The components of IMU, such as the gyroscope, could be found in mechanical forms. However, IMUs in small-sized robotics and IoT applications are based on the technology of MEMS. An example of this is depicted in Figure 8.

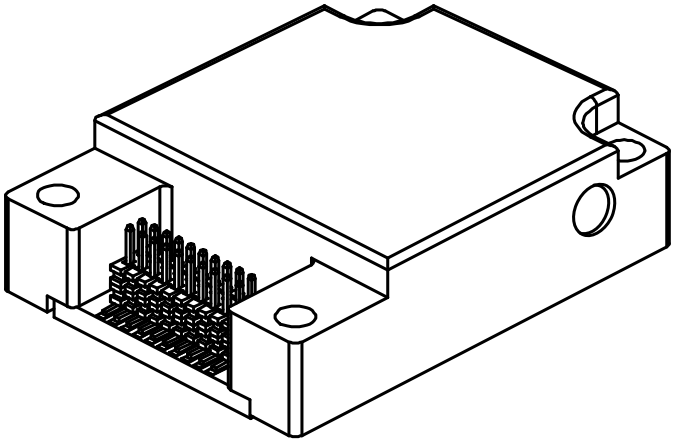

Figure 8: The ADIS16448 inertial sensor from Analog Devices [19].

Accelerometers are sensors that measure acceleration. Three-axis orthogonal accelerometers measure acceleration in three dimensions. To compute the location of the body, the signals from the accelerometers are double integrated. It is important to correct for gravity by subtracting acceleration due to it from the vertical axis signals prior to integrating. Table 1 shows the main error sources for the accelerometer and their effects on the result of integration.

Table 1: A summary of accelerometer error sources [2].

Error Type	Description	Result of Integration
Bias	Constant bias in output signal.	Quadratically growing error.
Bias Instability	Bias fluctuations.	A third-order random walk.
White Noise	White noise with standard deviation.	A second-order random walk.
Calibration	Errors in scale factors, alignments, and accelerometer linearities.	Drift proportional to the squared rate and duration of acceleration.

Gyroscopes on the other hand measure angular velocity. Three-axis orthogonal gyroscopes measure angular velocities in three dimensions. Orientation angles could be computed by integrating once. Table 2 shows the main error sources for the gyroscope and their effects on the result of integration.

Table 2: A summary of gyroscope error sources [2].

Error Type	Description	Result of Integration
Bias	Constant bias in output signal.	Steadily growing error.
Bias Instability	Bias fluctuations.	A second-order random walk.
White Noise	White noise with standard deviation.	An angle random walk.
Calibration	Errors in scale factors, alignments, and gyro linearities.	Drift proportional to the rate and duration of motion.

Magnetometers measure the magnetic field strength in a given direction. IMUs contain three-axis orthogonal magnetometers which are combined with the three-axis accelerometers and gyroscopes. By measuring the strength and direction of the local magnetic field, magnetometers compute the north direction. Magnetometers are not as accurate as accelerometers or gyroscopes. However, fusion of magnetometers could make the system more independent.

The advantage in using IMUs is that they are internal sensors that provide data on position and movement without relying on external infrastructure as is the case in GNSS, Wi-Fi, and Bluetooth based systems. GNSS signals could be heavily attenuated due to multipath effect or blocked completely in urban canyons. Wi-Fi and Bluetooth based positioning systems are even more vulnerable to this, due to the fact that their signals mostly propagate indoors where maintaining a direct LOS is extremely hard. On the other hand, IMUs are prone to drift and accumulative error due to the fact that the position is obtained through the double integration of the target's acceleration. Small errors could accumulate quickly and thus provide vey inaccurate estimates of position if used alone. Therefore, in most cases, systems fuse IMU measurements with measurements from more accurate positioning sensors. Sensor fusion enhances the accuracy of the positioning system and grants it robustness, allowing it to function even when one of the system components fail.

For long time, Kalman Filters have been the main method used in computing estimates for positioning solutions. Kalman Filters are based on weighting the accuracy of sensors

measurements against predicted estimates on every epoch. For different types of problems, variants of Kalman Filter have been developed, such as the Extended Kalman Filter (EKF) and Unscented Kalman Filter (UKF). With the rise of Artificial Intelligence (AI) and Machine Learning (ML) approaches, designers started utilizing them to build novel positioning solutions. Deep Learning, an important branch of ML, is considered as one of the very efficient approaches to develop novel positioning systems and enhance existing systems.

2.2 *Deep Learning Fundamentals*

Deep Learning (DL) is a subset of ML, which in turn is a subset of Artificial Intelligence (AI). AI is an umbrella term for what enables machines to mimic human behavior, while ML is the study of algorithms that enable machines to learn from experience through the use of data [20]. DL is a ML method that tries to learn through structures that are similar to those in the human brain called Neural Networks (NNs), shown in Figure 9. A basic NN is made of three types of layers. Namely, an input layer, one or more hidden layers, and an output layer. Each node (artificial neuron) connects the output of one layer to the input of another. Each node has a weight and an output threshold associated with it. If that threshold is exceeded, that node is activated making it pass data to the next node. If not, no data is passed along.

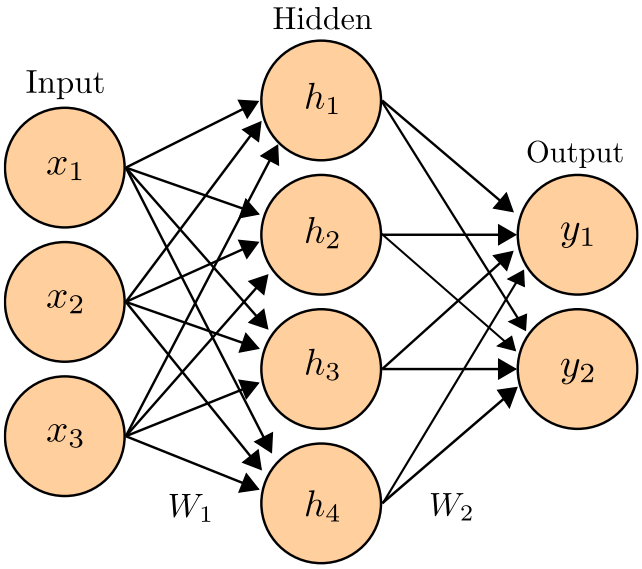

Figure 9: The basic structure of a NN.

The performance of ML algorithms is heavily dependent on the representation of the input data. Pieces of information in this representation are called features and they are what is used to find patterns in the data. This makes these algorithms "flat", meaning they cannot be fed raw data as an input. The process of creating these representations is called feature extraction, which is considered a very complex task due to the many choices of representations that are available. A deep understanding of the domain of the problem is therefore required to get acceptable results. For example, if a model that tries to detect cats from images were to be constructed, it would be very difficult to realize what makes a cat a cat in terms of pixel values. DL solves this problem by creating representations that are in terms of other, simpler representations, as shown in Figure 10. Visible layers (input and output layers) contain variables that are observable, while hidden layers contain variables we cannot observe that are constructed implicitly inside the DL model [21]. Due to this, NNs are able to do tasks that other ML algorithms can do, but the opposite is not true.

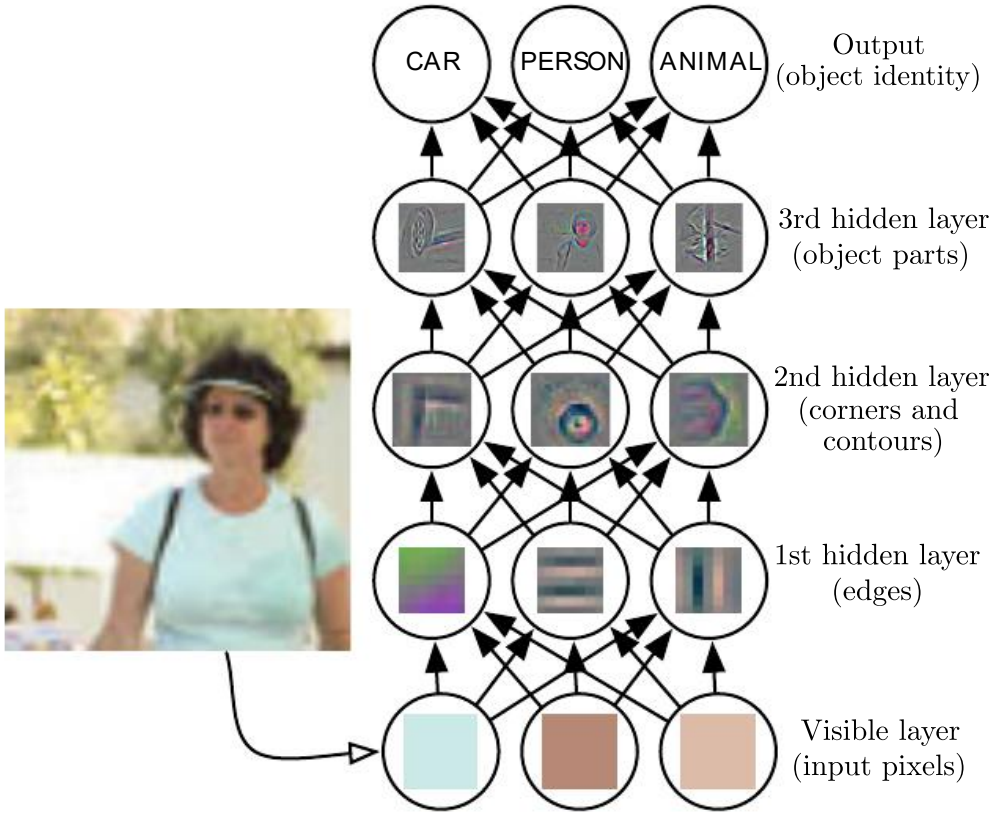

Figure 10: The feature hierarchical structure found in DL models [21].

Along with the needlessness for feature extraction, the rise in popularity of DL is due to the digitization of society. More and more activities are done through computers which means more data is available. This helps DL models create a more accurate representation of the data and thus achieve better results. Figure 11 shows how the performance of DL models increases as datasets increase in size, while ML models plateau after a certain point. Additionally, the accelerating growth in Graphics Processing Units (GPUs) and Central Processing Units (CPUs) memory size and performance made it possible to create larger models, which in turn made it possible to deal with very large datasets.

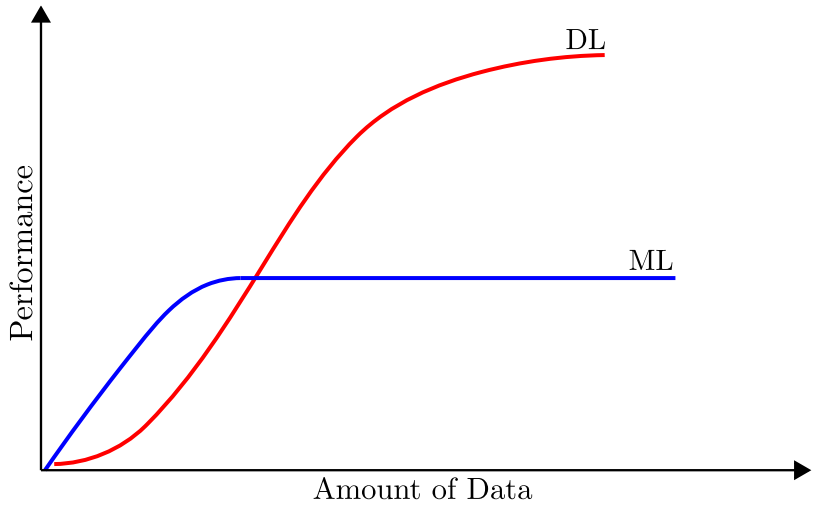

Figure 11: The relationship between performance and amount of data in the case of ML and DL models.

As mentioned before, NNs are made of layers, each containing one or more artificial neurons. These artificial neurons model biological neurons in the human brain. These neurons represent numerical values and the connections between them (or weights) are also numerical values. NNs attempt to learn these weights through the training process. Going back to Figure 9, the input layer receives an input vector $\boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$, which represents the data the NN learns from. The output layer outputs a vector $\boldsymbol{y} = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}$, which represents the predictions the NN came up with. To obtain the vector $\boldsymbol{y}$, certain operations must be done between the input and output layers in the hidden layer. To understand this, consider the simplified NN shown in Figure 12. This NN consists of two input neurons in the input layer and three output neurons in the output layer. Each connection between two neurons is expressed by a weight $w_{ij} = \{w_{11}, w_{12}, \dots, w_{23}\}$ with two indices. The first index represents which neuron the connection starts from, and the second index represents where the connection leads to. Weights in a NN can be represented in a matrix called the weight matrix $\boldsymbol{W}$. For the case shown in Figure 12, $\boldsymbol{W} = \begin{bmatrix} w_{11} w_{12} w_{13} \\ w_{21} w_{22} w_{23} \end{bmatrix}$.

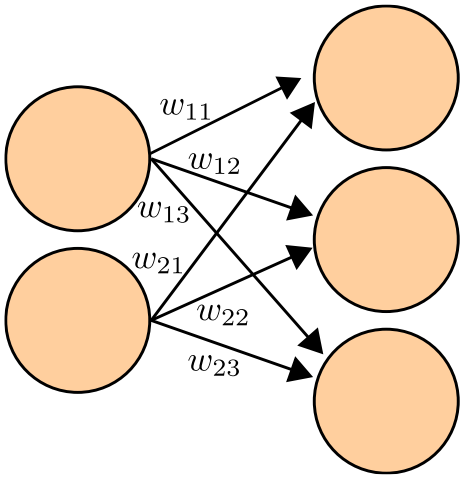

Figure 12: Simplified NN.

For an input vector $\boldsymbol{x}$, the NN calculates a prediction vector $\boldsymbol{h}$. This step is known as forward propagation. This is done by applying what is called an activation function to a vector $\boldsymbol{z}$ that is the result of a dot product between $\boldsymbol{x}^T$ and $\boldsymbol{W}$ to map it to the vector $\boldsymbol{h}$. The output is then fed to the next layer as an input. Activation functions are used to classify incoming information into useful and not-so-useful since not all information is equally important. If an activation function is not used, the NN would essentially be a linear regression model, which would make the NN simple but much less powerful. Examples of non-linear activation functions are shown in Figure 13. The Rectified Linear Unit (ReLU) can be expressed as $f_1(x) = \begin{cases} x, & x > 0 \\ 0, & x < 0 \end{cases} = \max(0, x)$. In this activation function, when the output of the linear transformation is less than zero, the neurons would be deactivated. Due to this, ReLU is more efficient than other activation functions because only a certain number of neurons are activated at a time. Another advantage is the non-saturating nature of the ReLU function, since $\lim_{x \to \infty} f_1(x) = +\infty$. Comparatively, the sigmoid function, which can be expressed as $f_2(x) = \frac{1}{1+e^{-x}}$, squashes real numbers to a range between [0,1], and the tanh function, which can be expressed as $f_3(x) = \tanh(x) = 2f_2(2x) - 1$, squashes real numbers to a range between $[-1,1]$. There is no rule of thumb when choosing activation functions, however, the ReLU function is the most widely used function and performs better than other functions in most cases. If it does not give satisfactory results, other activation functions are investigated [22].

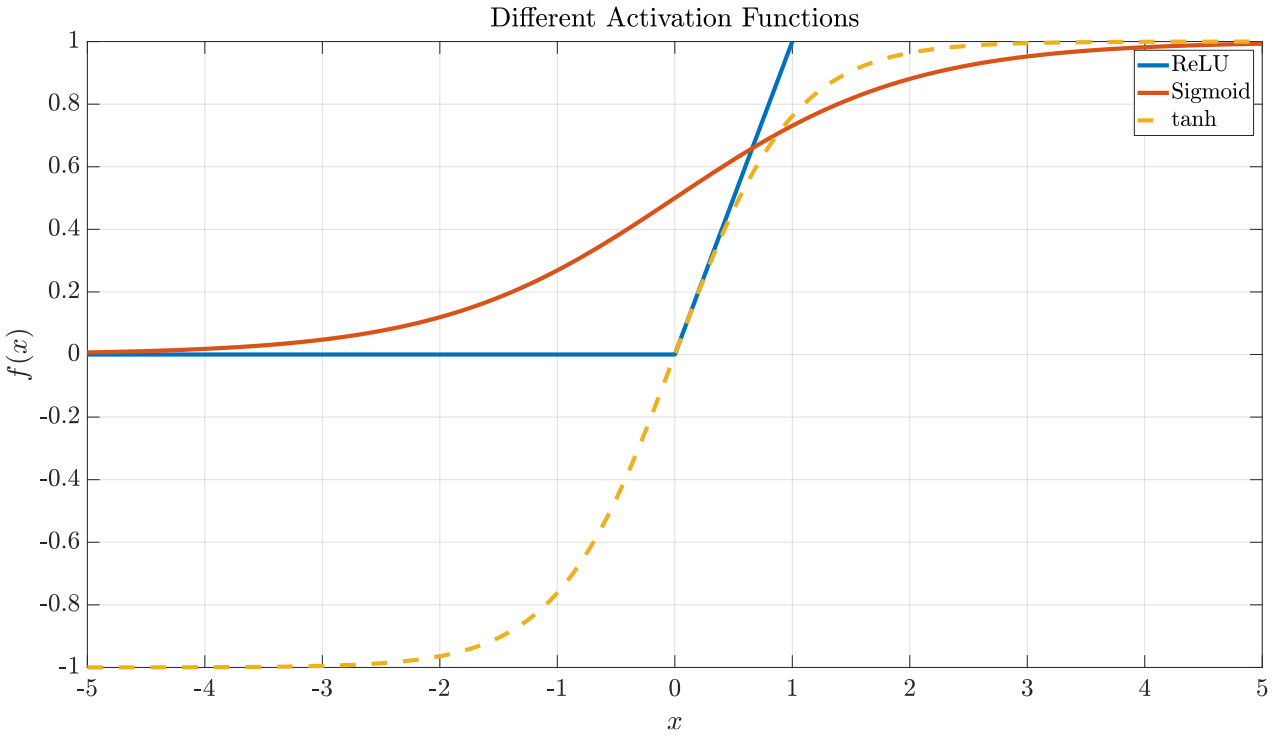

Figure 13: Commonly used activation functions in NNs.

By observing the vector $\boldsymbol{z}$, we can realize that the value of a neuron in a layer consists of a linear combination of neuron values of the previous layer weighted by some numeric values. During training, these weights are adjusted. Some neurons become more connected, while others become less connected. Accordingly, the values of $\boldsymbol{z}$, $\boldsymbol{h}$ and the final output vector $\boldsymbol{y}$ change with the weights. Some weights make the predictions of a NN closer to the actual ground truth vector $\hat{\boldsymbol{y}}$, while others make them worse.

The next step in a NN is to evaluate how close or far the predictions of $\boldsymbol{y}$ are from $\hat{\boldsymbol{y}}$. This is done through calculating the loss function J. An example of this is the Root Mean Square Error (RMSE), which is defined as the average of sum of squared differences between predictions and ground truth, expressed mathematically as

$$\text{RMSE} = \sqrt{\frac{1}{\eta}\sum_{i=1}^{\eta}(y_i - \hat{y}_i)^2}\,, \tag{4}$$

where η represents the number of examples in the dataset. Another example is the Mean Absolute Error (MAE), which is defined as the average of sum of absolute differences between predictions and ground truth, expressed mathematically as

$$\text{MAE} = \frac{1}{\eta}\sum_{i}^{\eta} |y_i - \hat{y}_i| \,. \tag{5}$$

Minimizing the value of this function directly correlates to a higher accuracy in predictions. The idea is to find a certain $\boldsymbol{W}$ that minimizes this loss which can be done through Deep Learning Optimization Algorithms (DLOAs) such as Gradient Descent (GD) [23]. In GD, the derivative of the loss function is calculated with respect to each of the weights in $\boldsymbol{W}$. This indicates where the highest rate of increase is towards. By multiplying the gradient by -1, we obtain the highest rate of decrease of the loss function and the weights are then adjusted accordingly. This is called the backward propagation step. The GD algorithm can be expressed as

$$\text{repeat until convergence } \left\{ w_{ij} \coloneqq w_{ij} - \alpha \frac{\partial J}{\partial w_{ij}} \right\}, \tag{6}$$

where α is the learning rate which controls the amount that the weights are updated during training. More significant learning rates allow the model to learn faster at the cost of arriving on a sub-optimal final set of weights, while lower learning rates enable the model to learn a more optimal or even globally optimal set of weights but may take significantly longer to train. As a rule of thumb, it is good to try values of α along the lines of 0.1, 0.01, 0.001, etc. Figure 14 illustrates GD on a hyperbola. Obviously, in DL the application of GD is much more complicated since there are sometimes millions of weights to adjust and it is impossible to reach a global minimum most of the time.

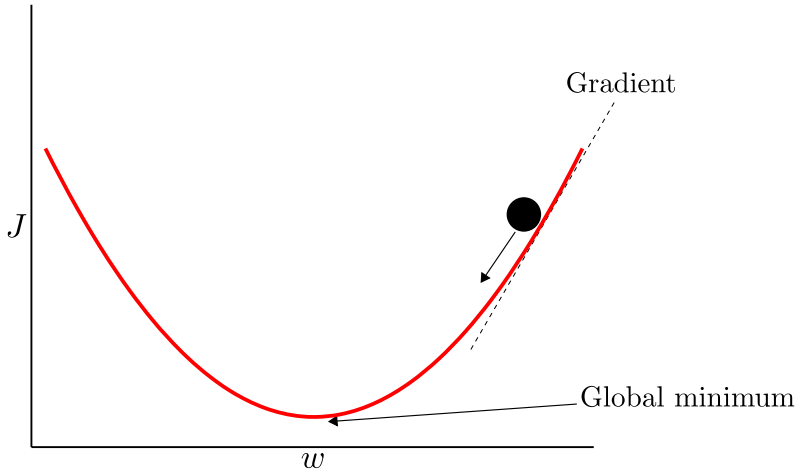

Figure 14: The process of GD.

Parameters such as α in GD are considered model hyperparameters, not to be confused with model parameters. Model hyperparameters are external to the DL model and their value cannot be estimated from the data. Typically, the values of hyperparameters control the learning process, are specified by the practitioner, and are set using heuristics from rules of thumb, copying values used in similar problems, or from hyperparameter tuning techniques. Model parameters on the other hand are internal to the DL model and are estimated from the data. They are required when the model tries to make predictions and are not set by the practitioner. An example of model parameters that is relevant to DL models is the weights associated with artificial neurons.

After deciding on the model architecture, activation function, and tuning hyperparameters, the dataset is split into training data, validation data, and testing data. The training data is used to fit the model. The DL model sees this data and learns from it. On the other hand, the validation data is used for frequent evaluation of the model. The DL model sees this data but does not learn from it. The testing data is used for an unbiased, final assessment of the model. The DL model only sees this data when the training process is completed. Typically, the dataset is split into training and testing data with some ratio, and a part of the training data is set aside for validation purposes.

The entire training data (or an epoch) is fed multiple times through a NN. This is due to the fact that learning is an iterative process and the amount of data is limited, so calculating weights from a single passing of the dataset is not enough and will cause underfitting, meaning the model fails to generalize well from the training data and hence does not fit it adequately.

Since an entire epoch is too large to feed to the NN at once, it is usually split into what is known as batches. After each batch, the DL model is updated. A hyperparameter that is related to batches is the batch size, which represents how much of the data is in a single batch. Additionally, it is considered good practice to shuffle the data on every new epoch. This helps the training converge faster, mitigates bias during training, and prevents the model from learning the order of training and hence overfitting the training data.

Due to the high complexity in the structure of NNs, chances are the model will overfit the data. This means that the model learns the training data too well and does not generalize as well as it should to data it has not seen yet. To prevent this, the complexity of the NNs is reduced, or, more commonly, regularization is used. Some of these techniques include L1 and L2 regularization. In both of these, a regularization term is introduced to the cost function, causing a reduction in $\boldsymbol{W}$. In L1 regularization, $\text{Cost} = \text{Loss} + \lambda \sum|w|$. This means that weights could be reduced to zero in some cases which is helpful for compressing DL models. On the other hand, in L2 regularization, $\text{Cost} = \text{Loss} + \lambda \sum|w|^2$. The weights are driven towards zero, but are not exactly zero. It should be noted that λ, the regularization parameter, is the hyperparameter that controls the amount of regularization and hence, needs to be tuned [24].

Other techniques include early stopping and dropout. In early stopping, part of the dataset is used as a validation set. Once the performance of the DL model over this set starts degrading, training is stopped [24]. Figure 15 illustrates this process.

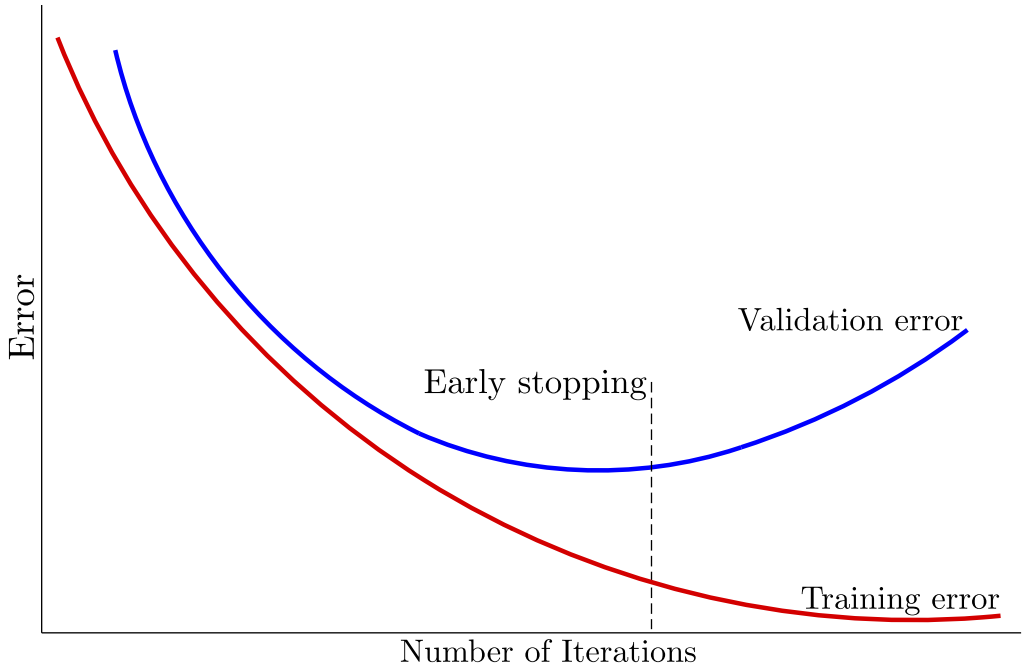

Figure 15: The process of early stopping.

In dropout, non-output random nodes in the NN are turned off along with the connections from and to them at a certain probability σ which is very similar to ensembles in classic ML algorithms. This probability is also considered a hyperparameter that needs to be tuned. Figure 16 shows this process [24].

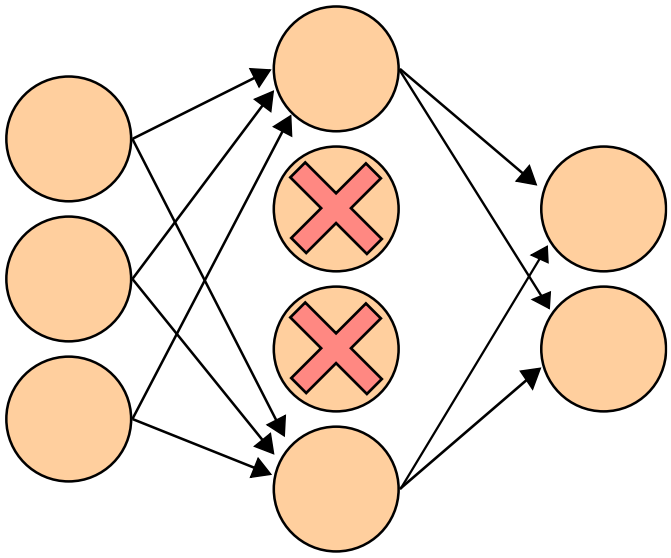

Figure 16: The process of dropout.

Due to the large size of DL models and the sheer amount of data, it is vital to use a proper DLOA such that good results are achieved within a reasonable amount of time. The literature offers many DLOAs. Some of the most popular ones include Stochastic Gradient

Descent (SGD) [25], Root Mean Square Propagation (RMSprop) which was proposed by Geoff Hinton in lecture 6 of the online course "Neural Networks for Machine Learning" on Coursera, and Adaptive Moment Estimation (Adam) [26].

SGD fixes some downsides with GD, namely the fact that there is a redundancy in the number of calculations. To put things into perspective, for a dataset with 1,000 examples and 10 features, the squared difference has to be calculated 10,000 times per iteration. This is typically considered as a lot of overhead and makes GD converge slowly on big datasets. The idea behind SGD is to calculate the gradient stochastically by using a subset of the data instead of all of it. Stochastic Gradient Descent with Momentum (SGDM) [27] makes use of exponentially weighed averages for this purpose.

The magnitude of the gradient can be very different for different weights and can change during learning, which makes it hard to choose a single global learning rate. RMSprop is an adaptive learning rate method that solves this issue by dividing the learning rate for a weight by a running average of the magnitudes of recent gradients of that weight. This makes the algorithm work exceptionally well on online and non-stationary problems.

Adam is considered as an algorithm that combines the advantages of both RMSprop and Adaptive Gradient Algorithm (AdaGrad) [28], which is a DLOA that adapts the learning rate to the weights using smaller learning rates for weights associated with frequently occurring features and higher learning rates for weights associated with infrequent features, which makes this algorithm work best with problems that have sparse gradients. The idea behind Adam is to calculate an exponentially decaying moving averages of the gradient and the squared gradient by making use of the first and second moments.

The literature also offers a lot when it comes to DL model architectures. Most notably Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs). CNNs are mainly concerned with image processing (image recognition, image classification, object detection, etc.), but they are sometimes used for classifying non-image data such as audio and signal data. Filters are applied to each example at a different resolution and each convolved image is used as an input for the next layer [29]. RNNs on the other hand are an architecture that uses past information to improve the performance of the DL model on current and future inputs through looping information in hidden states, though simple RNNs face issues remembering what they learn in the long term. This gave rise to Long Short-term Memory

(LSTM) networks which use gates to control what information in hidden states make it to the output and the next hidden state [21].

2.3 *Optimization Fundamentals*

Mathematical optimization of a specific function (called an objective function) is the process of finding variables $\boldsymbol{x}^*$ within a certain set (called the feasible set F) that best minimize said function [30]. This can be formulated mathematically as

$$\boldsymbol{x}^* = \underset{\boldsymbol{x} \in \mathrm{F}}{\operatorname{argmin}} f(\boldsymbol{x}), \tag{7}$$

When the optimization variables are not limited to a certain set (i.e., $\mathrm{F} = \mathbb{R}^N$), the problem is called an unconstrained optimization problem. Otherwise, the problem is called a constrained optimization problem. Maximizing $f(\boldsymbol{x})$ could be achieved by minimizing $-f(\boldsymbol{x})$ such that

$$\boldsymbol{x}^* = \underset{\boldsymbol{x} \in \mathrm{F}}{\operatorname{argmin}} - f(\boldsymbol{x}). \tag{8}$$

One of the most critical issues in the optimization process is getting stuck at a local minimum or a saddle point. A local minimum of a function is a point at which the function has the smallest value within a certain range. One way of finding local minima is by using Fermat's theorem, which states that the gradient of a function at local minima must be zero. A saddle point is a point at which the function has zero gradient, but is not considered a local extremum of that function. Figure 17 illustrates the difference between a local minimum and a global minimum.

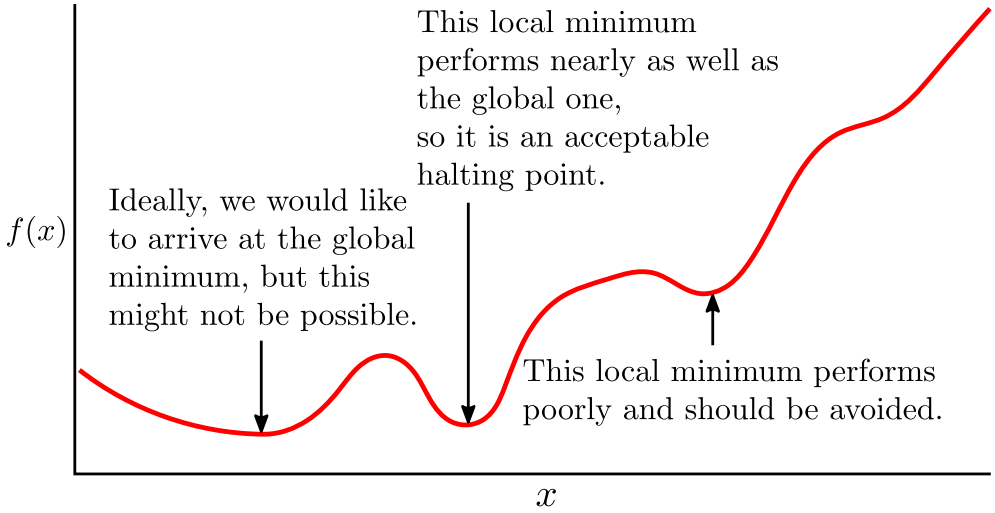

Figure 17: The difference between a local minimum and a global minimum [21].

Optimization problems can be classified into convex and non-convex problems [30]. A convex optimization problem is one where a convex function is optimized over a convex feasible set. $f(\boldsymbol{x})$ is a convex function if the following holds for all $\boldsymbol{x}$ and $\boldsymbol{y}$

$$f(\iota\boldsymbol{x} + (1-\iota)\boldsymbol{y}) \leq \iota f(\boldsymbol{x}) + (1-\iota)f(\boldsymbol{y}), \tag{9}$$

where $\iota \in [0,1]$. Additionally, convex functions have a positive semi-definite Hessian matrix $\boldsymbol{H}$ (i.e., all of the eigenvalues of the Hessian matrix have a non-negative value). The Hessian matrix $\boldsymbol{H_f}(\boldsymbol{x})$ of the function $f(\boldsymbol{x})$ is defined as

$$\boldsymbol{H}_f(\boldsymbol{x}) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix}. \tag{10}$$

If $f(\boldsymbol{x})$ is a convex function, then $-f(\boldsymbol{x})$ is a concave function. Figures 18-20 provide an example of a convex function, a concave function, and a nonconvex-nonconcave function. It should be noted that the local minima of a convex function are also global minima. Thus, convex functions have the advantage of not getting stuck at a local minimum.

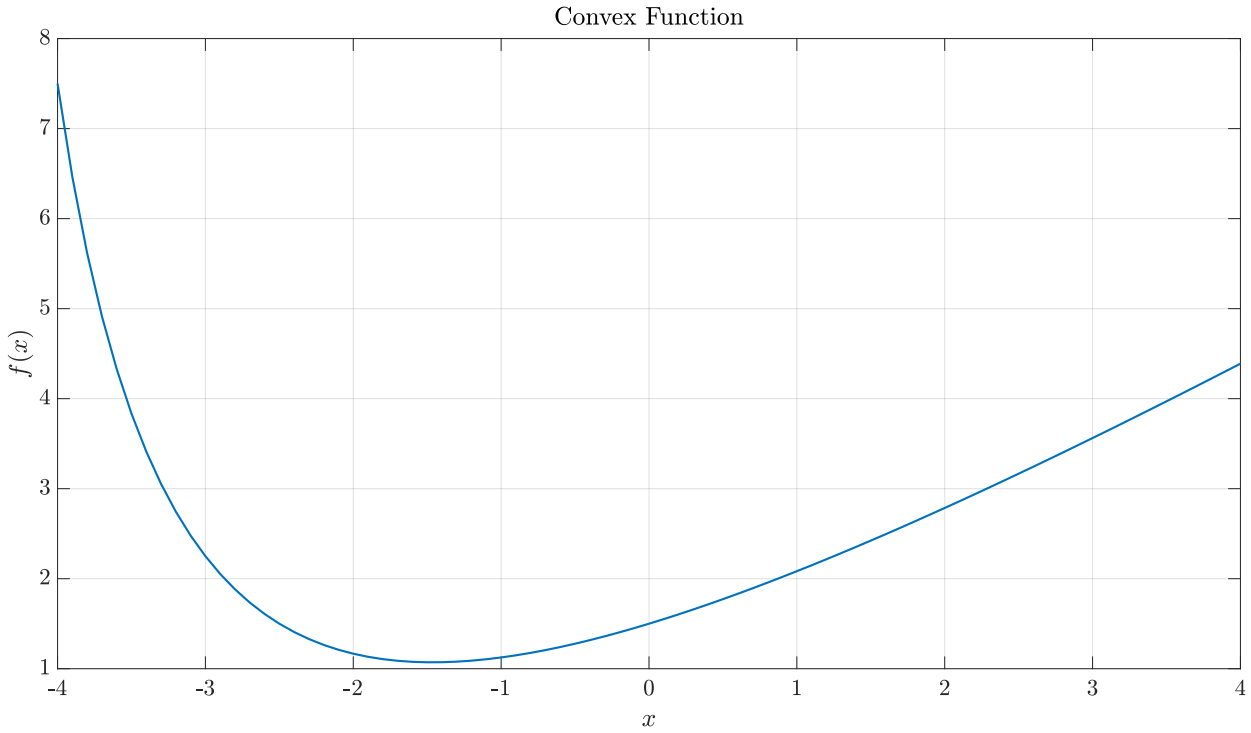

Figure 18: An example of a convex function.

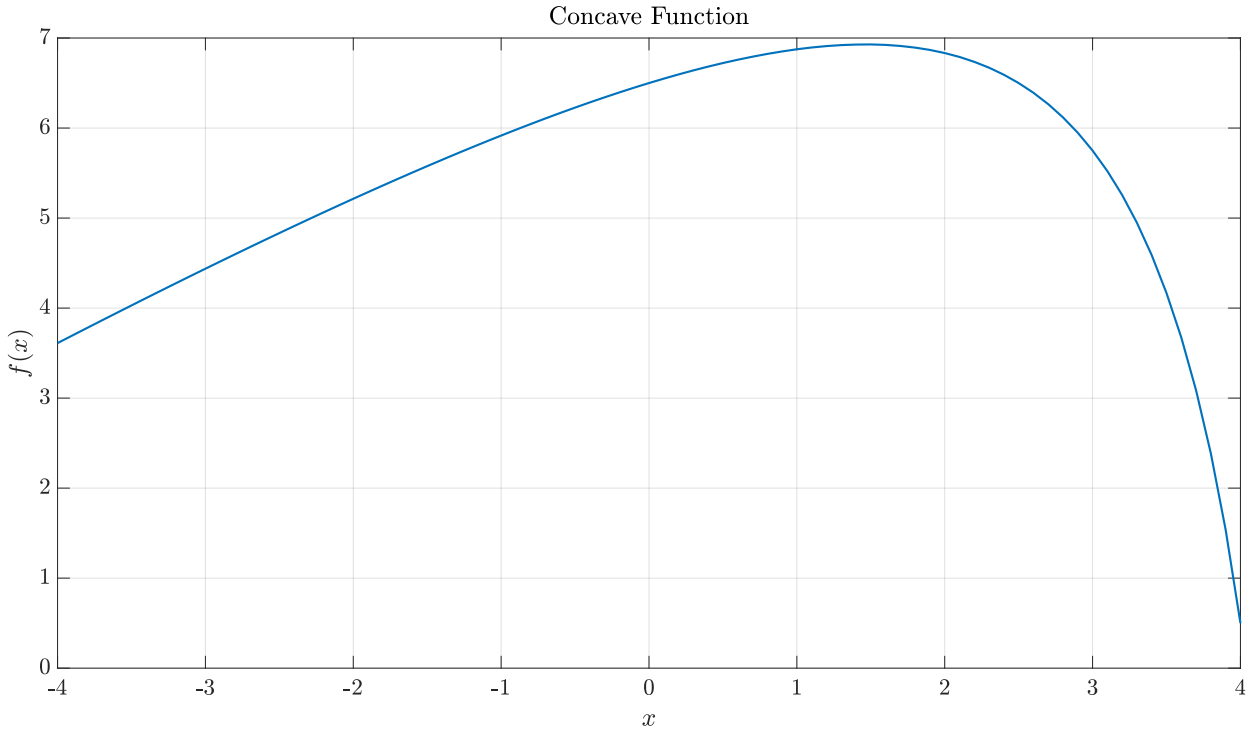

Figure 19: An example of a concave function.

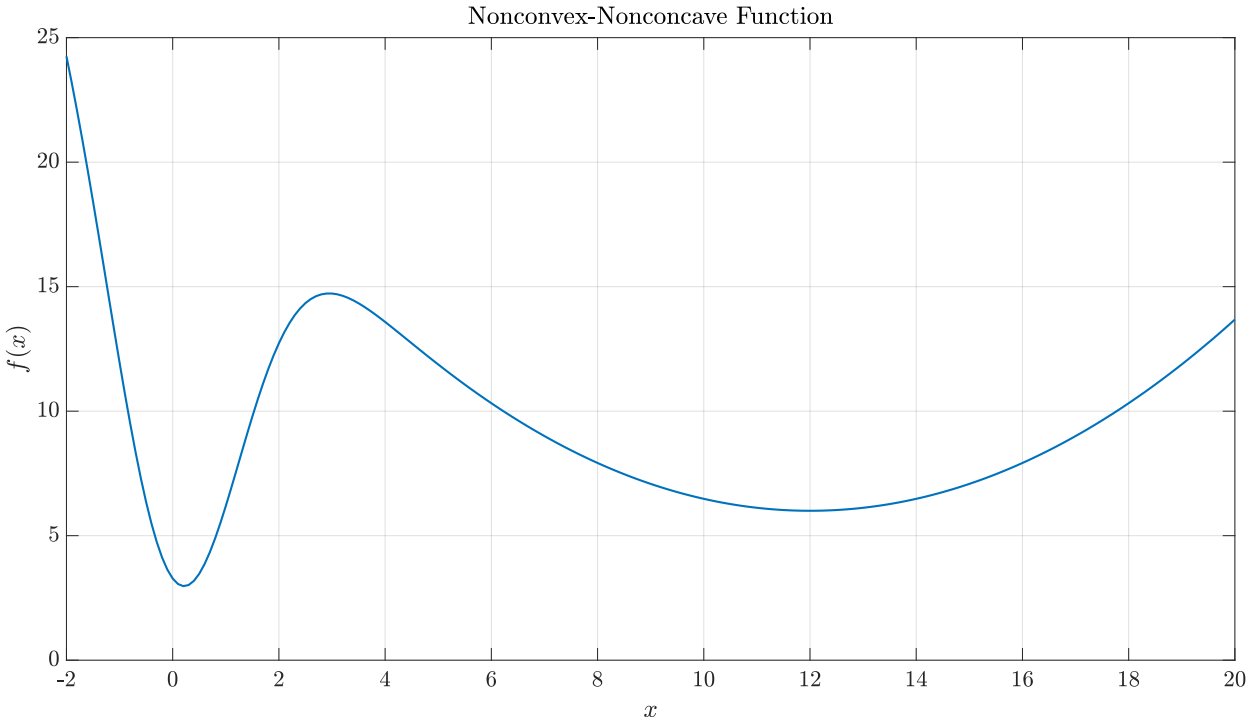

Figure 20: An example of a nonconvex-nonconcave function.

Although some optimization problems can be solved in a closed-form manner by equating the gradient with zero, most of them do not have this luxury, even if they are convex. For this reason, most problems are solved iteratively. One of the most crucial iterative optimization algorithms is the GD algorithm which was discussed earlier in 2.2 The GD algorithm is considered a first order optimization algorithm since it only uses first-order derivatives.

Second-order optimization algorithms make use of the second derivative of the objective function [31]. An example of this is Newton's method, which starts off by quadratically approximating the objective function around an initial point $\boldsymbol{x}^{(0)}$. Then, it solves for the critical point of this function from

$$\boldsymbol{x}^{*} = \boldsymbol{x}^{(0)} - (\boldsymbol{H}_{f}(\boldsymbol{x}^{(0)}))^{-1}\nabla_{\boldsymbol{x}} f\left(\boldsymbol{x}^{(0)}\right), \tag{11}$$

where $\boldsymbol{H}_{f}(\boldsymbol{x}^{(0)})$ is the Hessian matrix computed at $\boldsymbol{x}^{(0)}$ and $\nabla_{\boldsymbol{x}} f\left(\boldsymbol{x}^{(0)}\right)$ is the gradient of the objective function computed at $\boldsymbol{x}^{(0)}$. This critical point is now considered the initial point for the next iteration. The process continues until the algorithm converges to a global minimum. Figure 21 graphically illustrates Newton's method.

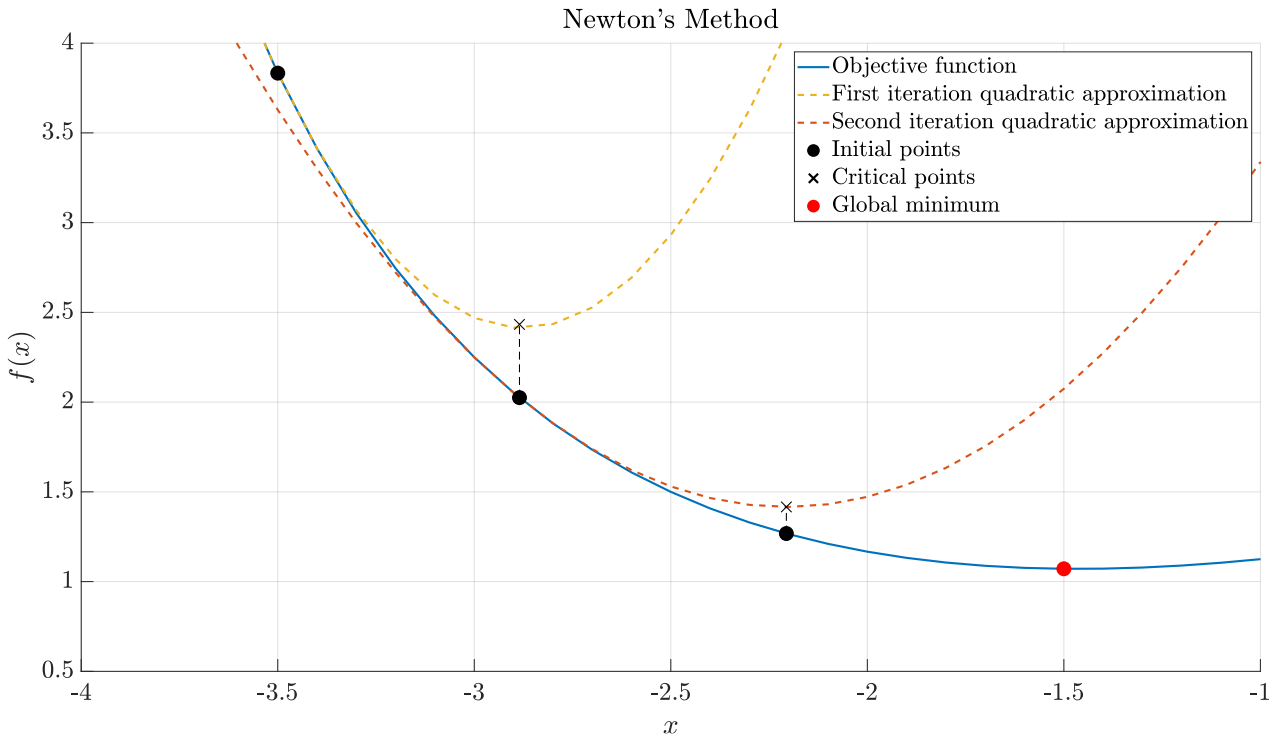

Figure 21: Illustration of Newton's method on a convex function.

One of the main drawbacks of Newton's method is that it makes use of critical points, which means that the algorithm may get stuck at a local minimum, local maximum, or saddle point. Newton's method can be applied effectively to convex optimization problems because all critical points are considered to be global minima. However, Newton's method cannot be applied directly to non-convex problems. A common way to overcome this problem is by adding a constant term Ω along the diagonal of the Hessian matrix to ensure that the Hessian matrix is modified to be a positive semi-definite matrix.

$$\boldsymbol{x}^{*} = \boldsymbol{x}^{(0)} - (\boldsymbol{H}_{f}(\boldsymbol{x}^{(0)}) + \Omega \boldsymbol{I})^{-1} \nabla_{\boldsymbol{x}} f\left(\boldsymbol{x}^{(0)}\right), \tag{12}$$

In case the Hessian matrix is already a positive semi-definite matrix, Ω will be zero. If the Hessian matrix has a large negative eigenvalue, Ω will be a large positive scalar. This means that the parameters will be updated towards the direction of the negative gradient such that

$$\boldsymbol{x}^{*} \approx \boldsymbol{x}^{(0)} - \frac{1}{\Omega} \nabla_{\boldsymbol{x}} f\left(\boldsymbol{x}^{(0)}\right). \tag{13}$$

Optimization problems can also be classified into discrete and continuous problems. Unlike continuous optimization problems, discrete ones have a countable feasible set. In other words, the optimizable variables are restricted only to discrete values.

Discrete optimization has three notable branches [32]. First, integer programming, in which the optimizable variables are restricted to be integer values. Second, combinatorial optimization, in which the feasible set is finite and countable, but the size of the feasible set is tremendous. Well-known problems in combinatorial optimization include the Travelling Salesman Problem (TSP) and the Minimum Spanning Tree (MST) problem. Third, constraint programming, which is very similar to combinatorial optimization, except the constraints of the optimization problem are explicitly stated.

Optimization algorithms used to solve optimization problems can be classified into deterministic and random algorithms. Parameters are updated deterministically in deterministic methods and semi-stochastically in random methods. Examples of random optimization methods include Simulated Annealing (SA) [33], Ant Colony Optimization (ACO) [34], and Random Search (RS). For instance, in SA, random parameters are generated around current parameters. Current parameters are then always replaced with generated parameters if the objective function has a smaller value at those generated parameters compared to the current parameters. If the objective function has a larger value, the new parameters will be selected with some probability, giving it the ability to avoid being trapped in local minima.

As stated before, being trapped in local minima is one of the most challenging issues to face when trying to optimize non-convex functions. Some optimization algorithms have been designed in order to overcome this issue. SGDM which was discussed earlier in 2.2 achieves this by averaging the gradients of past iterations.

Graduated optimization (or continuation) [35], shown in Figure 22, is another method for finding the global minimum of a non-convex function. This is done through optimizing multiple functions that are smoothed versions of the objective function (by utilizing filters such as the Gaussian filter) and using the solution of the current function as the initial point for the optimization of the next function. The process is repeated iteratively until the current function matches the objective function.

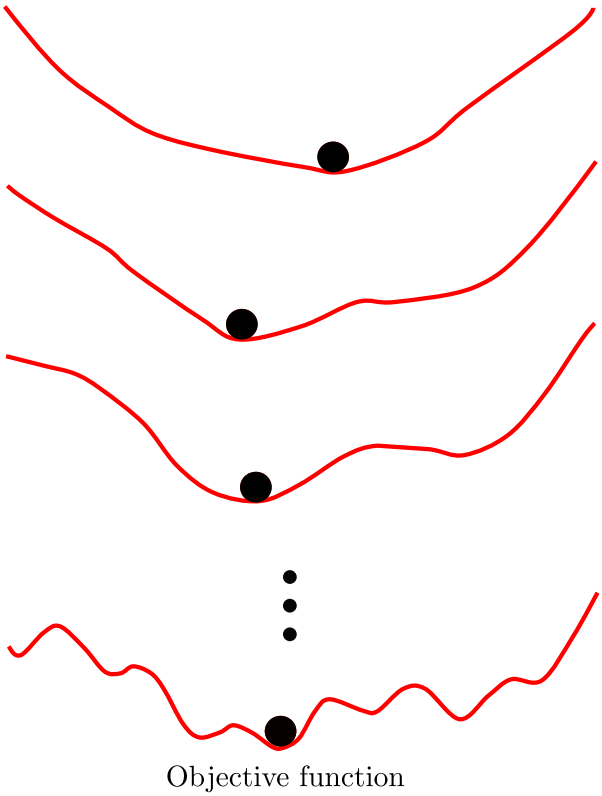

Figure 22: Graduated optimization of a non-convex function [36].

Chapter 3 DESIGN OPTIONS

3.1 *Design Options*

Three major design fields are approached in this project: data generation methods, deep learning model and optimization algorithm.

The need for a simulator arose from wanting to create a low-cost, flexible environment. Going with the experimental approach would have required a large, warehouse-like room to perform the experiments in along with relatively expensive hardware. It would have also been tiresome or even impossible to try to change aspects of the environment (i.e., the size of the room, the materials that make up the walls/floor, the distribution of objects inside the room, the type of reference beacons, the locations of reference beacons, etc.). The most notable advantage the experimental approach offers over the simulation one is more realistic measurements and while that is true, it could be remedied by using accurate models to generate measurements that are as close as possible to those resulting from experimenting. Simulation tools which are suitable for indoor navigation are scarce and extremely basic and under-developed, mostly developed by individual researchers to satisfy their research.

MATLAB software has been chosen for generating data, mainly due to its helpful packages and complimentary visualization tools, in addition to our previous knowledge and familiarity with MATLAB. Going through the simulation option introduces more sub options to carry on the simulation, such as choosing the suitable path planning algorithm and ray tracing algorithm, and an appropriate graphic design software for 3D representation of the environment. The choices and selection criteria are further explained in 3.1.

Position estimation methods are abundant and diverse, with each method optimized for certain conditions; one of the most widely used methods is Kalman Filters, which are used either independently or in conjunction with other methods. Other position estimation techniques such as classic machine learning techniques and other numerical analysis tools are also used in the industry, but are suitable given lower accuracy demands and lower computing capabilities.

For optimizing the distribution of beacons, many algorithms have been investigated, including random optimization algorithms, discrete optimization algorithms, and various continuous

optimization algorithms. The process and method carried out is outlined and further explained in 3.3.

3.2 *Design Constraints and Standards*

2.2.1 Cost Constraint

The primary constraint of the design is cost feasibility, the navigation system should be cost effective by offering the optimal functionality with the least cost. Keeping costs to minimum is the driving force behind optimizing the number of beacons needed and the optimal distribution of these beacons. Costs are divided into overhead costs; the initial costs required for deploying the beacons and the necessary infrastructural modifications, in addition to the maintenance costs.

2.2.2 Environmental Constraints

The working environment has been assumed to be a warehouse, with shelves and passages distributed in an open space. This assumption has steered and confined the process of generating data and optimizing the locations of beacons. Such constraint guided us in choosing simulation tools and environments as they should allow us to manipulate the type and intensity of noise according to the desired environment. For example, certain environments have an intense magnetic noise, which can be accounted for in our simulation.

2.2.3 Viability Constraint

Viability has constrained the choice of electromagnetic communication method, and narrowed the options to WIFI, Bluetooth and RFID units.

2.2.4 Time Constraint

Time available is also a detrimental constraint of the design of the system, as the system should be designed within roughly seven months and given the limited experience and knowledge of the designing team, the expected outcomes have been modified in accordance with this constraint.

Chapter 4 METHODOLOGY

4.1 *Generating Data*

The need for a simulator arose from wanting to create a low-cost, flexible environment. Going with the experimental approach would have required a large, warehouse-like room to perform the experiments in along with relatively expensive hardware. It would have also been tiresome or even impossible to try to change aspects of the environment (i.e., the size of the room, the materials that make up the walls/floor, the distribution of objects inside the room, the type of reference beacons, the locations of reference beacons, etc.). The most notable advantage the experimental approach offers over the simulation one is more realistic measurements and while that is true, it could be remedied by using accurate models to generate measurements that are as close as possible to those resulting from experimenting.

We considered a $32 \times 32 \times 7$ meters warehouse floor with three small storage racks, two large storage racks, and four doors as shown in Figure 23 (the roof was removed for viewing purposes). The robot enters the warehouse floor, moves between the racks, carries several boxes that are on top of the racks, then leaves the warehouse floor. The walls and floor are assumed to be made out of metal. The robot's receiver is at a height of one meter and the nine reference beacons are 10 centimeters below the ceiling. The reference beacons numbered from 1 to 9 were placed at $(x, y) = (5, 5), (5, 16), (5, 26), (16, 5), (16, 16), (16, 26), (27, 5), (27, 16)$, and $(27, 26)$, respectively, as shown in Figure 24.

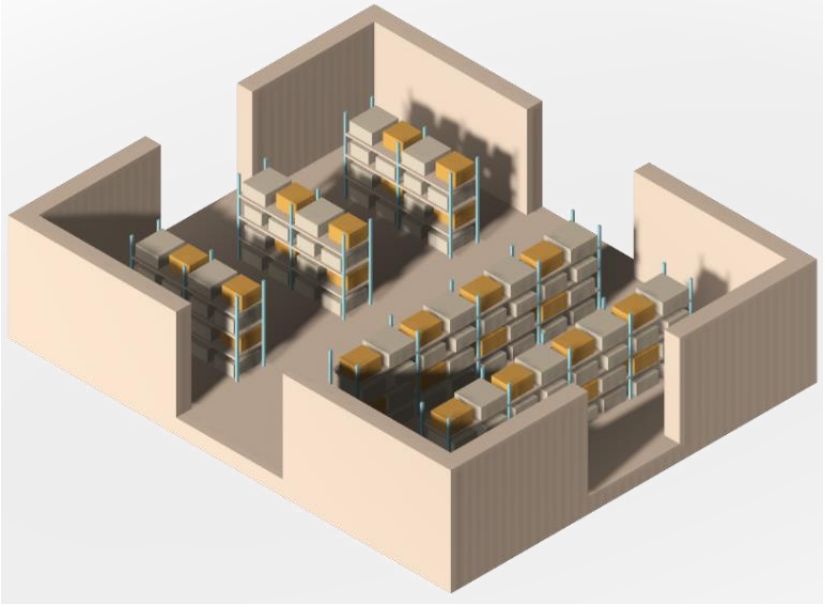

Figure 23: The warehouse floor 3D map.

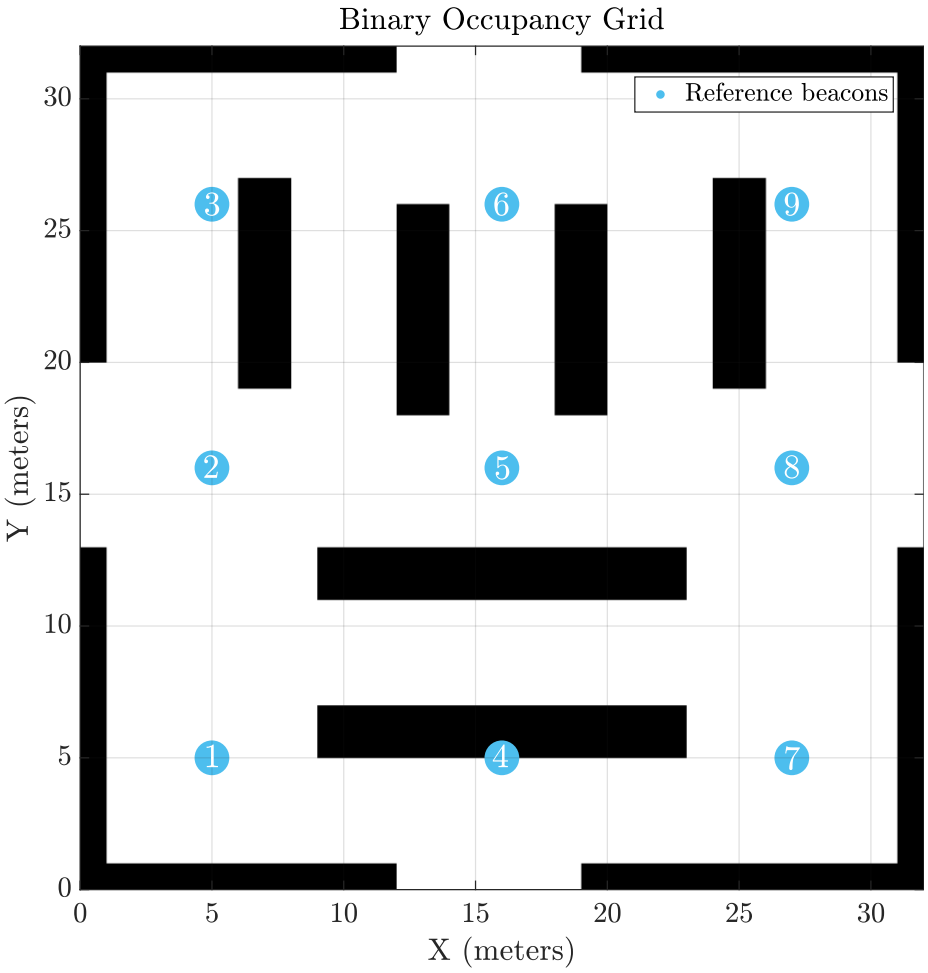

Figure 24: The warehouse floor 2D map.

The 3D map was built using the free, online 3D modelling tool Tinkercad from Autodesk [37]. This map will be used later on to simulate the RSSI measurements of the reference beacons. The 2D map is an object in MATLAB called a binary occupancy map. It will be used later to generate paths for the robot to move in and consequently accelerometer and gyroscope readings that correspond to said movement.

The simulator first attempts to plan a path for the robot. These paths consistently have four waypoints, always starting and ending with one of the four doors. The two remaining points were chosen to be points of interest around the warehouse floor. Uniform distributions of points around each of the points of interest and the doors' entrances were created such that the path planner does not create paths along the same waypoints every time. These distributions are highlighted in pink and shown in Figure 25.

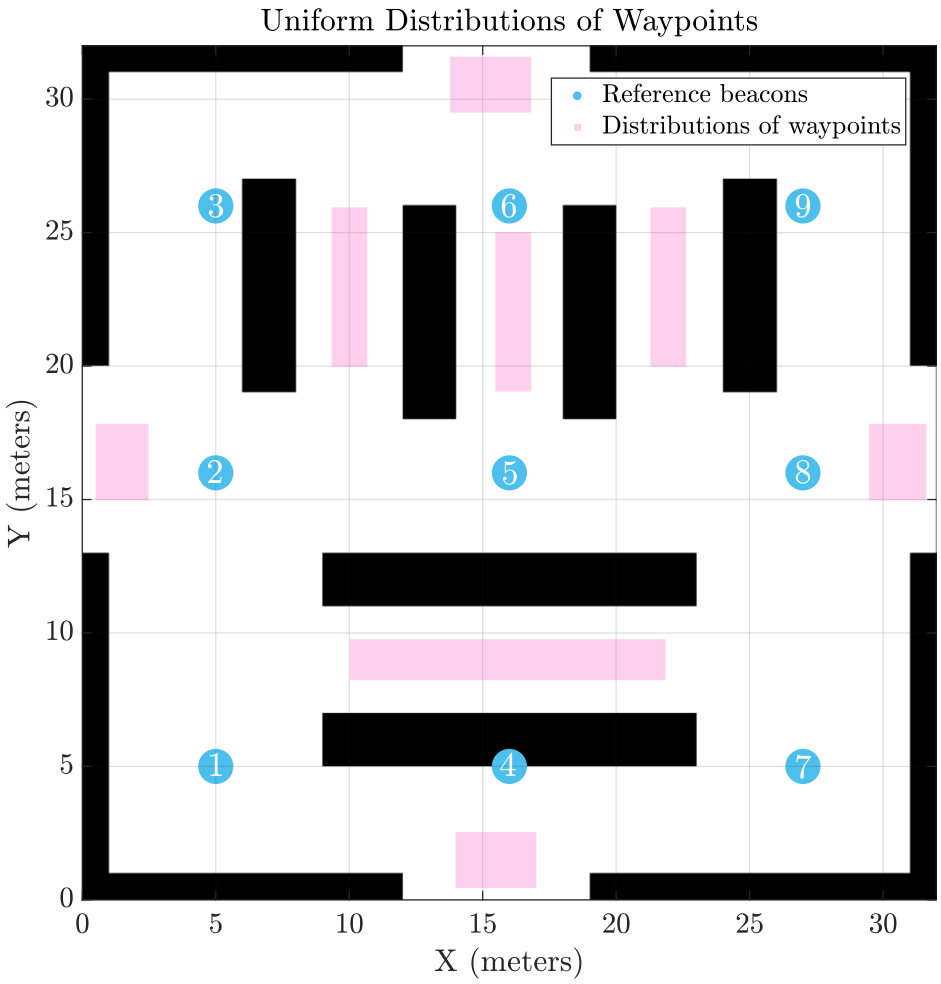

Figure 25: The uniform distributions of waypoints.

The literature offers many path planning algorithms, but we narrowed down our choices to three tree path planners: random tree, Rapidly-exploring Random Tree (RRT), and RRT*. In a random tree, as shown in Figure 26, a starting node is selected as the base of a tree and it is required to find a path from that node towards a goal. An edge is then added from that node in a random direction as long as no obstacles are in the way. This continues iteratively by adding more nodes around the starting node and adding edges from a random node until a path between the starting node and the goal is created. Random trees are slow at exploring the entire space and therefore require a very long time to converge.

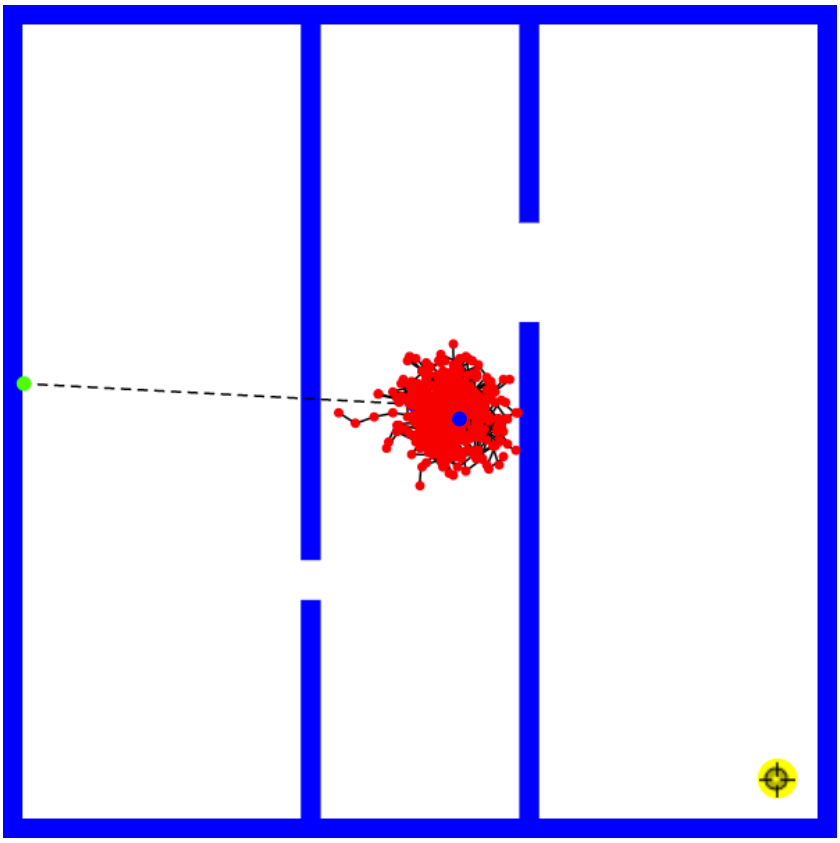

Figure 26: Tree expansion after 500 iterations. The blue dot represents the base of the tree, the red dots represent the added nodes, the green dot represents the randomly selected node, and the yellow dot with a cross inside of it represents the goal.

RRT, proposed in [38], is similar to random tree, except when choosing the edge to expand from towards the random node, it selects the one closest to that random node, as shown in Figure 27. The resulting path is usually jagged (not optimal) and adding many nodes does not result in a shorter path.

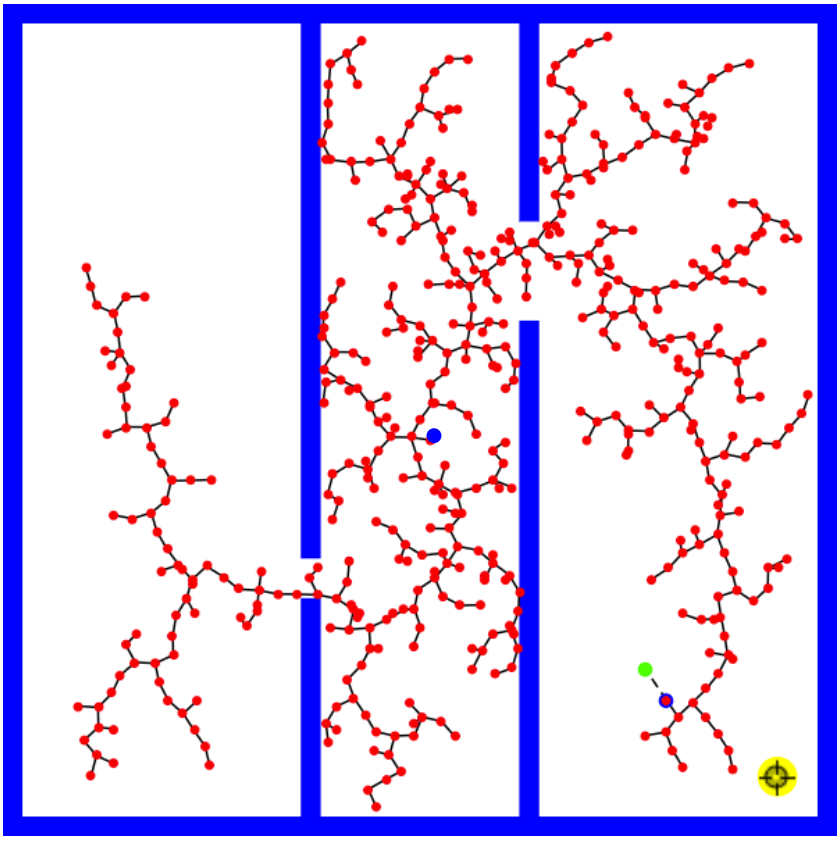

Figure 27: RRT after 500 iterations. The blue dot represents the base of the tree, the red dots represent the added nodes, the green dot represents the randomly selected node, and the yellow dot with a cross inside of it represents the goal.

To fix this, RRT* was proposed in [39]. In this algorithm, as the number of nodes approaches infinity, the shortest possible path will be found. A parameter called cost is defined representing the distance each vertex has traveled relative to its parent vertex. Each time the closest node to the random node is selected, a circle is drawn to check if any nearby nodes have a lower cost. If that is the case, that node is selected instead.v

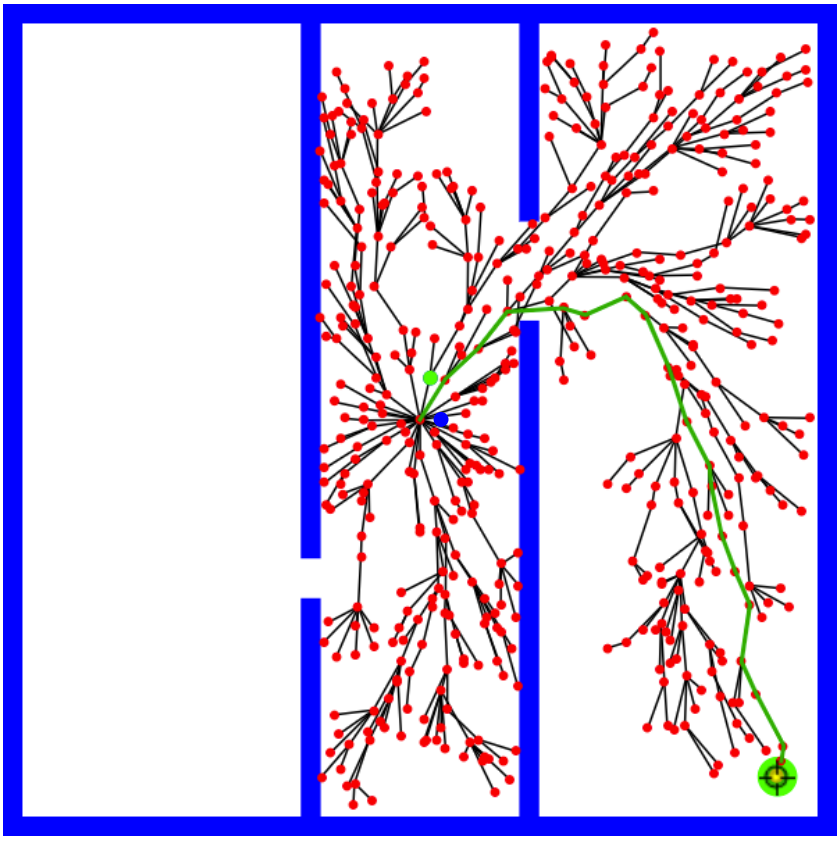

Figure 28: RRT* after 500 iterations. The blue dot represents the base of the tree, the red dots represent the added nodes, the green dot represents the randomly selected nodes, and the yellow dot with a cross inside of it represents the goal. The goal was reached and its length was 38.69 units.

Additionally, neighbors to the new node are checked to see if wiring them to that node would decrease the cost or not. If that is the case, the neighbors are rewired to the newly added node. This produces much smoother paths that gradually decrease in cost at a price of lower convergence speed. Since this algorithm treats the vehicle as a point, it was necessary to inflate the map [40] to account for the vehicle's geometry, as shown in Figure 29.

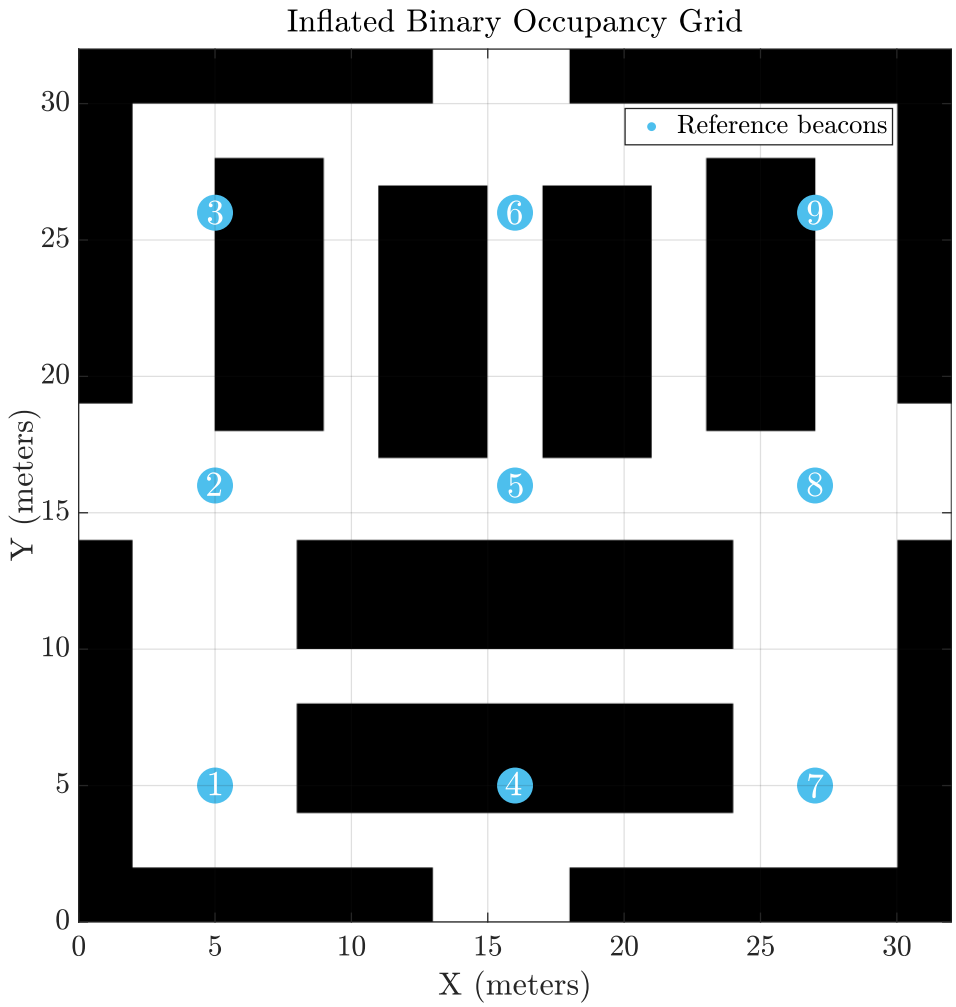

Figure 29: The inflated binary occupancy map.

The pure pursuit path tracking algorithm [41] is then used to calculate the linear and angular velocities of the robot along the generated path given that the pose (position and orientation) of the vehicle is known. In this algorithm, the curvature of a circular arc δ that connects the rear axle of the vehicle to a goal point at $(x, y) = (x_g, y_g)$ ahead of the vehicle on the path is geometrically calculated. This goal point is determined from a lookahead distance l_d, as shown in Figure 30.

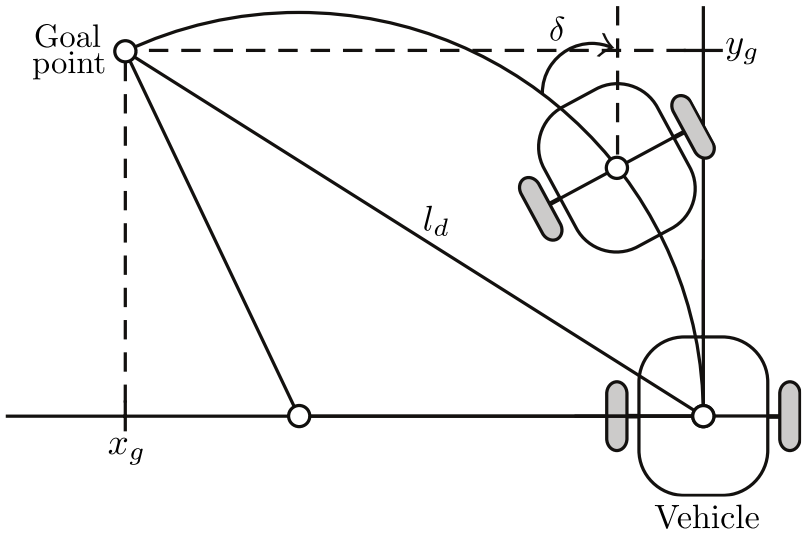

Figure 30: The geometry of the pure pursuit algorithm.

Selecting a suitable lookahead distance is crucial. A small lookahead distance will result in accurate but oscillatory paths, while a large lookahead distance will result in smoother paths but poor tracking, as shown in Figure 31.

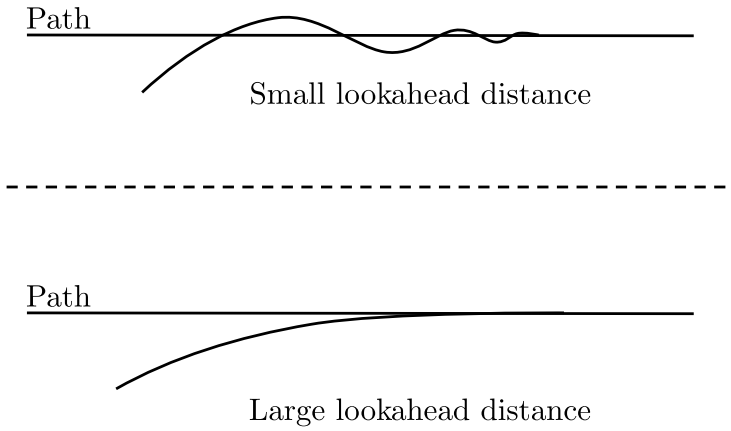

Figure 31: The effect of using small and large lookahead distances.

Figure 32 shows the generated paths for the robot for a single instance. Planning the entire path is done in three steps. First the RRT* path from waypoint A to waypoint B is found, then from B to C, and finally from C to D. For example, the planned path from A to B is shown in Figure 33. After that, the total resulting path is fed to the pure pursuit algorithm.

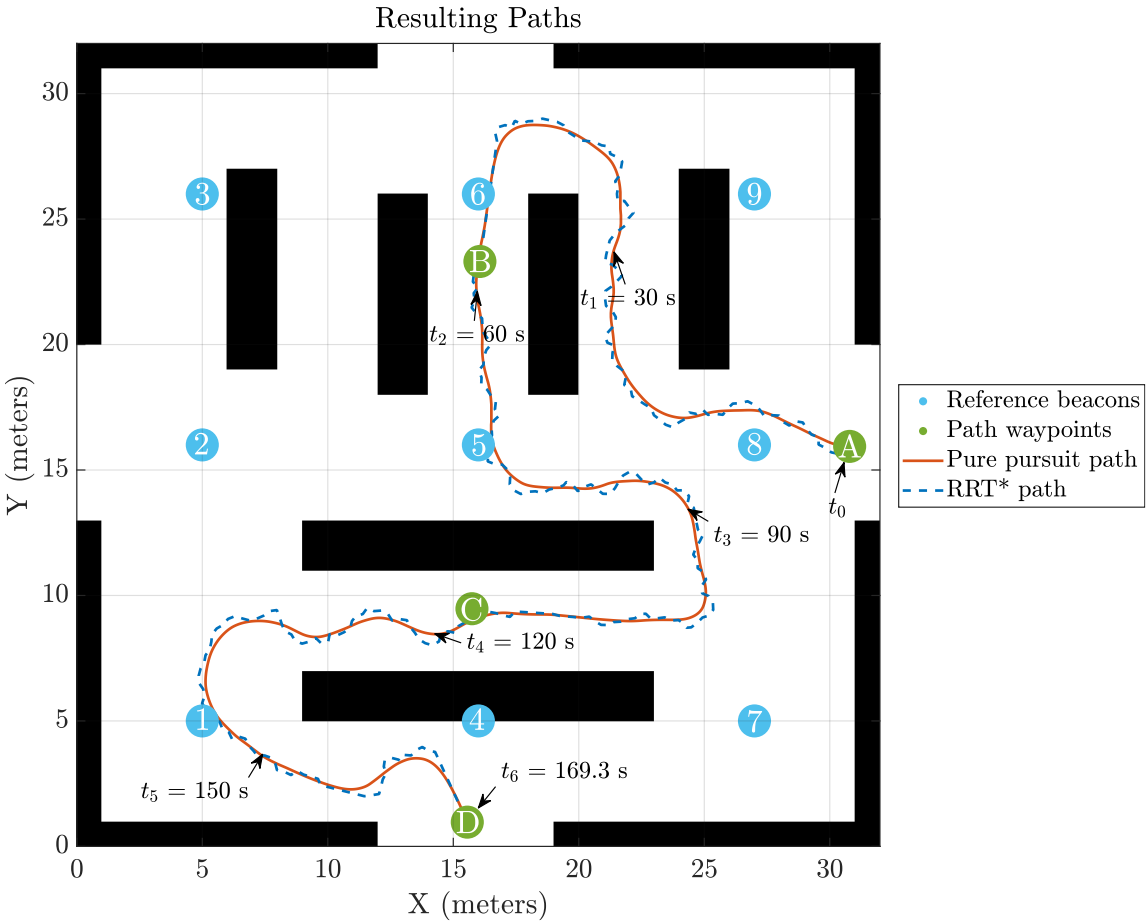

Figure 32: One of the generated paths.

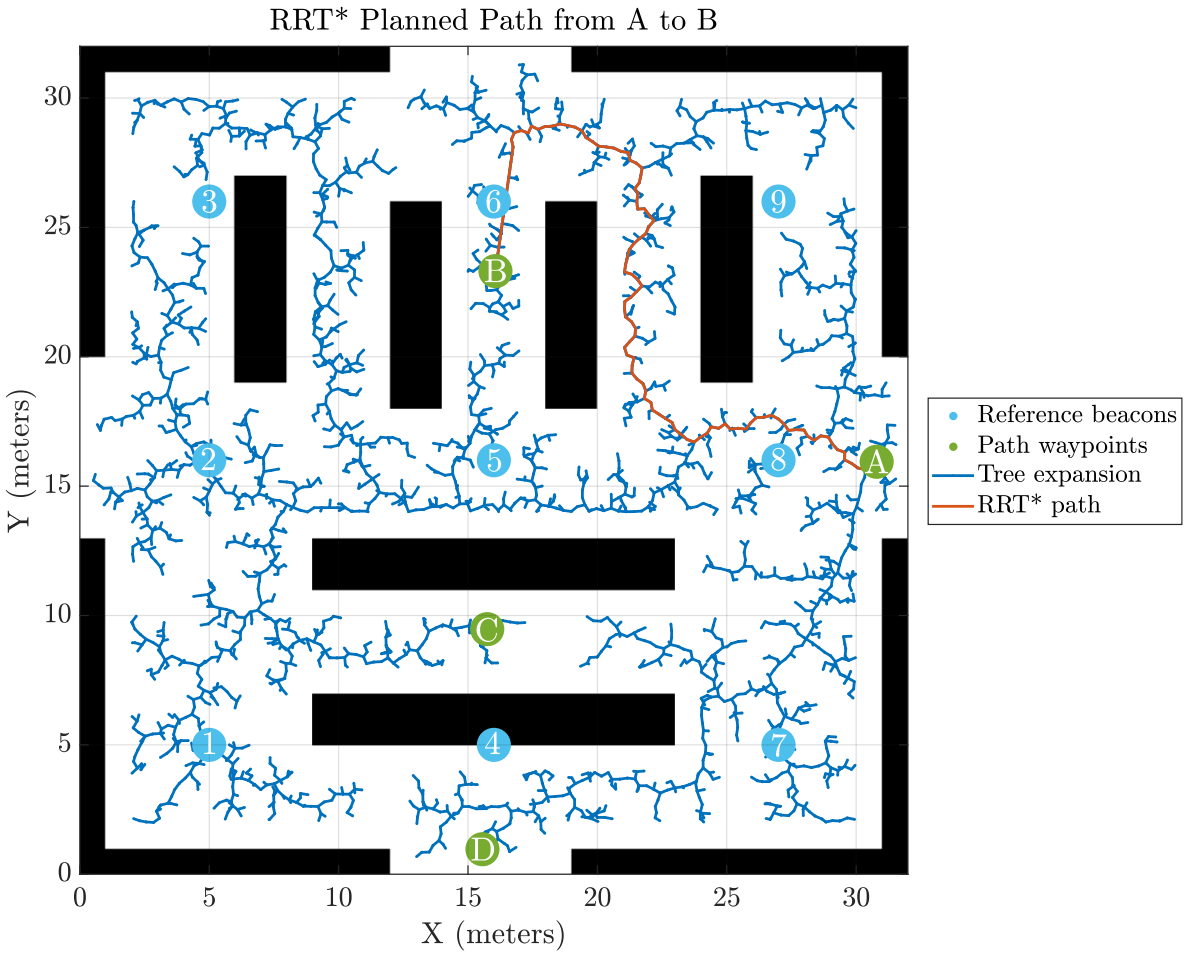

Figure 33: The planned path from waypoint A to B.

The acceleration and angular velocity are then fed into an IMU simulation model which, using that data, produces accelerometer and gyroscope measurements in the x-, y-, and z-axes, with a sampling frequency of 100 Hz. The IMU simulation model allows parameter tuning to produce realistic measurements. Some of these parameters include misalignment of axes (occurs when the sensors are mounted to the PCB), white noise, random walk (or Brownian noise), and bias instability (or pink noise). Typical values from the ADIS16448 inertial system data sheet [19] from Analog Devices were used. Figures 34-37 show the effect of introducing the parameters mentioned above to an example output sine signal of a gyroscope. The resulting accelerometer and gyroscope measurements for the path shown in Figure 32 are shown in Figures 38 and 39.

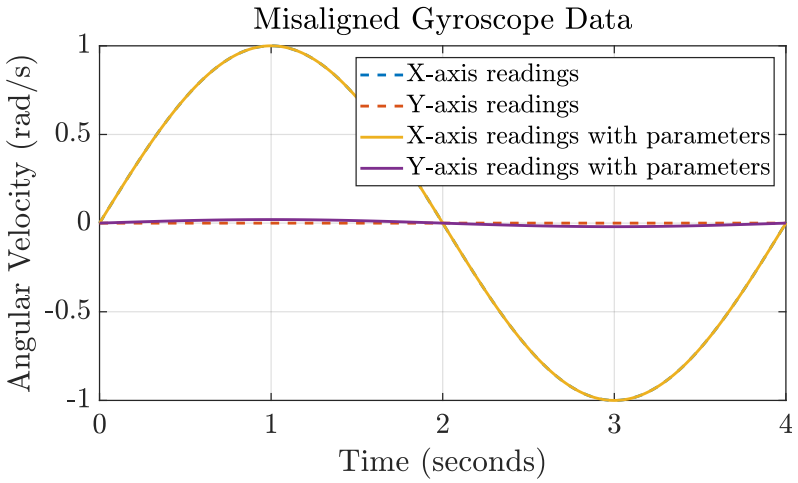

Figure 34: The effect of axes misalignment on gyroscope data.

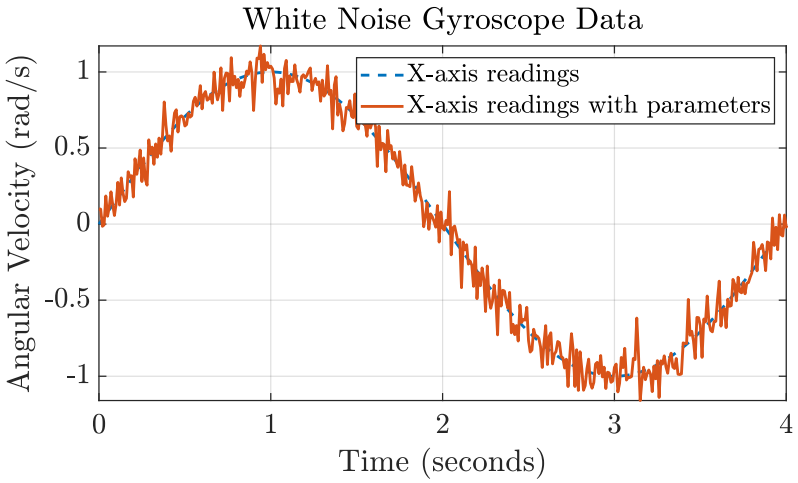

Figure 35: The effect of white noise on gyroscope data.

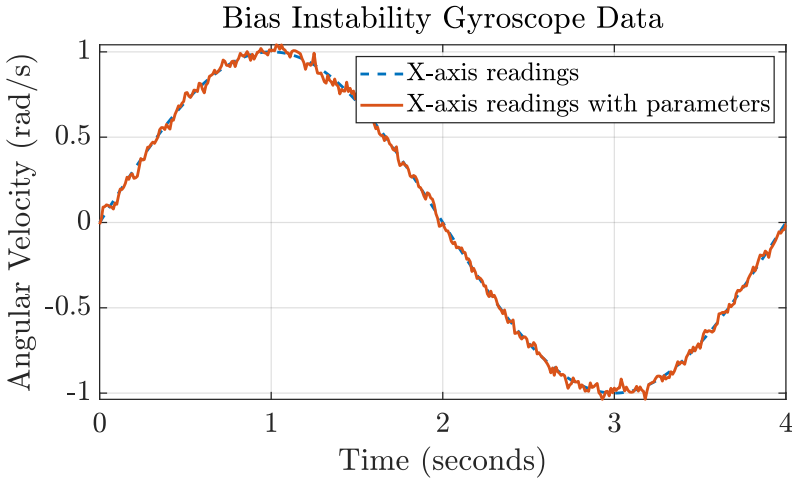

Figure 36: The effect of bias instability on gyroscope data.

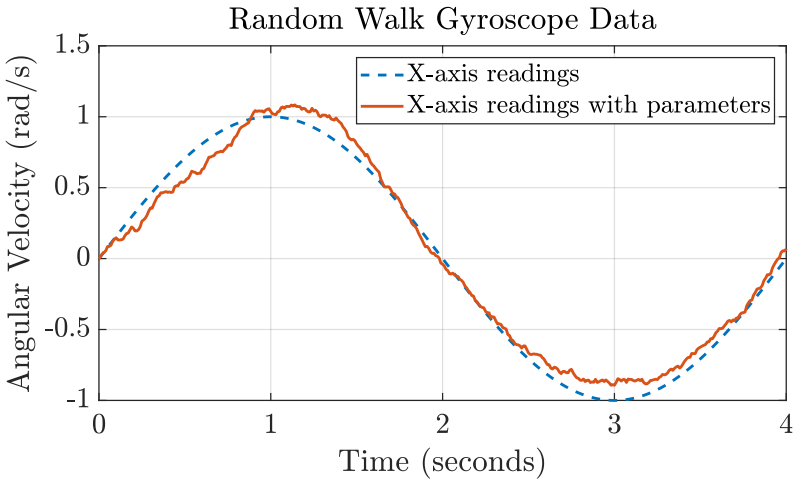

Figure 37: The effect of random walk on gyroscope data.

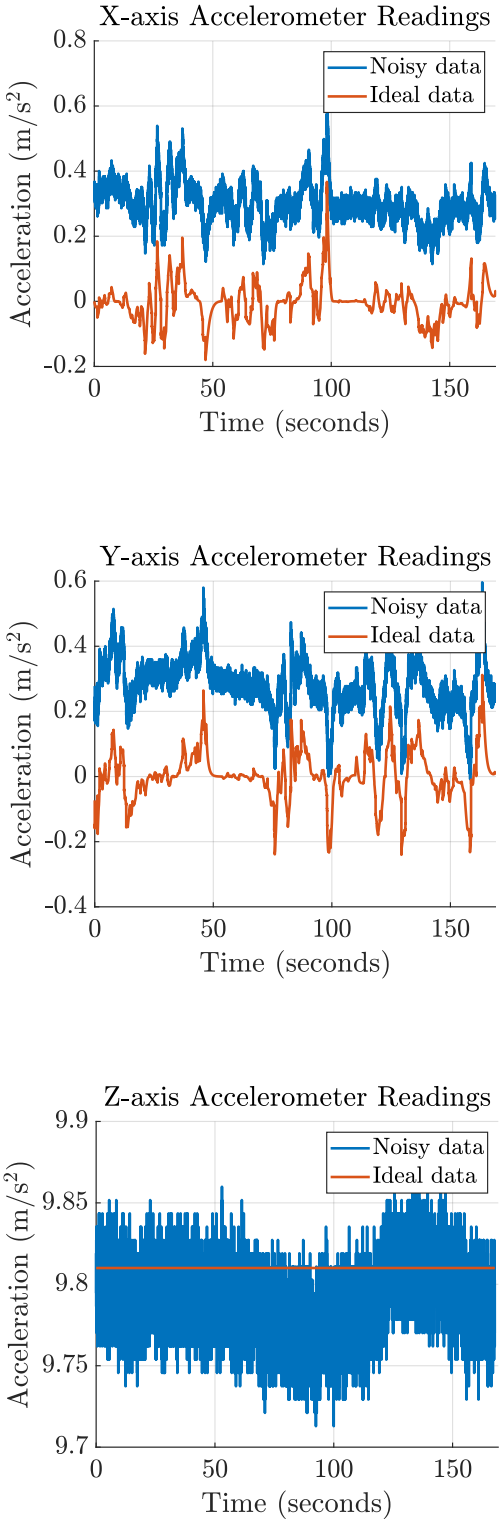

Figure 38: The ideal and noisy accelerometer readings.

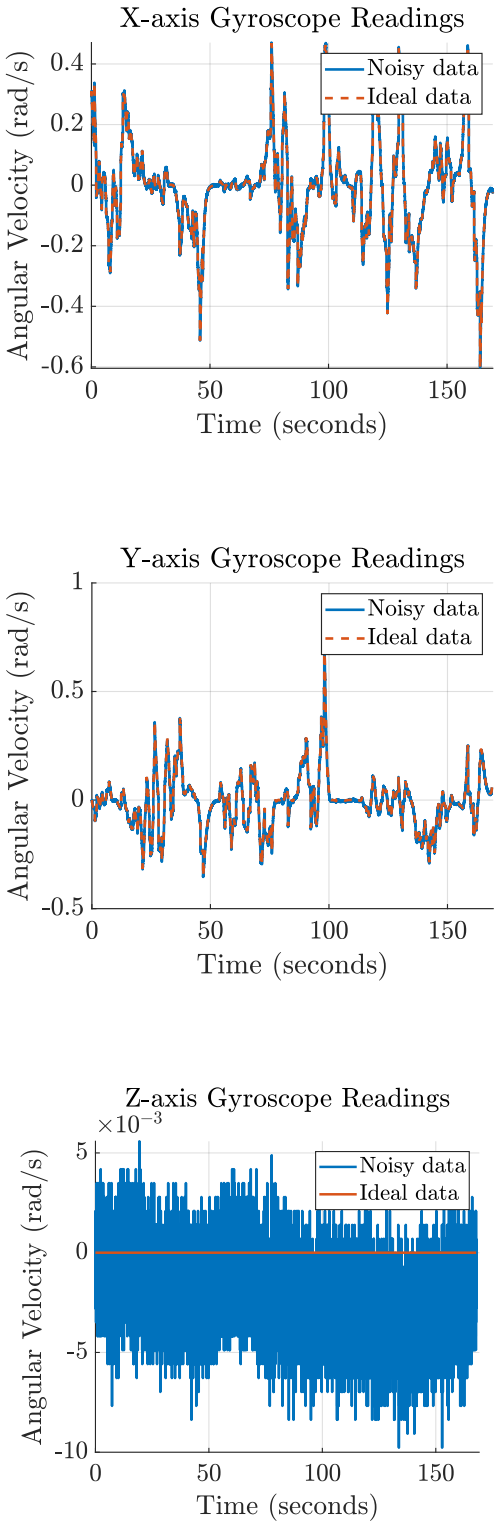

Figure 39: The ideal and noisy gyroscope readings.

A ray tracing algorithm, called the image method, was employed [18]. In essence, for a given transmitter T_x and receiver R_x locations, the image of the receiver Ri with respect to a reflection plane ρ is determined. After that, a line is drawn between R_i and T_x, intersecting ρ at a point Q. The trajectory of the reflected ray is then determined by considering the three points Tx, Q, and R_x, as shown in Figure 40. The converse can also be done by taking the image of T_x with respect to ρ and then drawing a line between T_i and R_x, yielding the same result.

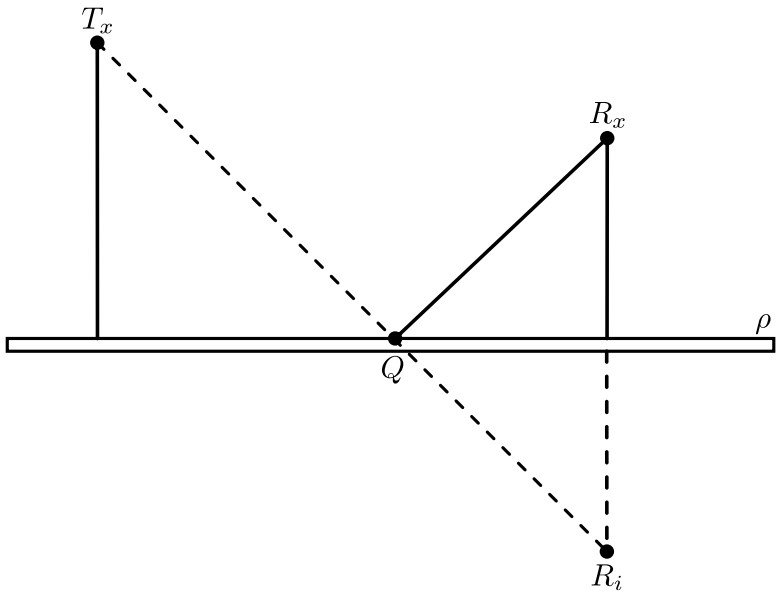

Figure 40: The image method [18].

Due to the computational overhead that is required for the algorithm to work, we decided to simplify the 3D map shown in Figure 23. The racks were approximated to be a grid of boxes, as shown in Figure 41 (the roof was removed for viewing purposes). Due to this, the differences in results between the original map and the simplified map were minimal, while the amount of time spent on simulation was substantially lowered.

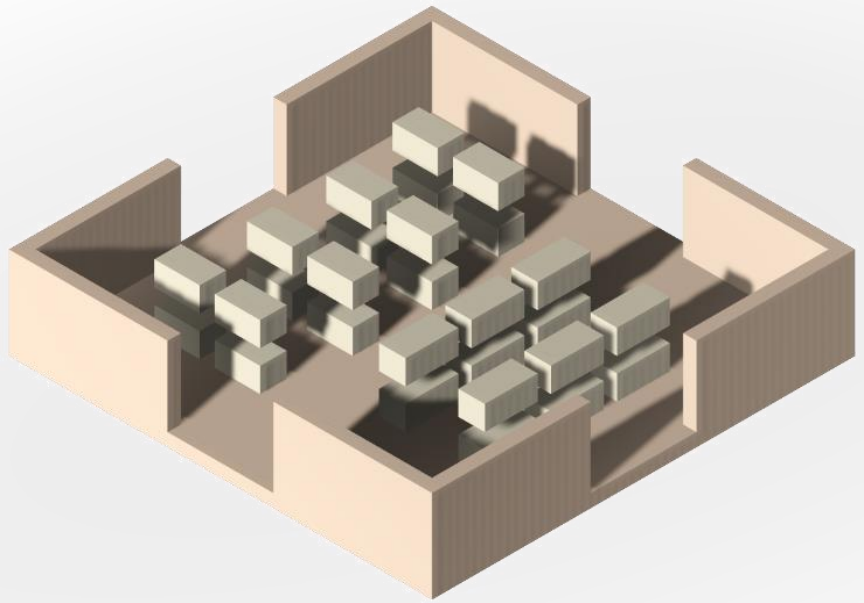

Figure 41 The simplified warehouse floor 3D map.

Furthermore, the RSSI was calculated every 0.25 meters. Figure 42 shows the RSSI map of beacon 5. After that, the RSSI values were interpolated linearly such that the results were for every 0.01 meters instead of 0.25 meters, as shown in Figure 43. Again, this saved a ton of simulation time while keeping the results very close to what they would have been if the RSSI was instead calculated every 0.01 meters. After that, the RSSI values at the points of the path the robot takes are extracted from the generated RSSI maps and saved as shown in Figures 44-46.

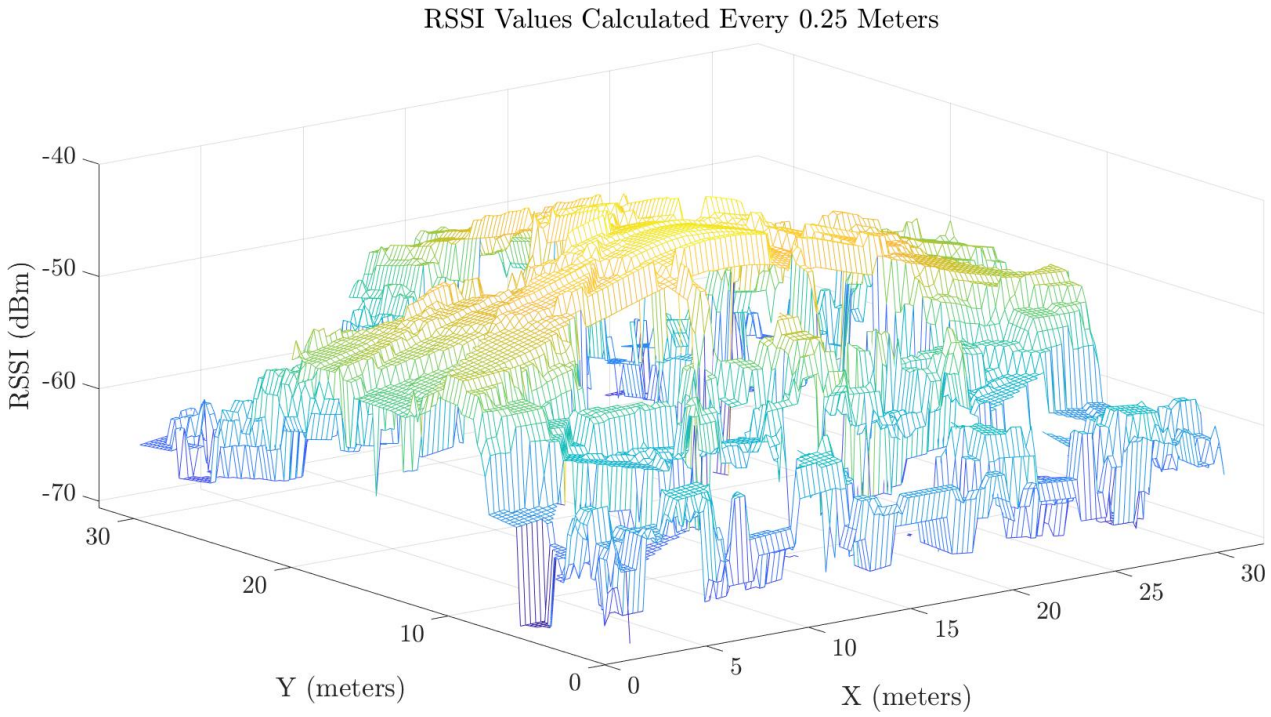

Figure 42: The calculated RSSI heatmap for beacon 5.

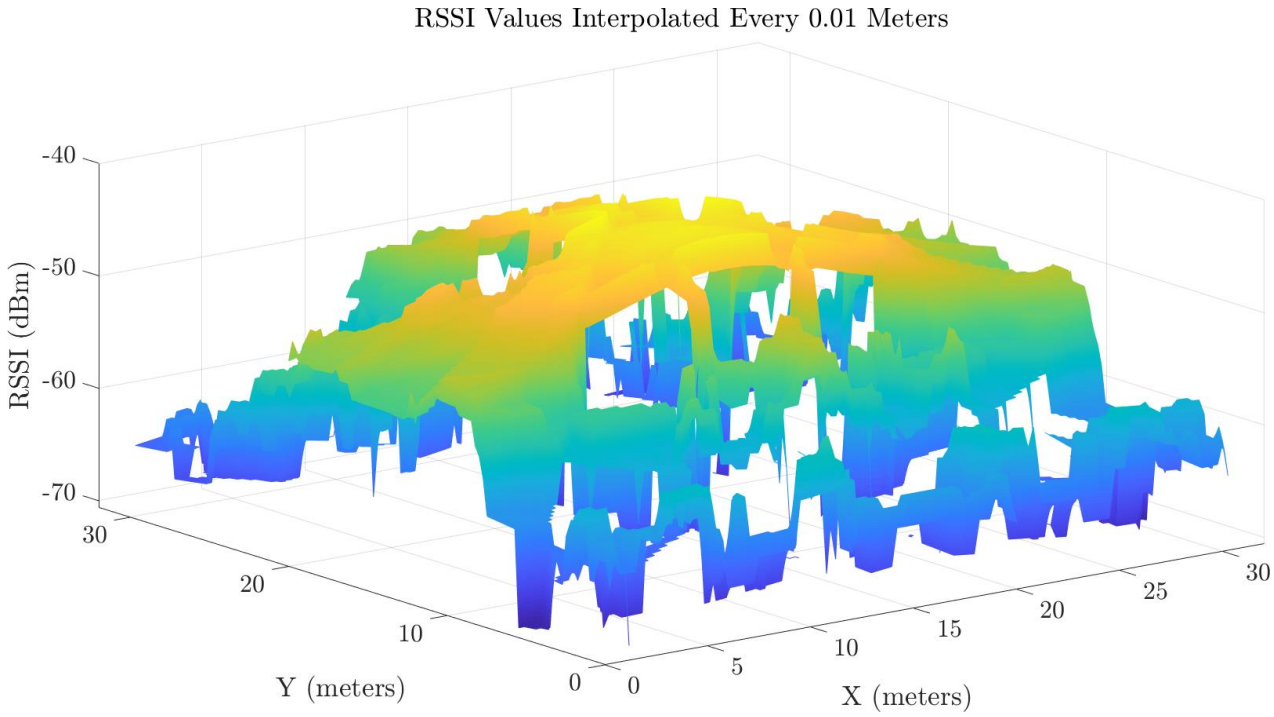

Figure 43: The interpolated RSSI heatmap for beacon 5.

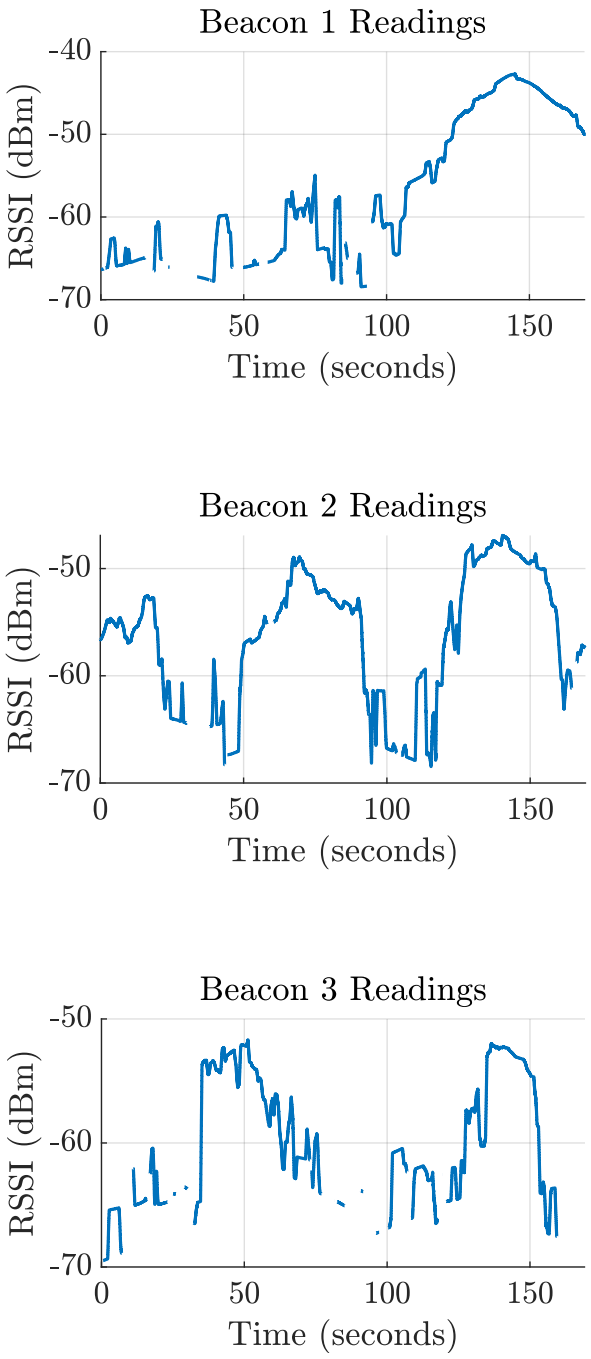

Figure 44: RSSI values for beacons 1-3.

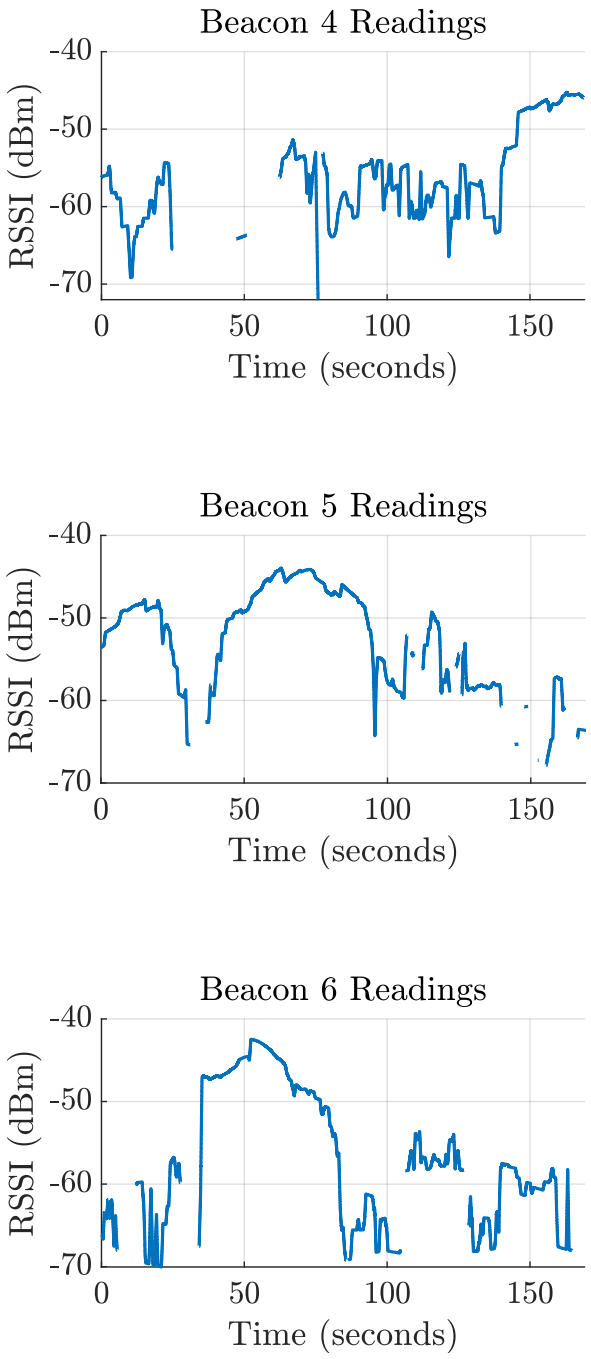

Figure 45: RSSI values for beacons 4-6.

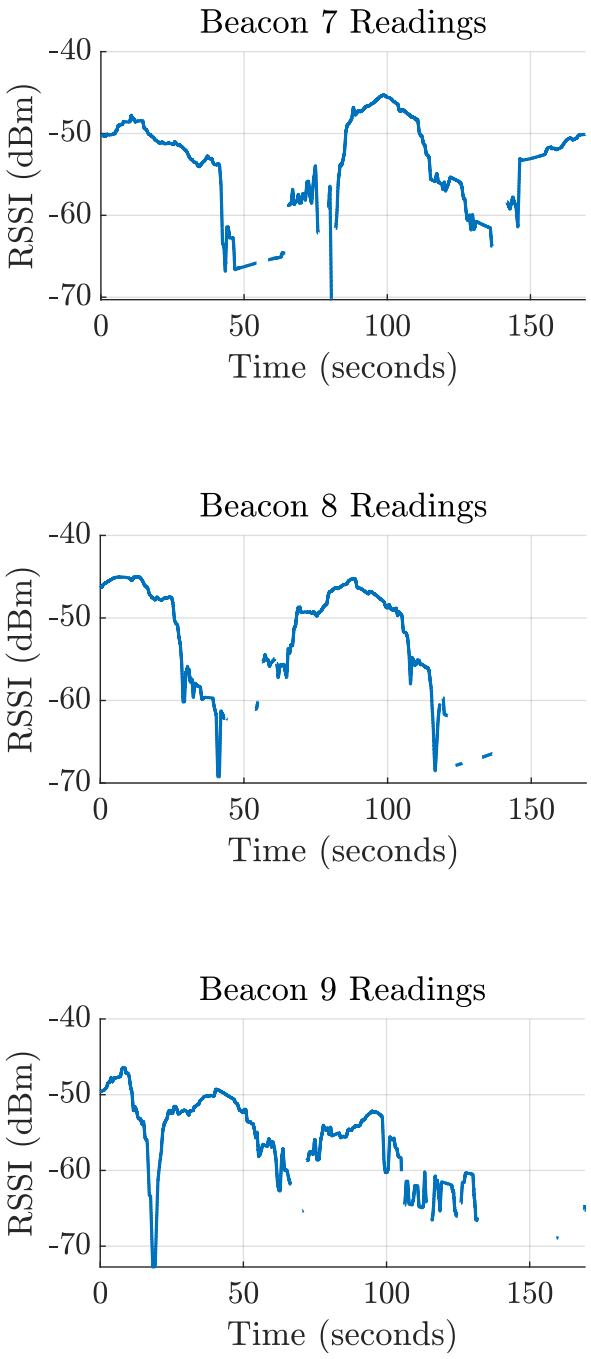

Figure 46: RSSI values for beacons 7-9.

Concretely, the process of generating a path and its data is shown as a flowchart in Figure 47. This process was repeated 1,200 times to generate the data that was used in 4.2 and 4.3. The resulting array contains: the number of iterations, the number of beacons and their locations, the sampling time, and for each path its waypoints, the beacon measurements, the

accelerometer and gyroscope readings in the x-, y-, and z-axes, and the paths themselves. Since the paths are of different lengths and hence the measurements are as well, zero padding was applied and the number of samples were recorded in the array.

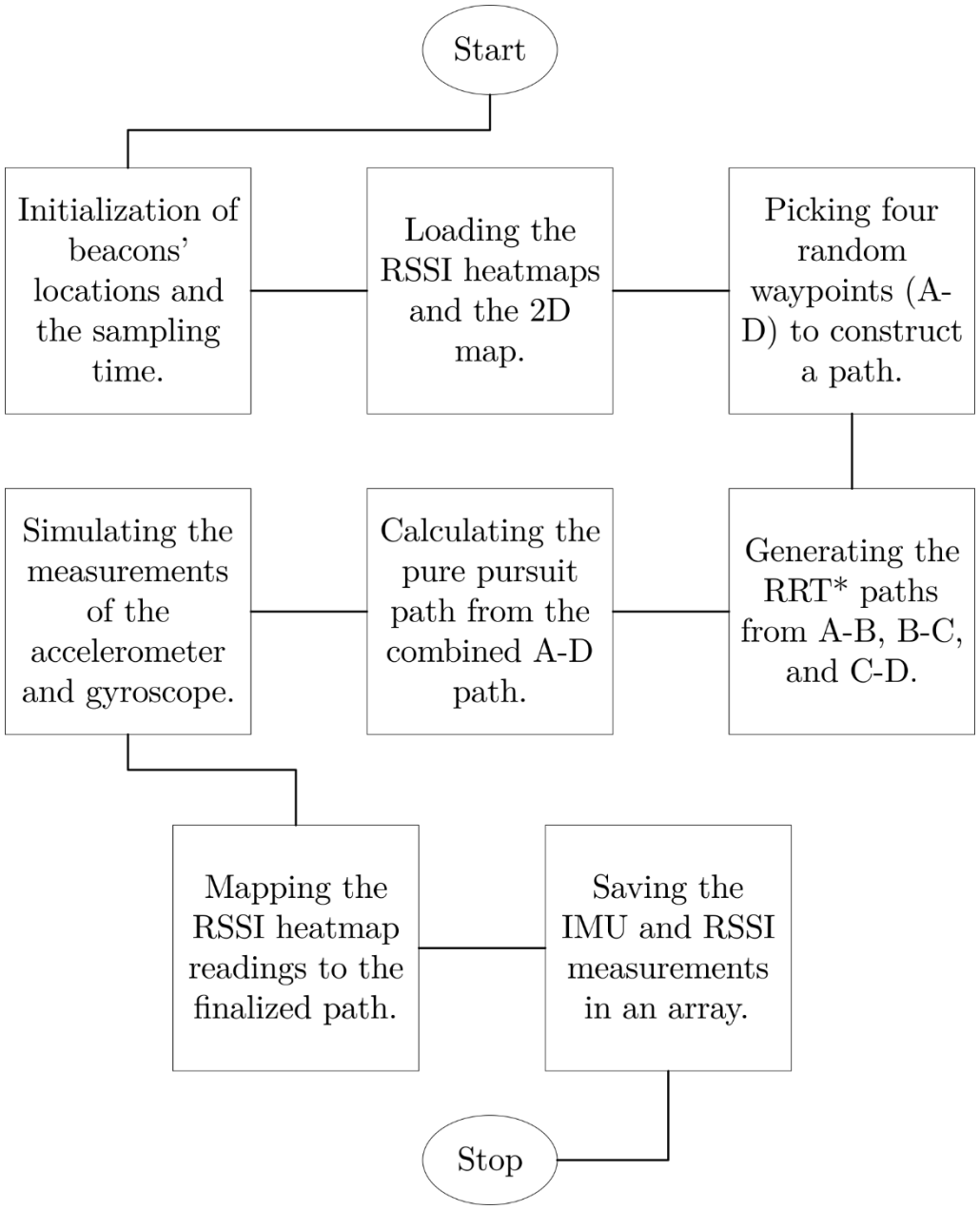

Figure 47: Flowchart of the process of generating a path and its data.

4.2 *Building DL Model*

The DL model was made of two networks. One was an Artificial Neural Network (ANN) and the other was an LSTM network. The idea was to output an estimated position from the ANN using only the RSSI measurements as a feature, and then using that output as a feature for the LSTM network alongside the IMU measurements to correct those predictions. Any attempts to create a model that features the IMU measurements and the RSSI measurements together did not pan out well. The structure of the ANN is shown in Figure 49. Each hidden layer used the ReLU function as the activation function. This means that the neurons are activated only when their weights are positive. A dropout probability of σ, which meant that neurons in the hidden layer were turned off at random with a probability of σ. On the other hand, the architecture of the LSTM network is shown in Figure 50. There were three LSTM layers, each containing 100 hidden units. Both models cannot deal with $-\infty$ RSSI readings. To fix this, a new log scale was suggested as shown in Figure 48. The majority of the RSSI readings lie within the range of -80 to 0, meaning the validity of the readings stands. However, when the RSSI reading is $-\infty$ (i.e., the radiated beacon signal is completely blocked in that area and no rays propagate there), the reading in dBm in the new scale will saturate to -120. More on this in 4.3.

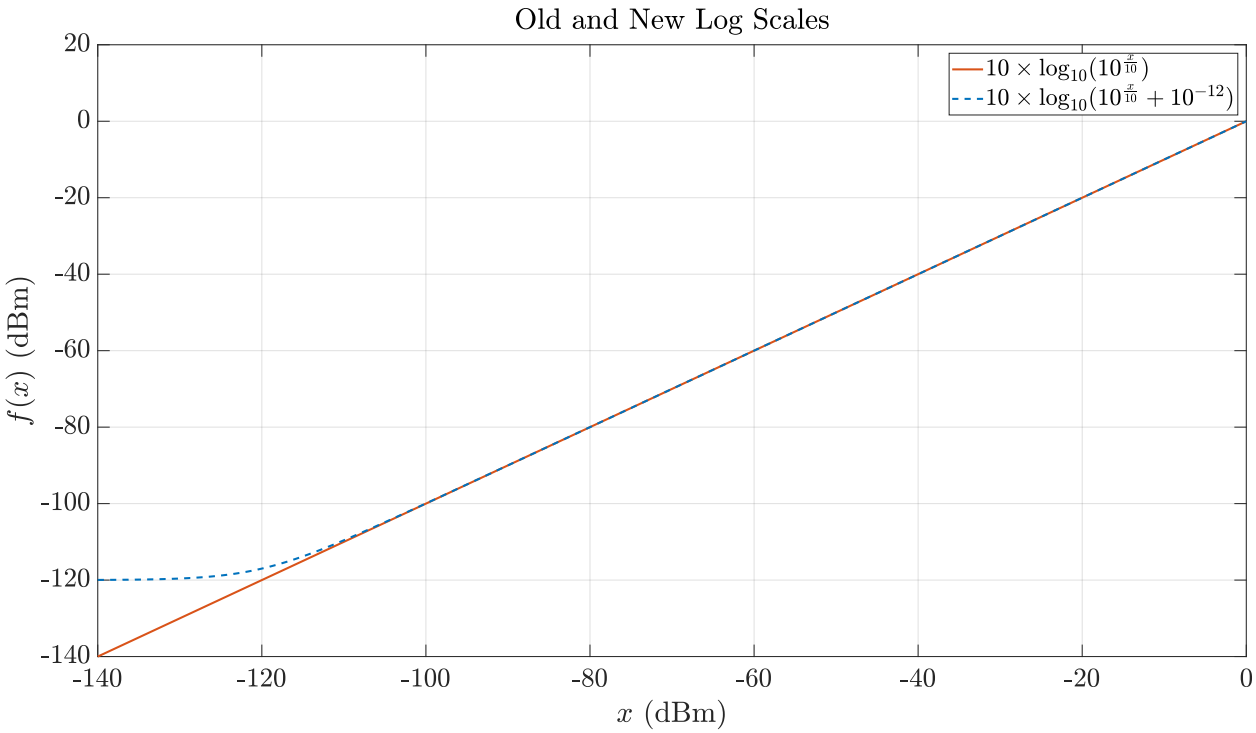

Figure 48: Old and new log scales.

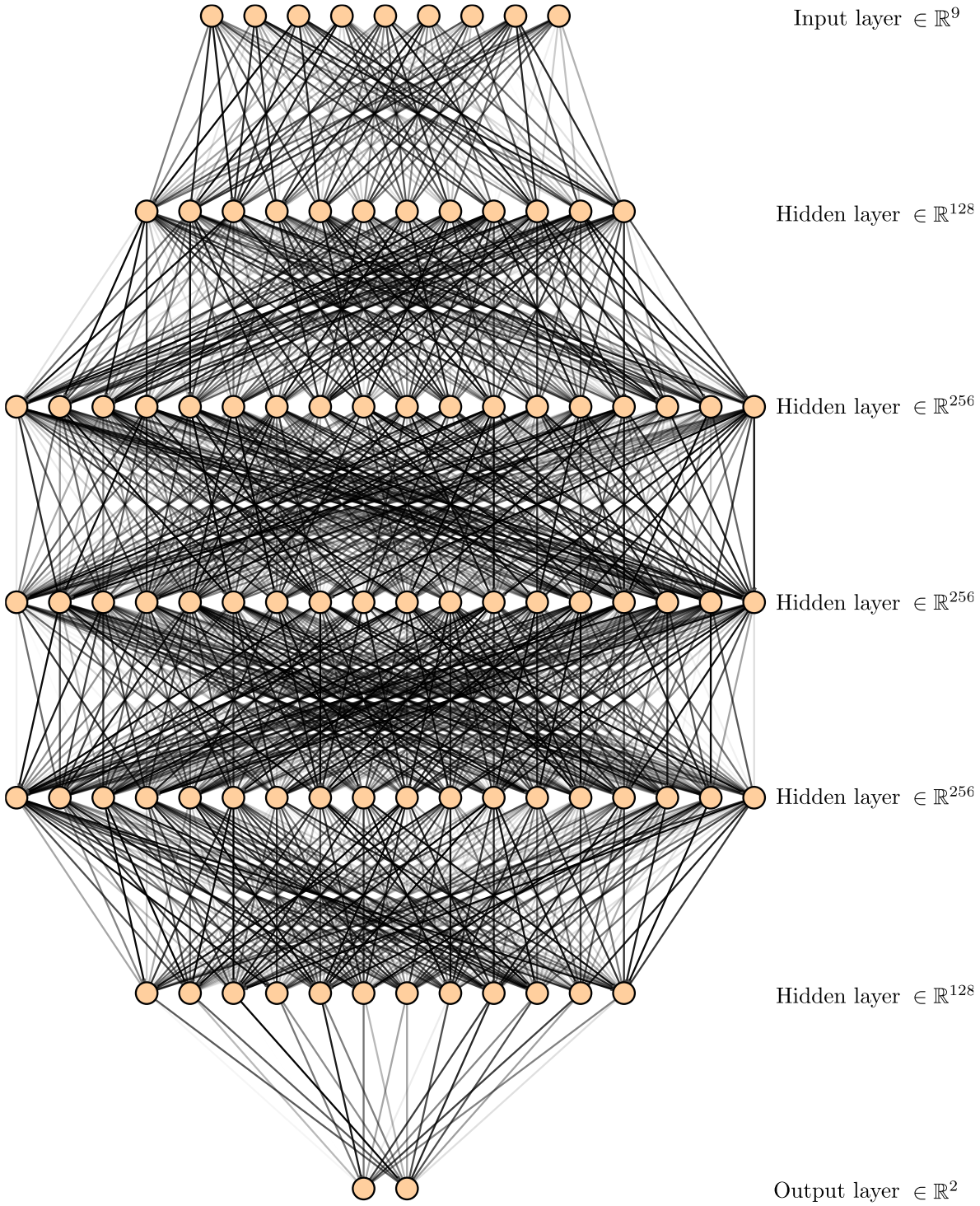

Figure 49: The architecture of the ANN.

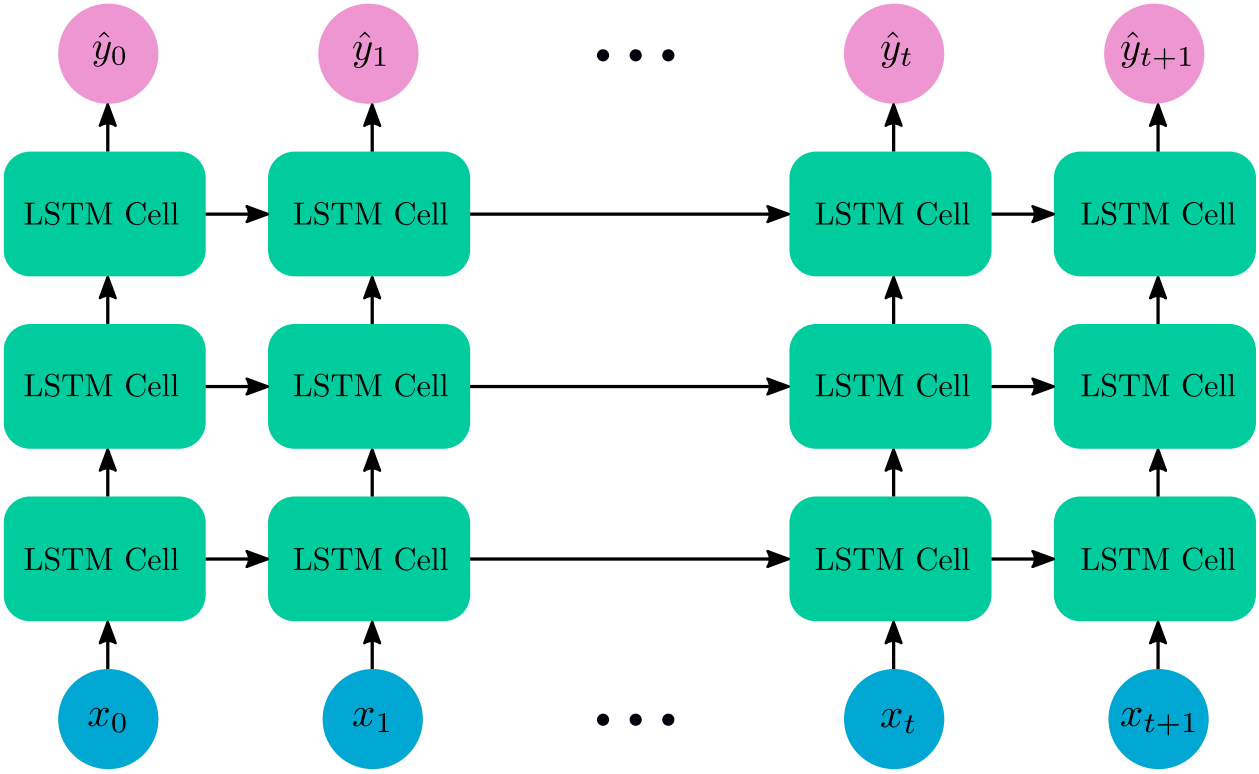

Figure 50: The architecture of the LSTM network.

80% of the training data (800 examples) was used for training, 20% (200 examples) was used for validation, and the rest (200 examples) was set aside for testing which is considered a rule of thumb among ML professionals. The ratio comes from the Pareto principle (or the 80/20 rule), which states that for 80% of the consequences come from 20% of the causes for many outcomes. This was observed in a work by Italian economist Vilfredo Pareto which stated that 80% of the land in Italy was owned by 20% of the population [42]. Statisticians later noticed that this statistically explained a lot of phenomena.

The features were also scaled prior to training the networks. Specifically, standardization (or whitening) was applied, which is done by subtracting the mean of the data μ off of each feature and dividing by the standard deviation of the data s, expressed as

$$x_{new} = \frac{x_{old} - \mu}{s}. \tag{14}$$

This would transform the data to have the properties of a standard normal distribution with zero mean and a variance of 1, making it unitless as well. The idea behind standardization

is to prevent the DL model from making assumptions that features with a higher range should significantly influence the outputs than features with a lower range. For example, if we were to have two features for classifying fruits based on weight and price, a fruit may weigh 10 grams and cost $10. It is obvious that these two numbers represent completely different things for humans, but for a DL model, they would be treated the same. Additionally, algorithms like GD work much faster when all features are scaled. This mathematically translates into algorithms oscillating less back and forth, making them take a short time before finding their way to the minimum point [43].

After that, the data representation was changed such that it was suitable for both networks. A cell array was created where each entry represented a path and had an array that contained the x- and y-accelerometer measurements, the x- and y-gyroscope measurements, and the measurements of the nine beacons.

The choice of the DLOA was narrowed down between Adam, RMSprop, and SGDM, each containing hyperparameters that needed to be tuned. Traditionally, methods such as grid search and RS were used. Grid search, or parameter sweep, is a method in which a set of values for each hyperparameter are exhaustively searched to find ones that produce the best results according to some performance metric. RS on the other hand replaces the exhaustive search with randomly picking values within the set of values from a specified statistical distribution. Because of this, it usually computationally outperforms grid search, but it should be noted that sometimes RS misses important combinations in the hyperparameters space. The search space grows exponentially in both methods as the number of hyperparameters to tune increases, meaning tuning becomes a time-consuming step. Additionally, both do not consider past results when iterating over combinations, meaning they are uninformed by past evaluations. [44]

An informed way of hyperparameter tuning which does take previous results into account when choosing the next combination of hyperparameters to evaluate is Bayesian optimization, which is capable of strictly searching for combinations of hyperparameters that seem promising, rather than waste time on ones that are not, causing it to reach to the optimal combination of hyperparameters in less iterations than the previous two methods. In Bayesian optimization, past evaluations are used to form a probabilistic model that maps hyperparameters to a probability of a score on the objective function. This is what is called the surrogate of the objective function. Considering the objective function in the case of a DL

model is very complex, rather than finding the hyperparameters that work best on it, hyperparameters that work best on the surrogate function would be investigated and then applied on it the objective function. Afterwards, the surrogate model would be updated to incorporate the new results and the process is then repeated until an optimal solution is obtained. This means that more time is spent on choosing the next hyperparameters to try out but as a result less time is spent evaluating changes of hyperparameters on the objective function. Since the time spent on choosing the next hyperparameters is much less than the time spent in the objective function, the Bayesian optimization approach is much quicker than grid search and RS [45].

We first had to tune the hyperparameters of the ANN with the RSSI readings as its features when using each of the three previously mentioned DLOAs. Starting with the SGDM DLOA, the RMSprop DLOA, and the Adam DLOA. The range of the hyperparameters, the scale used, and when they were applicable is shown in Table 3. Note that the ranges chosen are based on rules of thumb by practitioners. The RMSE for the trained ANN using hyperparameters obtained from the Bayesian optimization for the three DLOAs are shown in Figures 51-53. The logarithmic scale was used to search a bigger hyperparameter space quickly. Note that 30 iterations were done, but for each DLOA, only three results were chosen to be displayed (the best RMSE, the worst RMSE, and one that is in-between). A comparison between the best hyperparameters for the three DLOAs is shown in Figure 54. The same was done for the LSTM network with the IMU readings and the output of the ANN as its features, and the results are depicted in Figures 55-57, while the comparison between the best hyperparameters is shown in Figure 58.

Table 3: The range, scale, and applicability of the hyperparameters.

Hyperparameter	Range	Scale	Applicable
Dropout Probability (σ)	0.3 – 0.5	Linear	All three
Regularization Parameter (λ)	10^{-10} – 10^{-2}	Logarithmic	All three
Initial Learning Rate (α)	10^{-3} – 10^{-2}	Logarithmic	All three
Momentum Parameter (γ)	0.5 – 0.95	Logarithmic	SGDM only

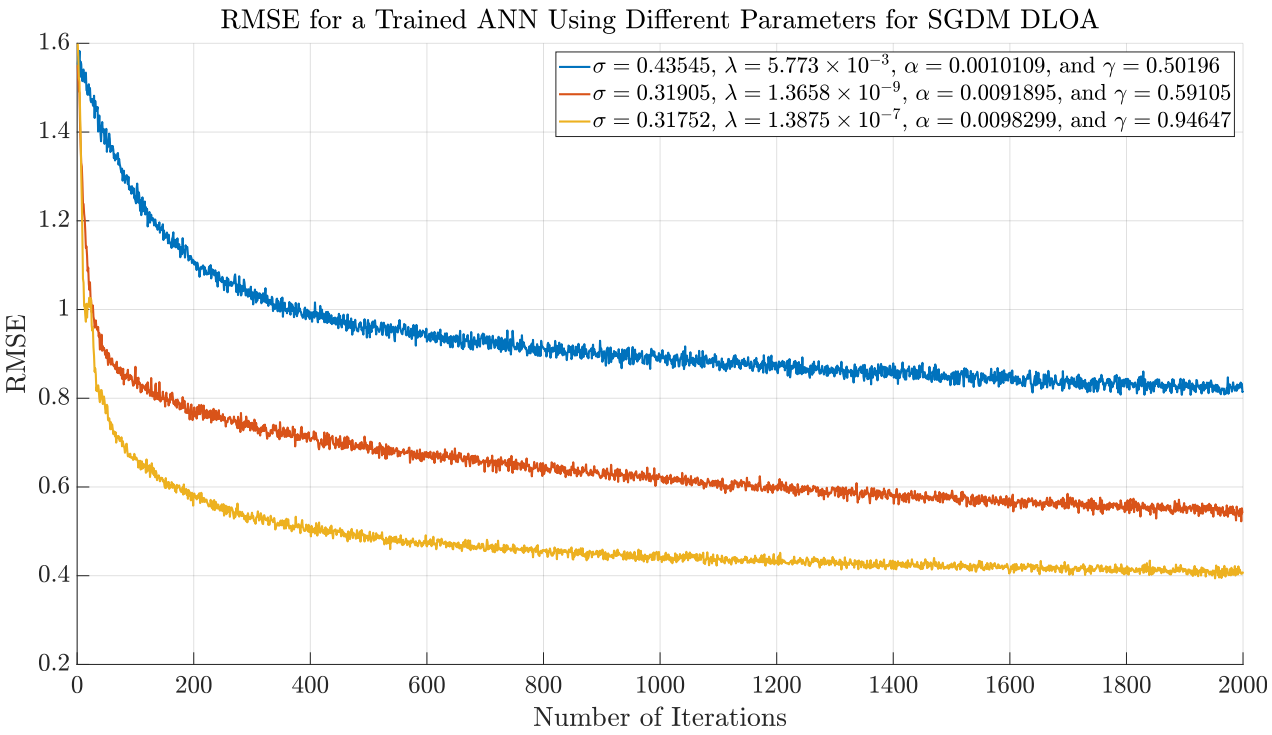

Figure 51: The resulting RMSE when training the ANN using different parameters for the SGDM DLOA.

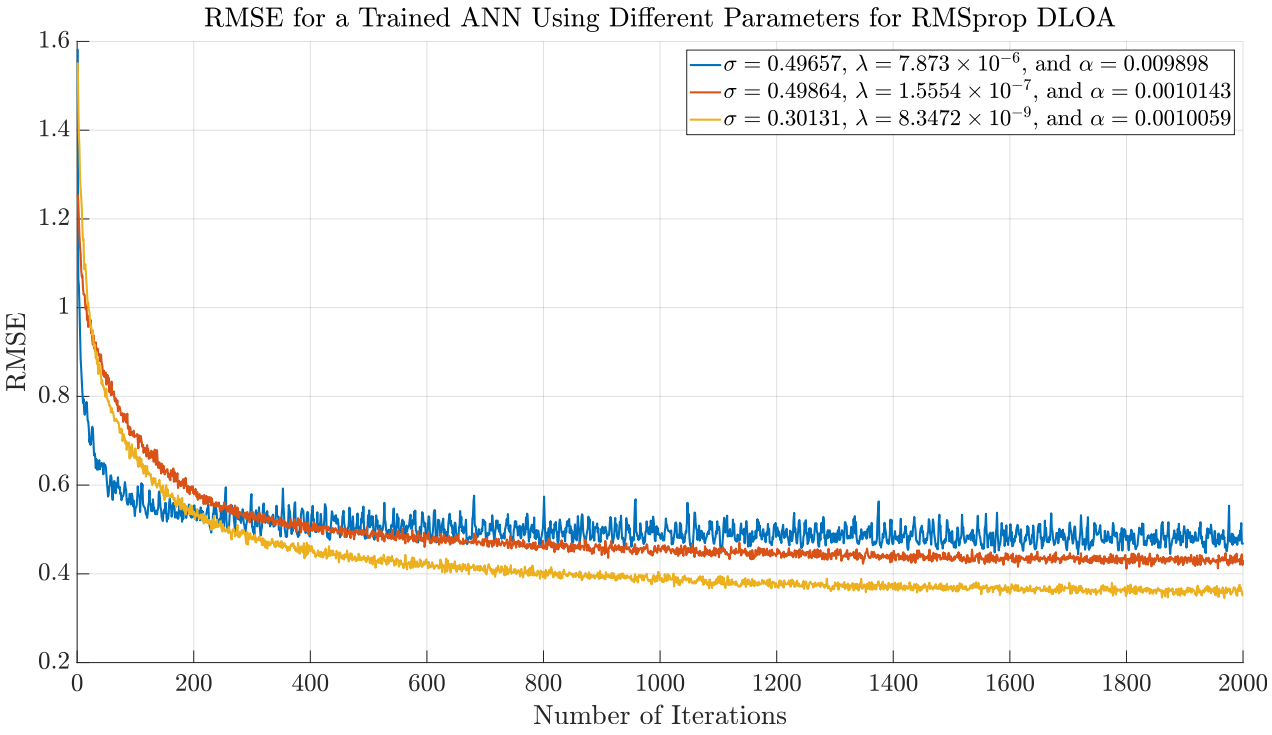

Figure 52: The resulting RMSE when training the ANN using different parameters for the RMSprop DLOA.

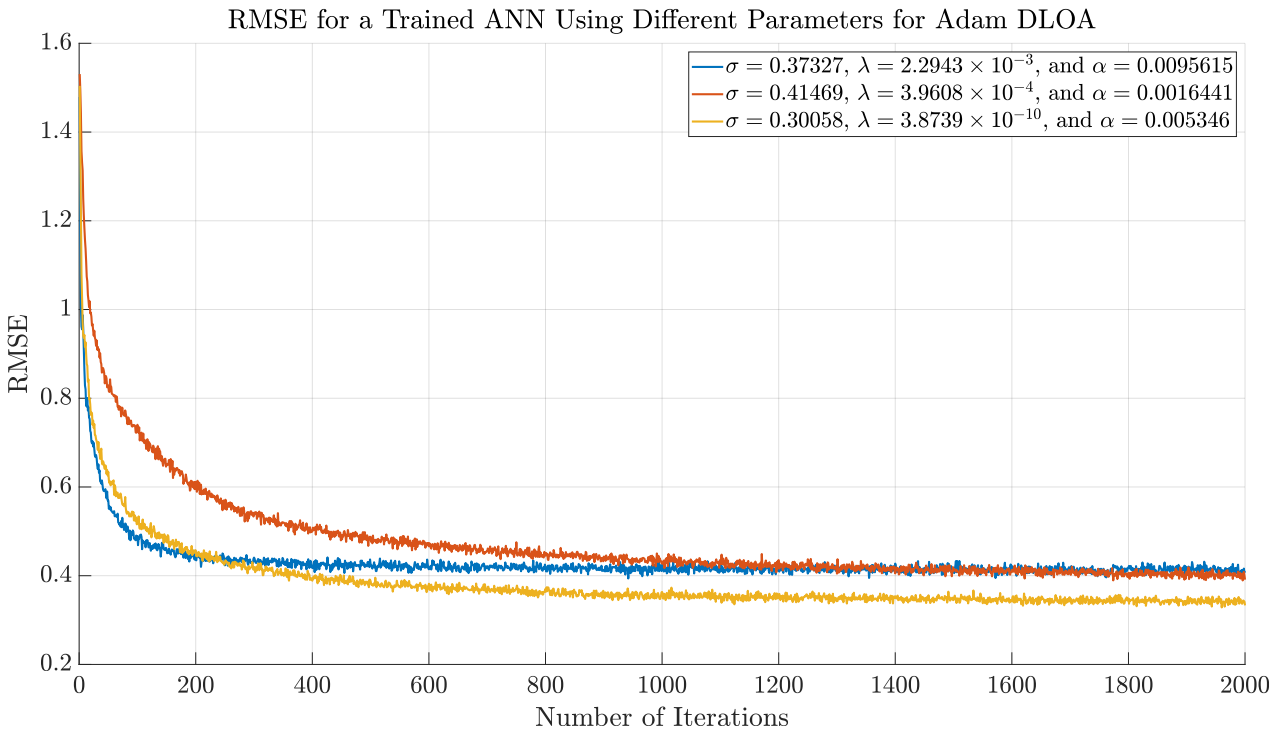

Figure 53: The resulting RMSE when training the ANN using different parameters for the Adam DLOA.

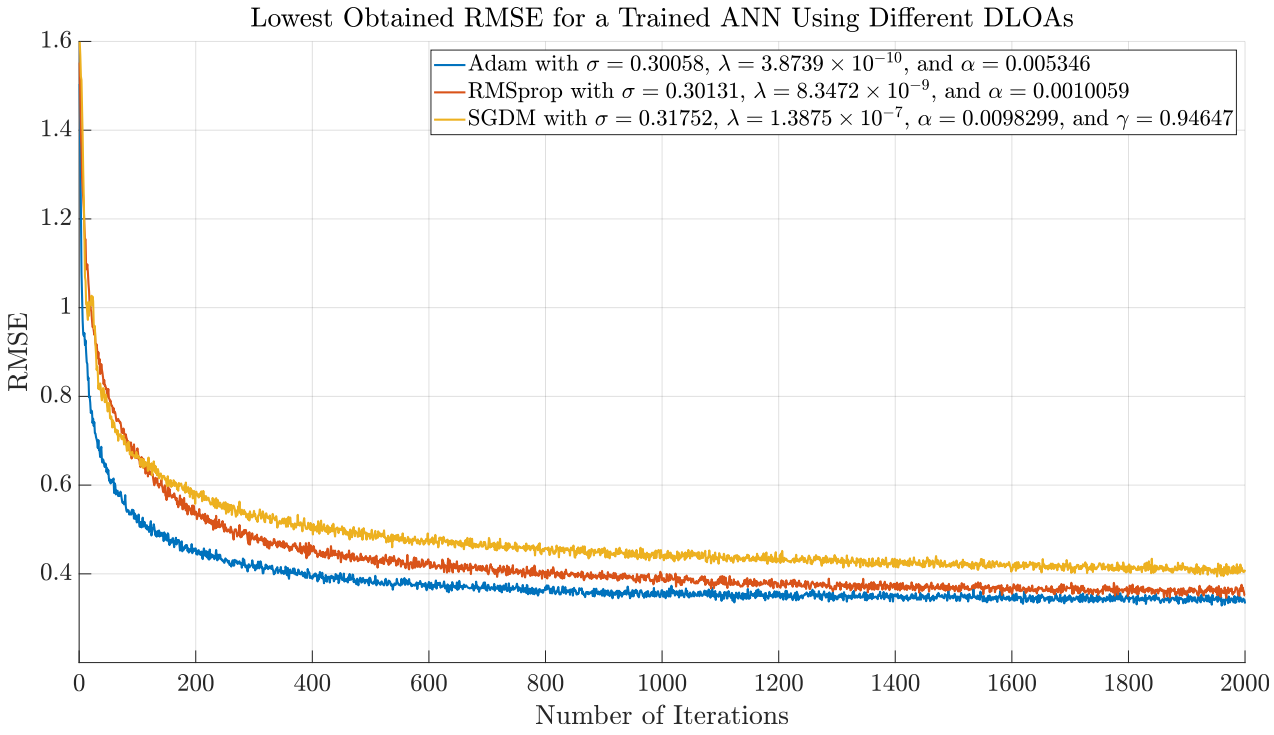

Figure 54: The best RMSE obtained from training the ANN using the three DLOA.

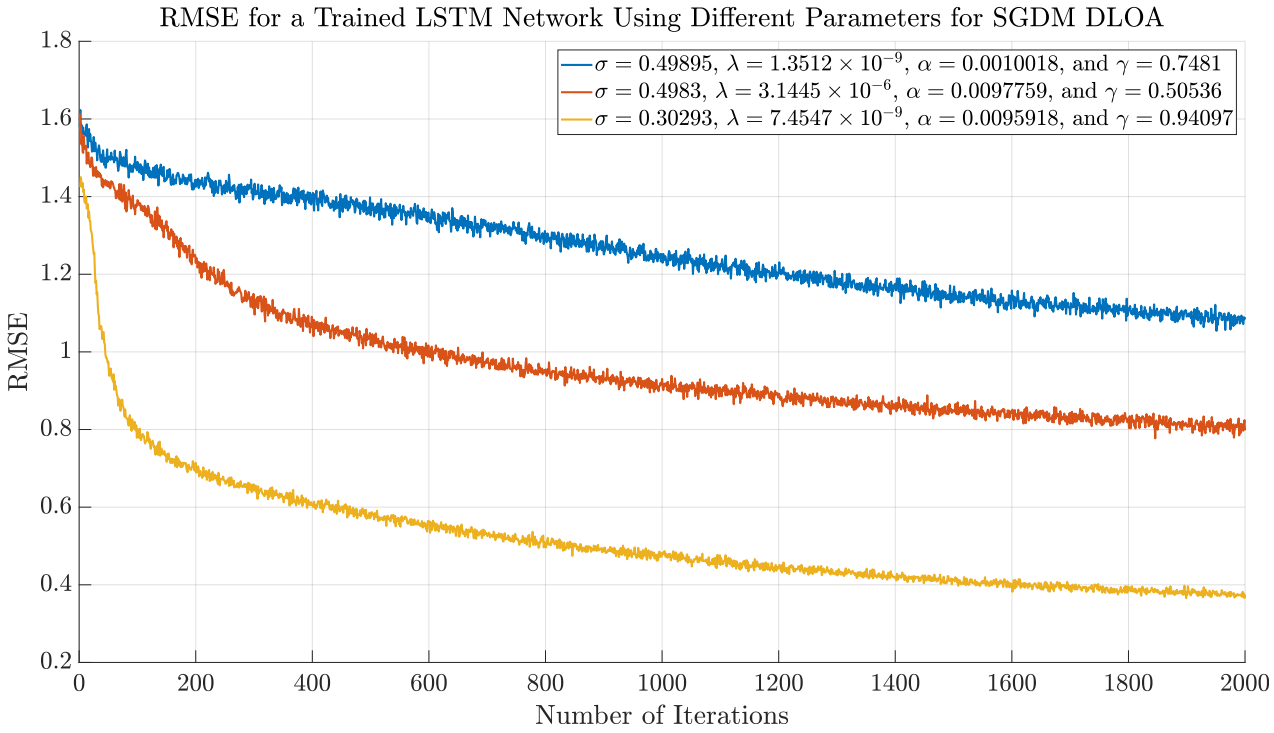

Figure 55: The resulting RMSE when training the LSTM network using different parameters for the SGDM DLOA.

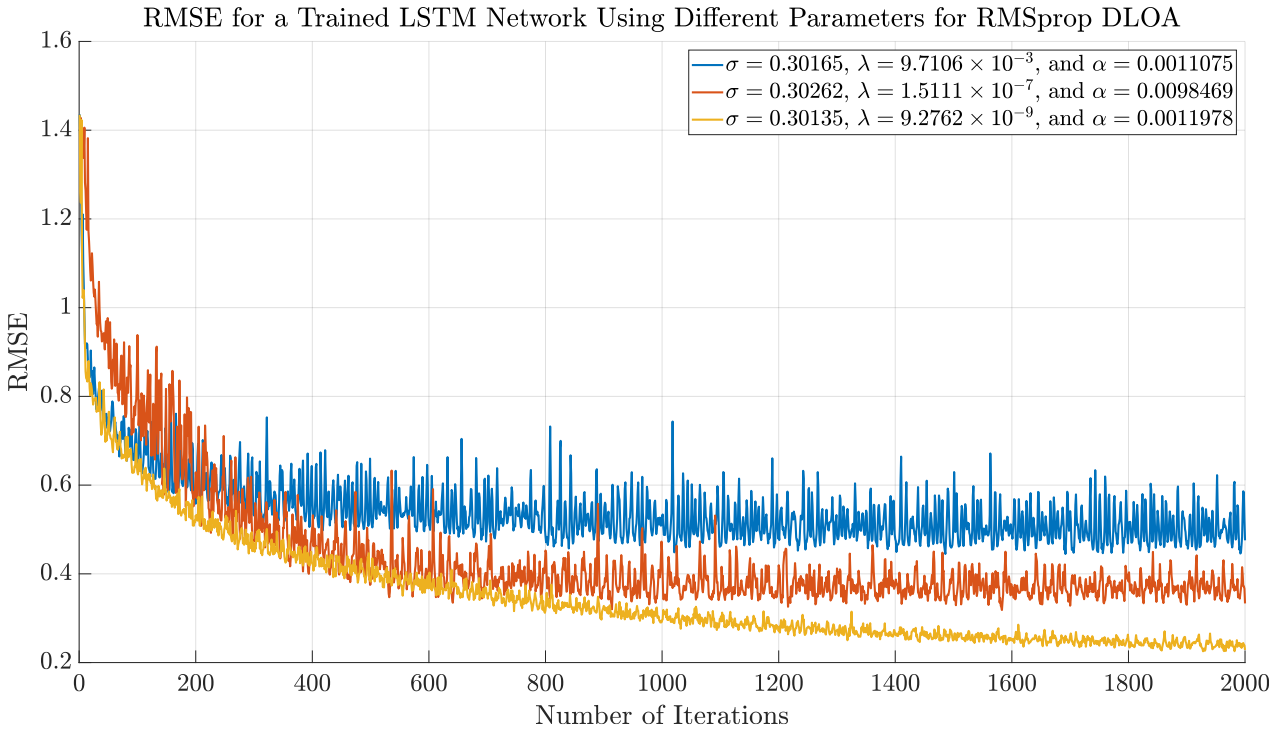

Figure 56: The resulting RMSE when training the LSTM network using different parameters for the RMSprop DLOA.

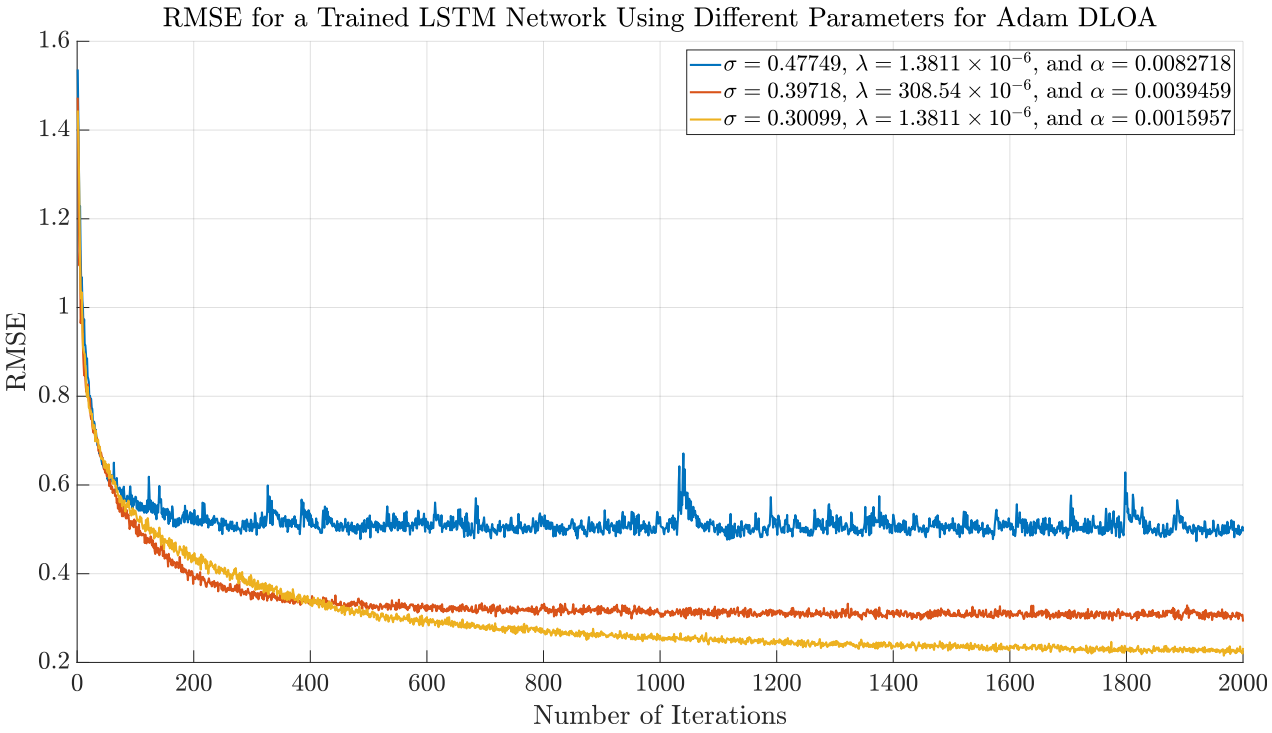

Figure 57: The resulting RMSE when training the LSTM network using different parameters for the Adam DLOA.

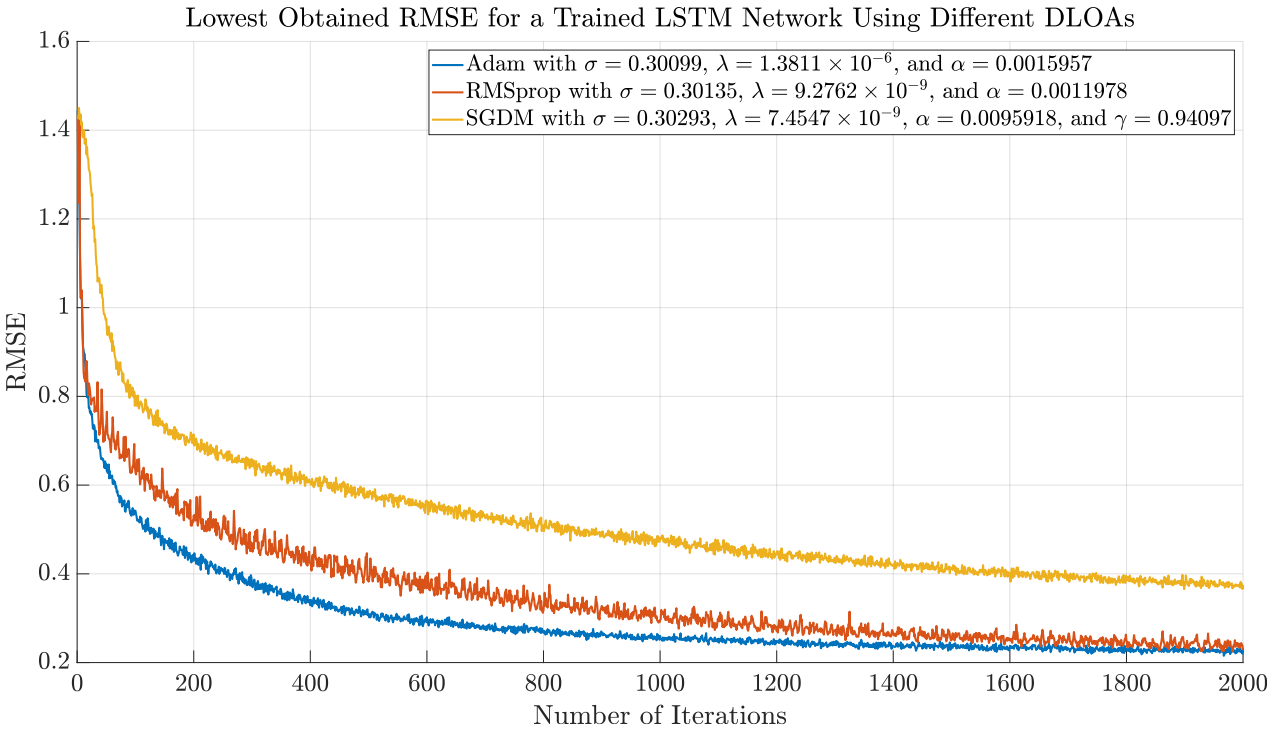

Figure 58: The best RMSE obtained from training the LSTM network using the three DLOAs.

4.3 *Optimizing Beacon's Locations*

The goal of this part was to design an algorithm that is capable of determining the optimal locations of a certain number of beacons N to increase the average accuracy of the DL model in 4.2 without having to increase the number of those beacons, thereby saving costs that would result from having to buy more beacons. This is done for an indoor environment with a given Probability Density Function (PDF) of the robot's position in that environment (i.e., the probability of the robot being at a certain position).

The expected accuracy of a certain placement of the beacons $\mathcal{U}$ can be expressed as

$$\mathcal{U}(\boldsymbol{\alpha}^*, \boldsymbol{\beta}^*) = \iint_{\mathbb{R}} f_{xy}(x, y)\, U^{xy}(\boldsymbol{\alpha}^*, \boldsymbol{\beta}^*)\, dxdy, \tag{15}$$

where $f_{xy}(x, y)$ is the PDF of the robot's location, $\boldsymbol{\alpha}^* = [\alpha_1^*, \alpha_2^*, \dots, \alpha_N^*]$ where α_n^* is the x-coordinate of the n^{th} beacon, $\boldsymbol{\beta}^* = [\beta_1^*, \beta_2^*, \dots, \beta_N^*]$ where β_n^* is the y-coordinate of the n^{th} beacon, $U^{xy}(\boldsymbol{\alpha}^*, \boldsymbol{\beta}^*)$ is the achieved accuracy by the DL model at the point (x, y), and $\mathbb{R}$ is the region of consideration (i.e., the environment of where the robot is presented).

Notice that $\mathcal{U}$ considers the robot's location density function. Thus, optimizing it will result in a higher accuracy in areas where the robot moves frequently, which increases the average accuracy of the DL model. To make the problem more tractable, $f_{xy}(x, y)$ was discretized by replacing it with the Probability Mass Function (PMS) of the robot's location $g_{xy}(x, y)$. The PMS is defined only for multiples of 0.25 meters (i.e., 0, 0.25, 0.5, etc.). $\mathcal{U}$ can now be expressed as

$$\mathcal{U}(\boldsymbol{\alpha}^*, \boldsymbol{\beta}^*) = \sum_x \sum_y g_{xy}(x, y)\, U^{xy}, \tag{16}$$

where x and y are defined over 0.25-meter multiples of the region of interest. In our case, x and y range from 0 to 32 meters each with a step size of 0.25 meters. The PMS of the environment is shown in Figure 59. This was generated while considering all the simulated paths. As can be seen, the PMS reaches its maximum in regions of interest and areas connecting them like the main aisle and the upper three aisles.

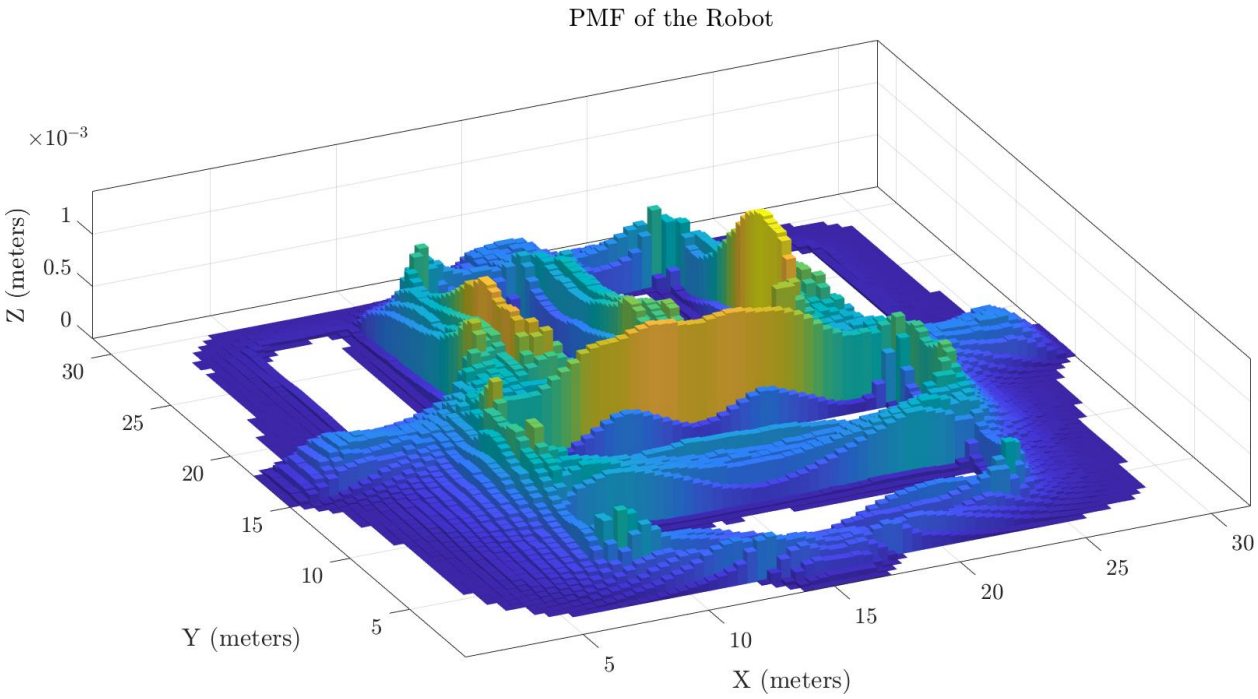

Figure 59: The PMF of the robot's location. PMF of the Robot

Since there is no obvious relationship between the average accuracy and the locations of beacons, some a prioris were made based on intuition. The first being "increasing the number of received signals will increase the average accuracy". For instance, at a certain point, a higher accuracy would be achieved from receiving four signals from four different beacons, than in receiving just two signals from two different beacons. This is true since the measurements from all beacons contain an amount of error and, assuming these errors are not correlated, having data from multiple sensors will result in less erroneous predictions.

The second a priori was "increasing the space diversity of the locations of beacons will increase the average accuracy". For instance, if two beacons were to at the same location or even very close to each other, the received signals from these two would be highly correlated, making predictions based on these two signals just as good as ones made from one signal from one beacon at that location. Thus, it is required to increase the distance between every two beacons as much as possible.

When taking these two a prioris into account, $\mathcal{U}^{xy}$ could be expressed as

$$\mathcal{U}^{xy}(\boldsymbol{\alpha}^*, \boldsymbol{\beta}^*) = \sum_{n=1}^{N} P(x, y, \alpha_n^*, \beta_n^*) + \sum_{n=1}^{N} \sum_{k=n+1}^{N} d(\alpha_n^*, \beta_n^*, \alpha_k^*, \beta_k^*), \tag{17}$$

where $P(x, y, \alpha_n^*, \beta_n^*)$ is the received signal power in dBm by the n^{th} beacon at (x, y), and $d(\alpha_n^*, \beta_n^*, \alpha_k^*, \beta_k^*)$ is the distance between the n^{th} and k^{th} beacons. Thus, the average accuracy can be expressed as

$$\mathcal{U}(\boldsymbol{\alpha}^*, \boldsymbol{\beta}^*) = \sum_{x} \sum_{y} \left(g_{xy}(x, y) \left(\sum_{n=1}^{N} P(x, y, \alpha_n^*, \beta_n^*) + \sum_{n=1}^{N} \sum_{k=n+1}^{N} d(\alpha_n^*, \beta_n^*, \alpha_k^*, \beta_k^*) \right) \right), \tag{18}$$

and since $\sum_x \sum_y g_{xy}(x, y) = 1$, the above expression can be reformulated again to

$$\mathcal{U}(\boldsymbol{\alpha}^*, \boldsymbol{\beta}^*) = \sum_{x} \sum_{y} \left(g_{xy}(x, y) \sum_{n=1}^{N} P(x, y, \alpha_n^*, \beta_n^*) \right) + \sum_{n=1}^{N} \sum_{k=n+1}^{N} d(\alpha_n^*, \beta_n^*, \alpha_k^*, \beta_k^*). \tag{19}$$

One issue to consider while trying to optimize the above expression is that the received power could have the value of $-\infty$ dBm when a signal is not received at all, making it impossible to optimize since the average accuracy will be $-\infty$ if at least one signal is not received in at least one position. This could be resolved by using the linear scale instead of the log scale to express the received power, in which $-\infty$ dBm is replaced by 0 mW. However, allocating more power to areas where not much power is received in the log scale is much more noticeable than adding power to areas where the power received is already large, which is not the case when using the linear scale. For example, if we had a received signal with 1 nW of power, increasing it by 1 nW would increase it in the log scale by 3 dBm. On the other hand, if the originally received power is 1 μW, increasing it by 1 nW would increase it by just 4.34×10^{-3} dBm in the log scale. Thus, making the iterative methods like the GD more effective, since the gradient is very large at lower values of power compared to the higher

values of power, which pushes the algorithm to increase the power in areas where the received power is low and avoid increase the power in areas where the received power is already high.

The above can be mathematically done by modifying the received power scale in such a way as to have both of these luxuries (i.e., not having values of $-\infty$ and having a large gradient at when the received power is low), expressed as

$$\tilde{P}(x,y,\alpha_n^*,\beta_n^*) = 10\log_{10}\left(10^{-12} + 10^{\frac{P(x,y,\alpha_n^*,\beta_n^*)}{10}}\right). \tag{20}$$

The relation between $\tilde{P}(x,y,\alpha_n^*,\beta_n^*)$ and $P(x,y,\alpha_n^*,\beta_n^*)$ is shown in Figure 48. The distance between the beacons was also modified in the same manner, which could be expressed as

$$\tilde{d}(\alpha_n^*,\beta_n^*,\alpha_k^*,\beta_k^*) = 10\log_{10}\left(10^{-12} + d(\alpha_n^*,\beta_n^*,\alpha_k^*,\beta_k^*)\right), \tag{21}$$

where

$$d(\alpha_n^*,\beta_n^*,\alpha_k^*,\beta_k^*) = \sqrt{\left(\alpha_n^{*2} - \alpha_k^{*2}\right)^2 + \left(\beta_n^{*2} - \beta_k^{*2}\right)^2}, \tag{22}$$

and the objective function is formulated as

$$\begin{aligned}\mathcal{U}(\boldsymbol{\alpha}^*,\boldsymbol{\beta}^*) = \sum_x \sum_y \left(g_{xy}(x,y) \sum_{n=1}^{N} \tilde{P}(x,y,\alpha_n^*,\beta_n^*) \right) \\ + \sum_{n=1}^{N} \sum_{k=n+1}^{N} \tilde{d}(\alpha_n^*,\beta_n^*,\alpha_k^*,\beta_k^*)\,.\end{aligned} \tag{23}$$

When dealing with just one beacon, the last term of the objective function can be omitted and the objective function is then expressed as

$$\mathcal{U}(\boldsymbol{\alpha}^*,\boldsymbol{\beta}^*) = \sum_x \sum_y \left(g_{xy}(x,y) \sum_{n=1}^{N} \tilde{P}(x,y,\alpha_n^*,\beta_n^*) \right), \tag{24}$$

where $\boldsymbol{\alpha}^*$ and $\boldsymbol{\beta}^*$ are the variables to be optimized.

Many algorithms have been investigated to solve this problem, including random optimization algorithms, discrete optimization algorithms, and various continuous optimization

algorithms. One of the critical problems of that was constantly faced was having a lot of local maxima. For instance, Figure 60 shows the objective function when optimizing for one beacon's location. The objective function has a large value in the sensor's proximity where there is a large value of the PMS. The objective function's roughness also means that there are many local maxima, thus making it hard to optimize.

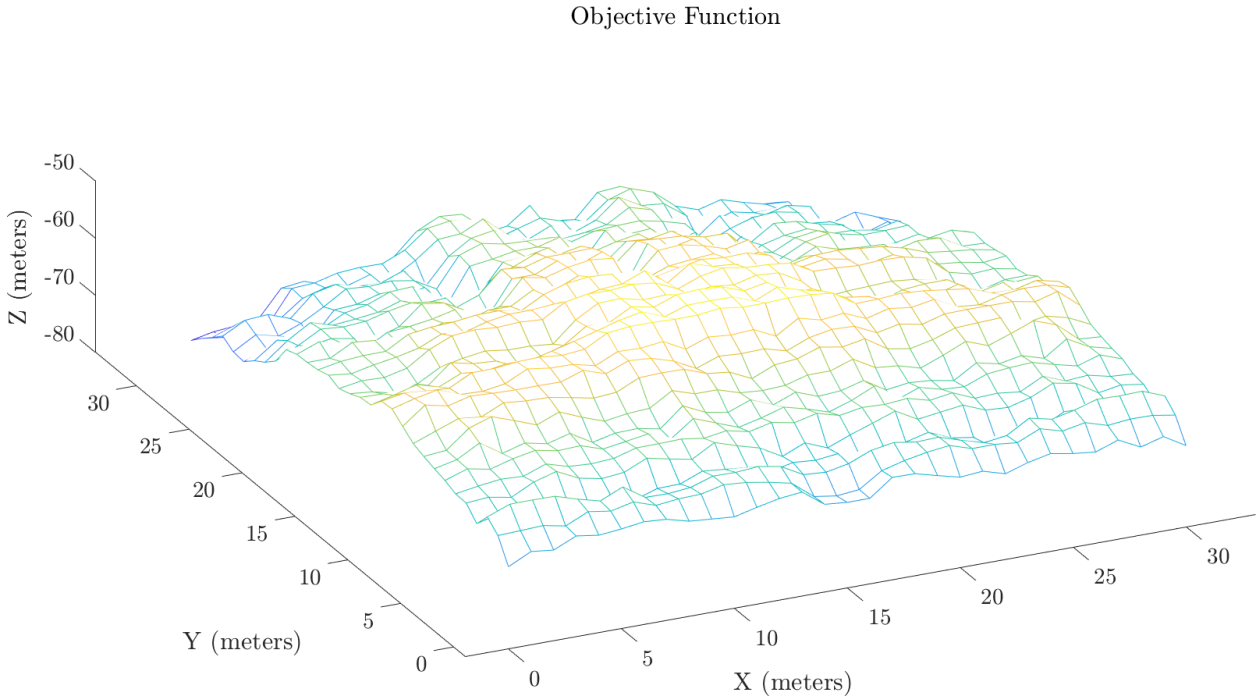

Figure 60: The objective function.

To overcome this issue, graduated optimization was used. The objective function is smoothed at every iteration. This smoothing is initially high but is reduced as more iterations are done. At each iteration, the smoothed function is optimized and the global maximum found is the starting point for the optimization process of the following smoothed objective function. Various smoothing filters could be used, but for this problem the Gaussian filter was chosen. The smoothed objective function using the Gaussian filter could be expressed as

$$\tilde{\mathcal{U}}(\boldsymbol{\alpha}^*, \boldsymbol{\beta}^*) = \mathop{\mathbb{E}}_{\boldsymbol{\epsilon} \sim \mathcal{N}(0, s^2)} \Bigg(\sum_x \sum_y \Bigg(g_{xy}(x, y) \sum_{n=1}^{N} \tilde{P}\big(x, y, \alpha_n^* + \epsilon_{\alpha_n}, \beta_n^* + \epsilon_{\beta_n}\big) \Bigg) + \sum_{n=1}^{N} \sum_{k=n+1}^{N} \tilde{d}\big(\alpha_n^* + \epsilon_{\alpha_n}, \beta_n^* + \epsilon_{\beta_n}, \alpha_k^* + \epsilon_{\alpha_k}, \beta_k^* + \epsilon_{\beta_k}\big) \Bigg). \tag{25}$$

In the proposed algorithm, both Newton's method and Gradient Ascent (GA) are used. Newton's method alone is not reliable since it uses the nearest critical point which could be a local minimum or a saddle point instead of a local maximum. Therefore, at every iteration, the eigenvalues of the Hessian matrix are checked. Newton's method is applied if the Hessian matrix is a negative-definite matrix (i.e., its eigenvalues are all negative). Otherwise, GA is used.

Since raytracing is used as the propagation model in the environment, objective function gradient cannot be computed in a closed-form manner. Therefore, it is computed numerically. The first derivative with respect to α_n^* is computed as

$$\frac{\partial \mathcal{U}(\boldsymbol{\alpha}^*, \boldsymbol{\beta}^*)}{\partial \alpha_n^*} \approx \frac{8\mathcal{U}(\boldsymbol{\alpha}^* + h\boldsymbol{\zeta}^n, \boldsymbol{\beta}^*) - 8\mathcal{U}(\boldsymbol{\alpha}^* - h\boldsymbol{\zeta}^n, \boldsymbol{\beta}^*) + \mathcal{U}(\boldsymbol{\alpha}^* - 2h\boldsymbol{\zeta}^n, \boldsymbol{\beta}^*) - \mathcal{U}(\boldsymbol{\alpha}^* + h\boldsymbol{\zeta}^n, \boldsymbol{\beta}^*)}{12h} \tag{26}$$

where h is a small number and $\boldsymbol{\zeta}^n \in \mathbb{R}^n$ is a vector with zero entries except the n^{th} element which has a value of 1. Similarly, the second derivative can be computed as

$$\frac{\partial^2 \mathcal{U}(\boldsymbol{\alpha}^*, \boldsymbol{\beta}^*)}{\partial \alpha_n^* \, \partial \alpha_k^*} \approx \frac{8\mathcal{U}'_{\alpha_n}(\boldsymbol{\alpha}^* + h\boldsymbol{\zeta}^k, \boldsymbol{\beta}^*) - 8\mathcal{U}'_{\alpha_n}(\boldsymbol{\alpha}^* - h\boldsymbol{\zeta}^k, \boldsymbol{\beta}^*)}{12h} + \frac{\mathcal{U}'_{\alpha_n}(\boldsymbol{\alpha}^* - 2h\boldsymbol{\zeta}^k, \boldsymbol{\beta}^*) - \mathcal{U}'_{\alpha_n}(\boldsymbol{\alpha}^* + h\boldsymbol{\zeta}^k, \boldsymbol{\beta}^*)}{12h}, \tag{27}$$

where $\mathcal{U}'_{\alpha_n} = \frac{\partial \mathcal{U}(\boldsymbol{\alpha}^*, \boldsymbol{\beta}^*)}{\partial \alpha_n^*}$. After computing the first and second derivatives, the gradient vector $\boldsymbol{g}$ and the Hessian matrix $\boldsymbol{H}$ can be easily obtained numerically. **ALGORITHM** shows the steps that were implemented to obtain the optimal placement of the beacons.

ALGORITHM: FINDING THE OPTIMAL PLACEMENT OF THE BEACONS

Input: $\boldsymbol{\alpha}^*, \boldsymbol{\beta}^*$
  while *stopping criterion not met* ***do***
    $\boldsymbol{g} \leftarrow \mathbf{0}$
    $\boldsymbol{H} \leftarrow \mathbf{0}$
    for $i = t \rightarrow m$
      Generate $\boldsymbol{\epsilon}$ *from* $\mathcal{N}(0, s(t)^2)$
      $\boldsymbol{g} \leftarrow \boldsymbol{g} + \nabla_{\boldsymbol{\alpha}^*, \boldsymbol{\beta}^*} \mathcal{U}(\boldsymbol{\alpha}^* + \boldsymbol{\epsilon}_{\boldsymbol{\alpha}}^*, \boldsymbol{\beta}^* + \boldsymbol{\epsilon}_{\boldsymbol{\beta}}^*)$
      $\boldsymbol{H} \leftarrow \boldsymbol{H} + \nabla^2_{\boldsymbol{\alpha}^*, \boldsymbol{\beta}^*} \mathcal{U}(\boldsymbol{\alpha}^* + \boldsymbol{\epsilon}_{\boldsymbol{\alpha}}^*, \boldsymbol{\beta}^* + \boldsymbol{\epsilon}_{\boldsymbol{\beta}}^*)$
    end for
    $\boldsymbol{g} \leftarrow \boldsymbol{g}/m$
    $\boldsymbol{H} \leftarrow \boldsymbol{H}/m$
    if $\boldsymbol{H}$ *is a negative-definite matrix*
      $\boldsymbol{\alpha}^*, \boldsymbol{\beta}^*$ *are updated according to Newton's method*
    else
      $\boldsymbol{\alpha}^*, \boldsymbol{\beta}^*$ *are updated according to GA with line search*
    end if
  end while

Notice that s^2 is a function of the iteration number. s^2 decreases every iteration according to a predefined function. For instance, $s(t)^2$ could be a rational, polynomial, linear, or any other decreasing function. In this work, $s(t)^2$ was set to $2/t$. The stopping criterion can be set based on a maximum number of iterations, a minimum gradient norm, or a minimum parameters difference.

Chapter 5 RESULTS AND DISCUSSION

5.1 *Results*

After figuring out the optimal configuration for both networks, we trained the ANN using the RSSI readings as features using the tuned hyperparameters. We then used the output of that network as a feature to the LSTM network in addition to the IMU readings and trained it on those using its tuned hyperparameters. In both training processes, we used 5 epochs with a batch size of 2048. This means that the data was passed to the networks 5 times and a subset of the training set of size 2048 was used to evaluate the gradient of the loss function. Before each epoch, the data was shuffled to prevent the DL model from learning the order of training and hence overfitting. As shown in Figures 61 and 62, we noticed an improvement in both the RMSE and the MAE. A comparison between the starting and final training RMSE and MAE and validation RMSE and MAE for both networks is shown in Tables 4 and 5. The validation RMSE and MAE are lower than the training RMSE and MAE because regularization mechanisms that were used (L2 regularization and dropout) are turned off at the time of validation.

It is important to note that the scaling of the data that was done produces outputs that are also scaled. This is acceptable in the case of training and validation, since we are only interested in a performance metric, but in the case of testing, "unscaling" of the data is done before calculating the RMSE or MAE to get a realistic measure of how good the predictions were. Using the trained LSTM network to predict the test data resulted in an RMSE of 0.4821 meters and an MAE of 0.4786 meters.

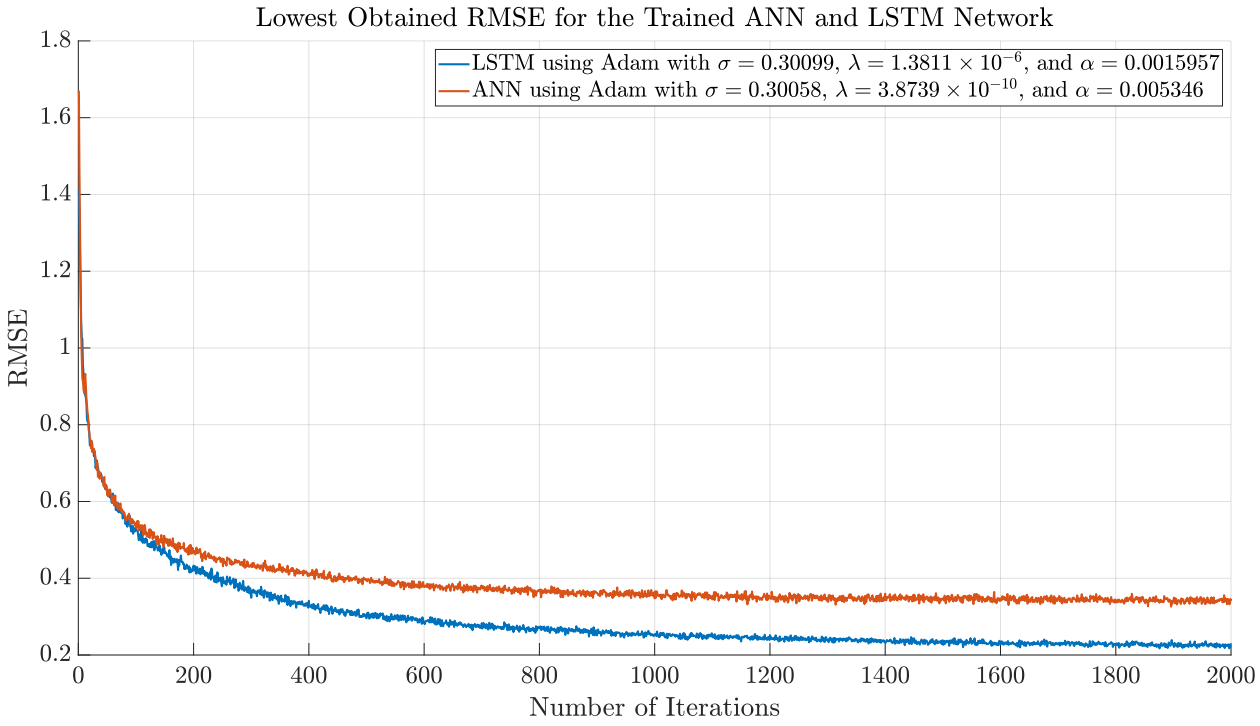

Figure 61: The best RMSE obtained from training the ANN and LSTM network using their tuned hyperparameters.

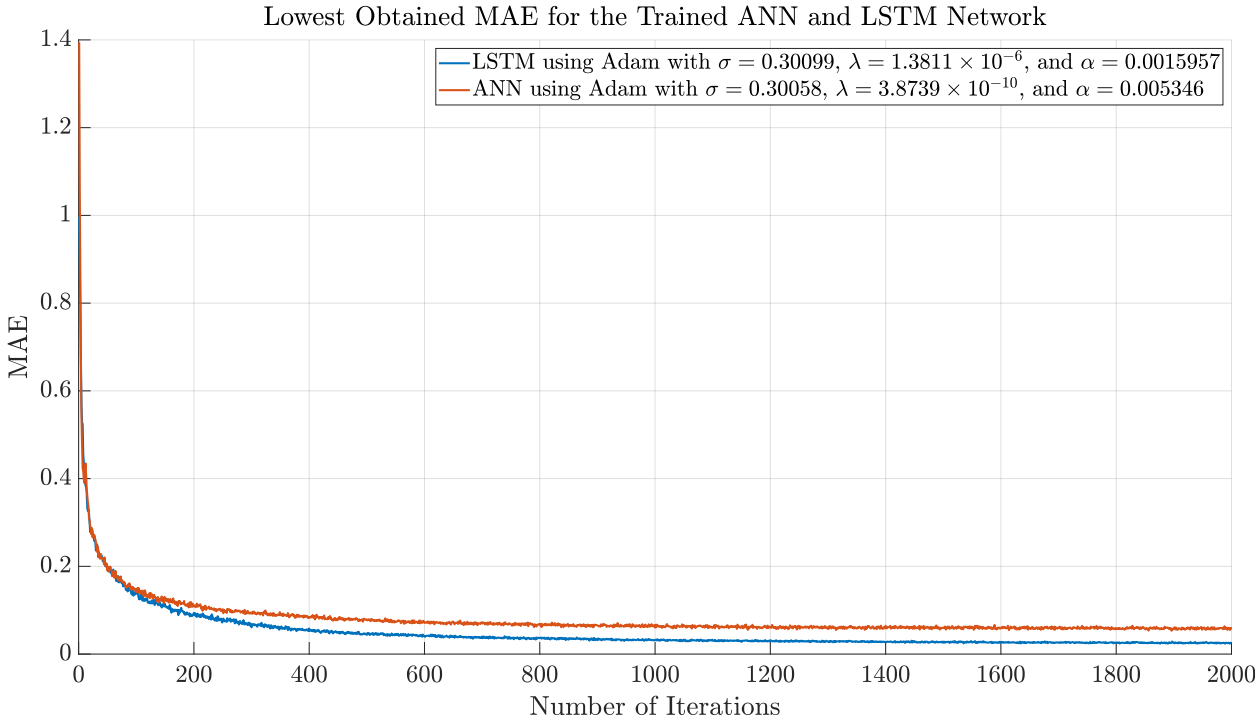

Figure 62: The best MAE obtained from training the ANN and LSTM network using their tuned hyperparameters.

Table 4: A comparison between the starting and final training RMSE and MAE for both networks.

Network	Starting Training RMSE	Final Training RMSE	Starting Training MAE	Final Training MAE
ANN	1.6702	0.3287	1.3948	0.0540
LSTM	1.4116	0.1996	0.9964	0.0199

Table 5: A comparison between the starting and final validation RMSE and MAE for both networks.

Network	Starting Validation RMSE	Final Validation RMSE	Starting Validation MAE	Final Validation MAE
ANN	1.4282	0.2419	1.0198	0.0292
LSTM	1.3006	0.0744	0.8458	0.0028

The ground truth and predicted paths using the ANN and the LSTM network for one of the paths are shown in Figures 63 and 64. It is important to note that RMSE on its own can be misleading at times. When referencing Figure 63, it could be noted that the the predicted path is very jagged and visually looks very off from the ground truth, yet the final training RMSE for the ANN is just 0.3287. This shows why it is important to plot the predictions on top of the ground truth, especially when it comes to positioning problems, to make sense of those predictions.

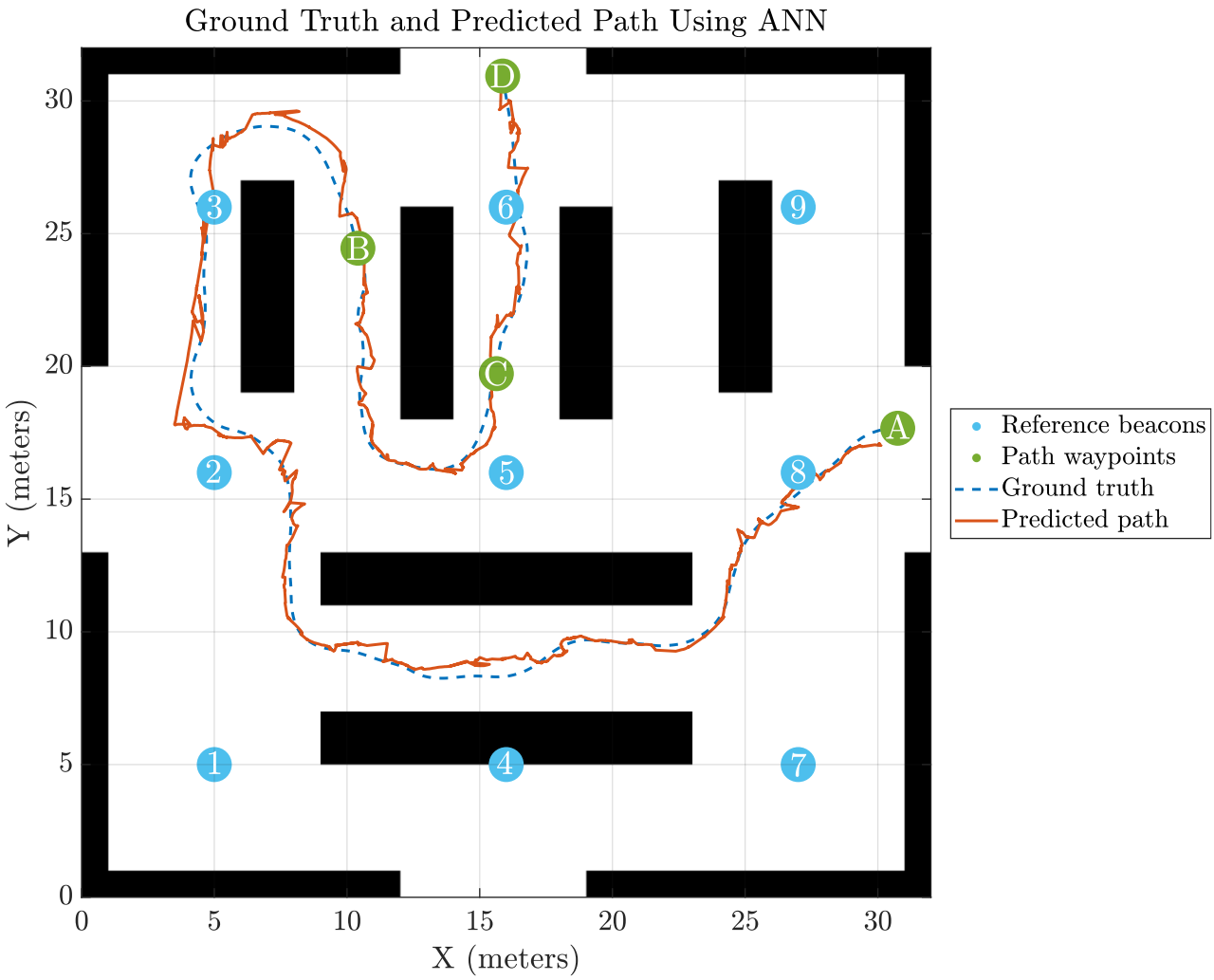

Figure 63: The ground truth and predicted paths using the ANN.

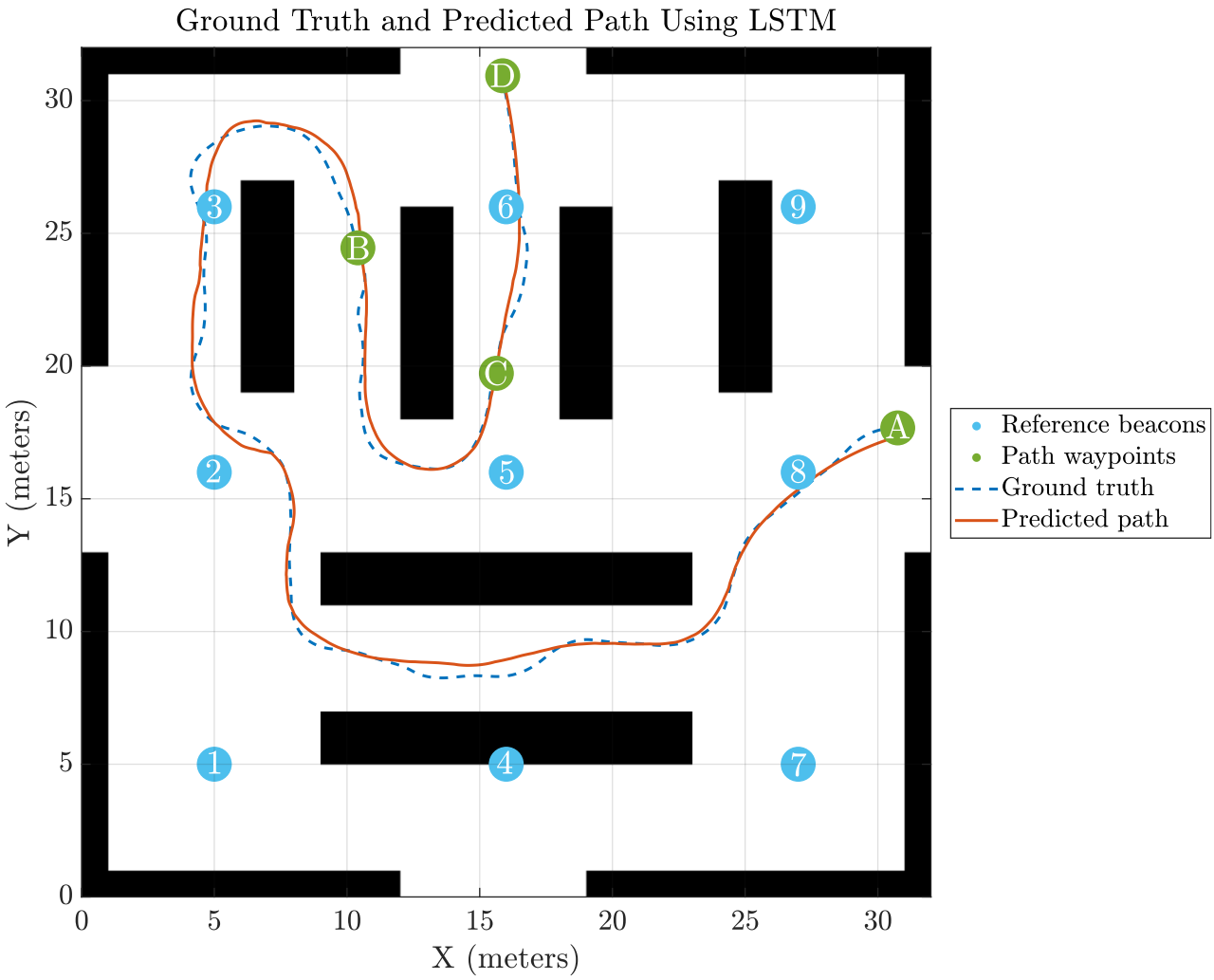

Figure 64: The ground truth and predicted paths using the LSTM network.

To effectively see the impact of the optimization algorithm on the DL model, we decreased the number of beacons in the environment. The increase in the objective function is not a good measure of whether the optimization process was a success or not since its formulation was based on two a prioris, which means that maximizing it does not necessarily mean that it would translate into a better performing DL model. The LSTM network was trained using the data of the initial and optimized locations. Figures 65-67 show the optimization algorithm results on scenarios with different numbers of beacons.

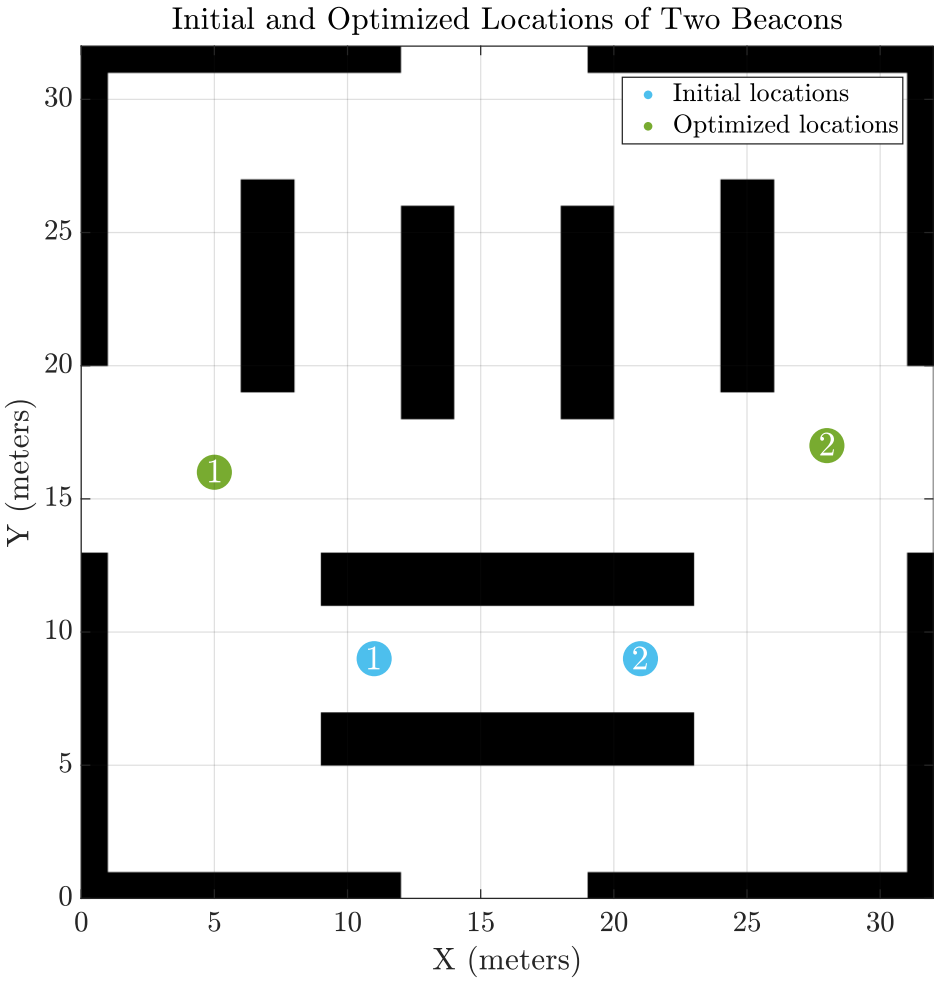

Figure 65: The initial and optimized locations for a two-beacon setup.

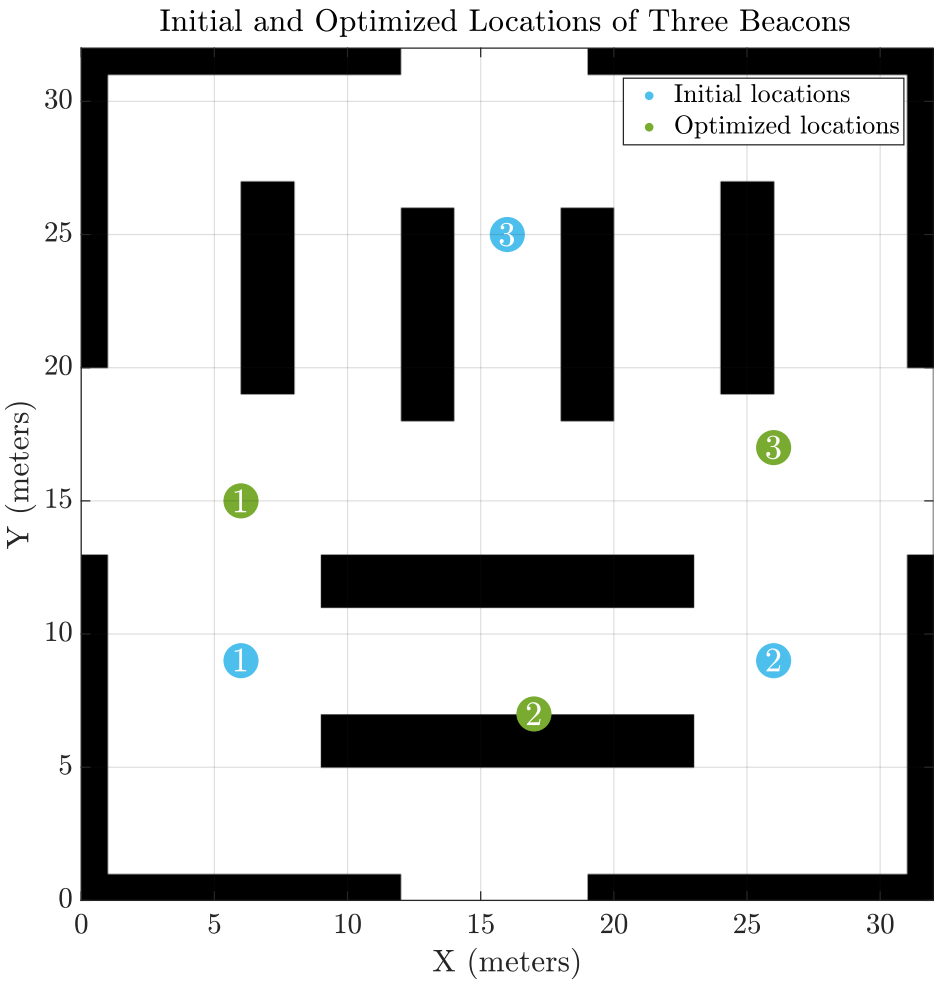

Figure 66: The initial and optimized locations for a three-beacon setup.

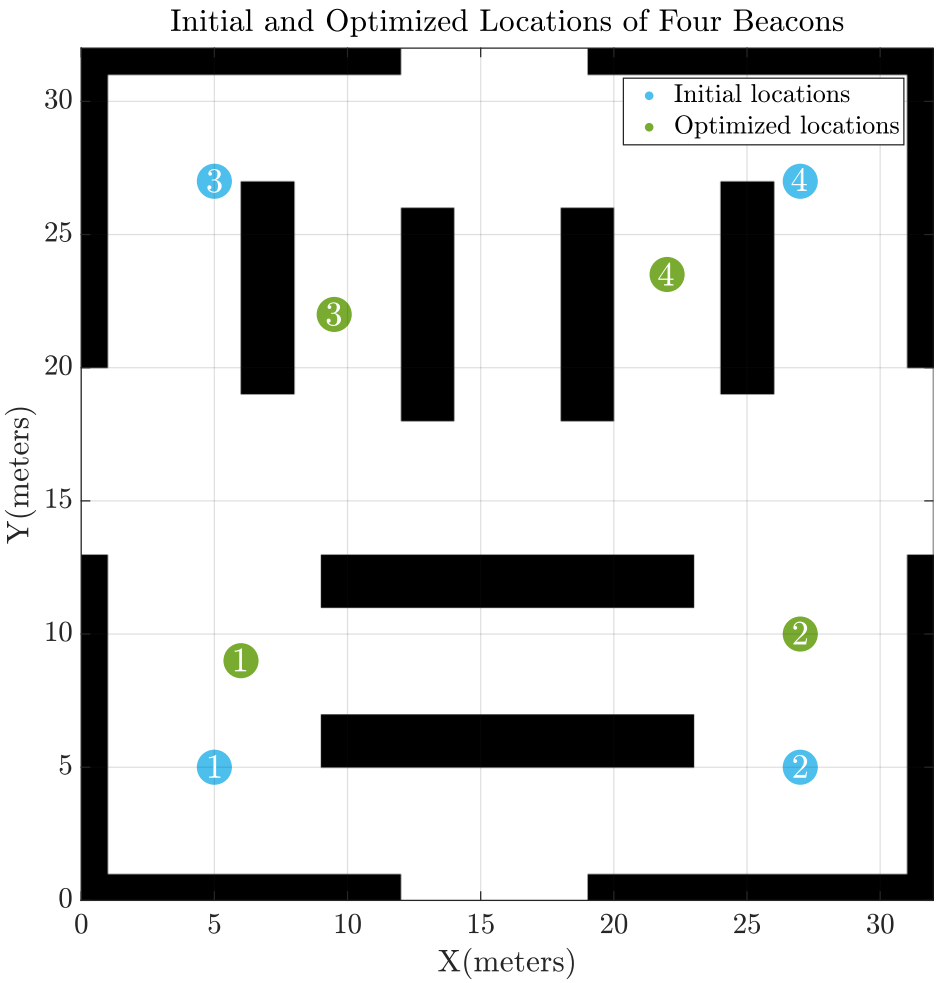

Figure 67: The initial and optimized locations for a two-beacon setup.

The reasonability of the results can be seen readily. For instance, in Figure 65 the two beacons were initially close to each other in the lower isle with a relatively low PMS compared to the upper isles. After applying the optimization algorithm, the beacons got separated by a large distance to create space diversity and were moved to the main isle which has the highest PMS out of all the isles, making the signals reach much of the map with a high level of power. To see the optimization effect on the average accuracy, the DL model was trained twice on the initial and optimized beacon locations for the scenario shown in Figure 66 using the LSTM network. Figures 68 and 69 show the RMSE and MAE of the LSTM network when trained in both cases with the same optimal hyperparameters obtained from the Bayesian optimization process. In contrast, Table 6 shows the starting and final RMSE and MAE in both situations. A predicted path using both the initial and optimized locations data is shown in Figure 70.

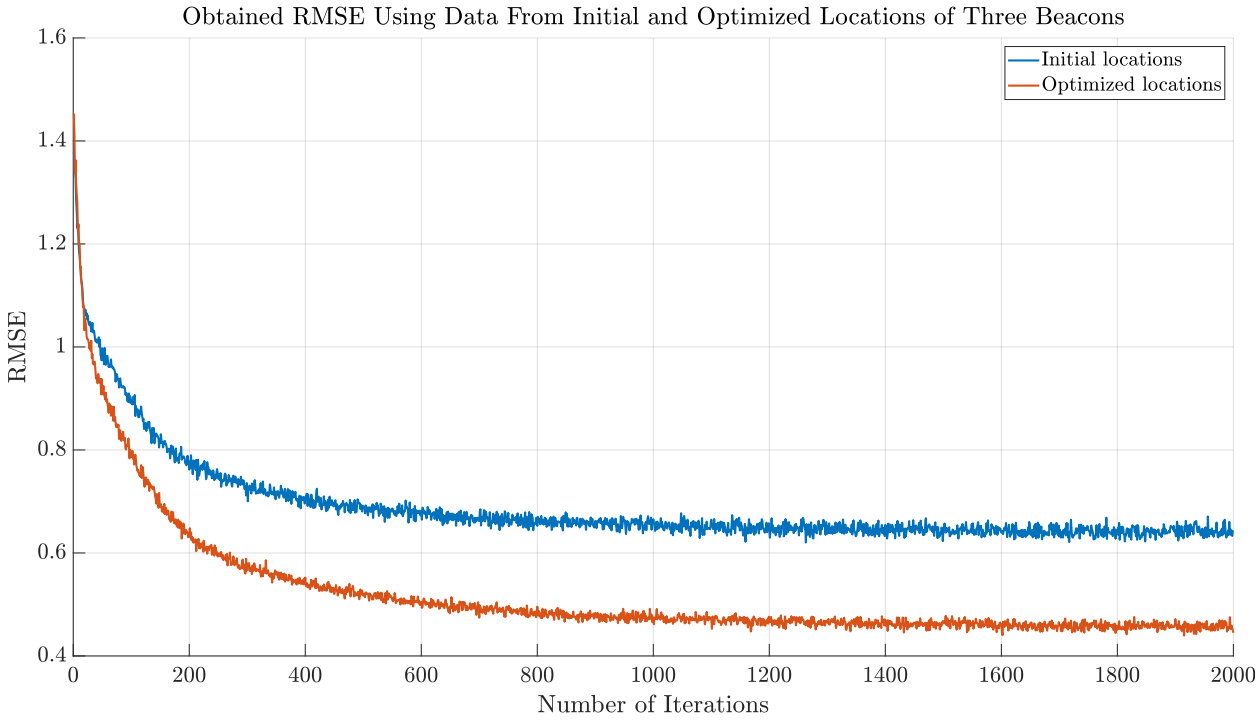

Figure 68: The resulting RMSE when training the LSTM network using initial and optimized beacon locations.

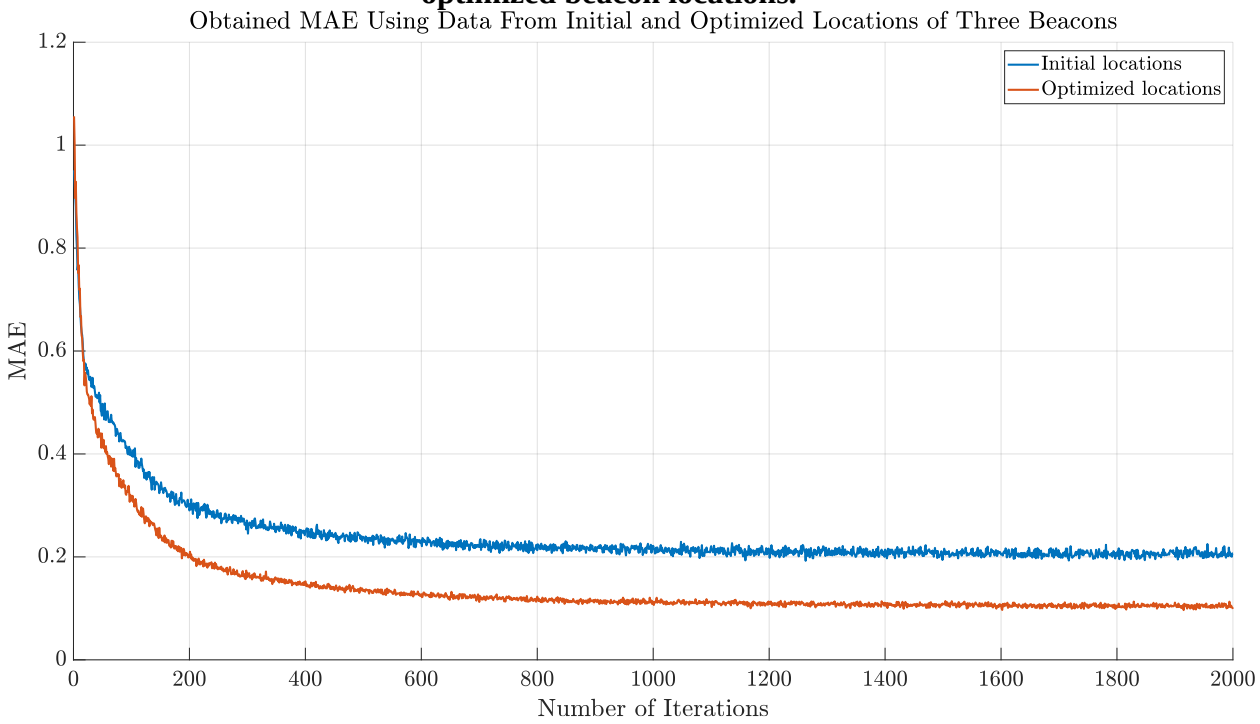

Figure 69: The resulting MAE when training the LSTM network using initial and optimized beacon locations.

Table 6: A comparison between the starting and final validation RMSE and MAE for both cases.

Locations	Starting Training RMSE	Final Training RMSE	Starting Training MAE	Final Training MAE
Initial	1.3792	0.6224	0.9510	0.1937
Optimized	1.4533	0.4499	1.0560	0.1012

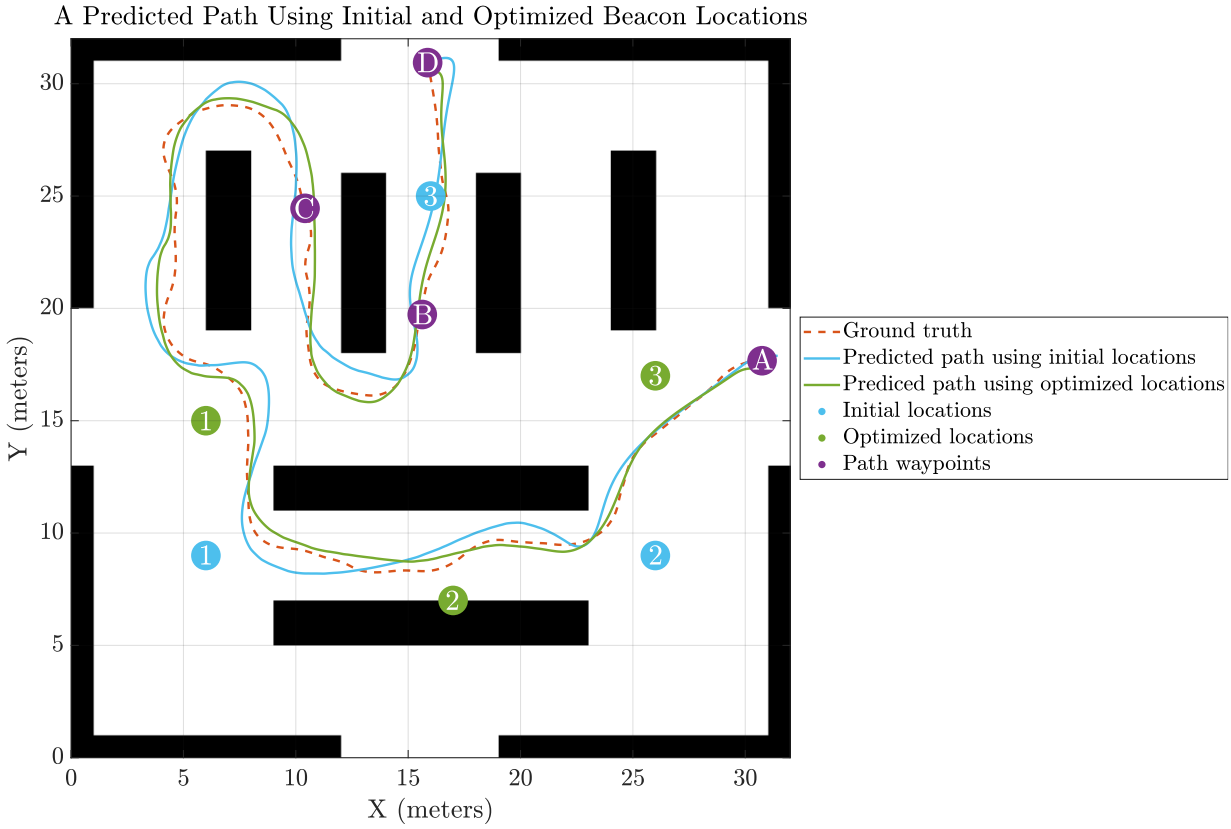

Figure 70: A predicted path using the initial and optimized locations using the LSTM network.

5.2 *IMPACT OF ENGINEERING SOLUTIONS*

Engineering solutions have always had a major impact upon society. In some cases, the impact has been mostly positive, such as in the case of house appliances and water purification, while in other cases, the impact has been mostly negative, as in the case of bombs with ever-increasing destructive power. In most cases, engineering solutions offered positives and negatives, as in the case of the automobile industry. Engineers have a responsibility to give proper attention to the cost of their solutions, but more importantly, to the safety of said solutions as they affect society as a whole [46].

Our work suggests moving towards an AI-driven future. The fourth industrial revolution promises enormous potential solutions in business, manufacturing, healthcare, education, military, and many other areas. Examples of this include, but are not limited to, facial recognition which could eliminate the need for human monitoring of houses by detecting break-ins and calling for emergency services, self-driving cars which could reduce traffic accidents and deaths, and in cybersecurity by improving the efficiency of identifying threats which could reduce the time needed to respond to incidents [47].

The enhancement of indoor positioning systems would deliver great value for different industries and environments. Indoor robots, IoT devices and humans need precise information on their positioning to enhance solutions with navigation as the most important application. Indoor positioning systems are also important for self-driving cars. While vehicles are driving outdoors most of the time with good GNSS coverage, they need to drive into tunnels, underground parking and urban canyons where they require indoor systems.

Building the project on the base of Wi-Fi infrastructure would allow easy adaptation of the new solution into already existing systems. Moreover, the integration of IMU sensors would enhance the positioning solution for several types of devices. The inexpensive IMUs based on MEMS technology are embedded in robots, mobile phones, and a diverse spectrum of IoT devices and gadgets. The positioning solution proposed would not require big infrastructure changes to function. Moreover, the optimization solution would help designers to best position the Wi-Fi nodes to get the best performance with available resources.

Chapter 6 CONCLUSIONS & FUTURE WORK

6.1 *Conclusions*

In this project, we have achieved work that is divided mainly into three parts; we designed a novel indoor positioning system using the minimum equipment installation cost and equipment cost possible while achieving good accuracies. This was done by fusing RSSI measurements from reference Wi-Fi beacons with measurements from an IMU in a robot. We have done simulation for both IMU and Wi-Fi RSSI measurements for a robot moving inside a 32 × 32 × 7 meters warehouse floor. Then, a DL model that uses LSTM cells was built to find the position of the robot within the indoor environment. Finally, the placement of those reference beacons was optimized such that the accuracy of the predicted position was maximized.

Results showed an RMSE of the predicted position of **0.4821** meters and an MAE of **0.4786 meters** using the developed LSTM model. The optimized placement of the beacons resulted in a **27.71%** decrease in the RMSE and a **47.75%** decrease in the MAE of the DL model.

6.2 *Future Work*

1- Synergistically combine the different parts of the project into one system with the scheme of the environment as its input and a detailed indoor positioning plan as the output. i.e., specify the type of indoor technology to be used, the distribution, the expected accuracy, etc…

2- Using LSTM network is computationally intensive, and its power consumption is unfeasible in the context of most rovers and indoor robots, therefore, modifying the deep learning model into a simpler one without significantly sacrificing the accuracy is critical.

3- Modify the optimization algorithm, not only to include the optimized distribution of the beacons, but also the optimized number of beacons to be employed, and the type of beacon to be used, accounting for a more detailed scheme of the environment (which includes the dominant types of noise and the minimum acceptable positioning accuracy).

Chapter 7 REFERENCES

[1] Goldstein, “Global Indoor Positioning and Indoor Navigation (IPIN) Market 2020-2024.” .

[2] O. J. Woodman, “An introduction to inertial navigation,” *Univ. Cambridge Comput. Lab.*, 2007.

[3] Y. Zhuang, J. Yang, Y. Li, L. Qi, and N. El-Sheimy, “Smartphone-based indoor localization with bluetooth low energy beacons,” *Sensors (Switzerland)*, vol. 16, no. 5, 2016, doi: 10.3390/s16050596.

[4] Y. Zhuang, H. Lan, Y. Li, and N. El-Sheimy, “PDR/INS/WiFi integration based on handheld devices for indoor pedestrian navigation,” *Micromachines*, vol. 6, no. 6, 2015, doi: 10.3390/mi6060793.

[5] D. Halperin, W. Hu, A. Sheth, and D. Wetherall, “Tool release: Gathering 802.11n traces with channel state information,” *Comput. Commun. Rev.*, vol. 41, no. 1, 2011, doi: 10.1145/1925861.1925870.

[6] M. Ciurana, F. Barcelo-Arroyo, and F. Izquierdo, “A ranging system with IEEE 802.11 data frames,” 2007, doi: 10.1109/RWS.2007.351785.

[7] N. El-Sheimy and A. Youssef, “Inertial sensors technologies for navigation applications: state of the art and future trends,” *Satell. Navig.*, vol. 1, no. 1, 2020, doi: 10.1186/s43020-019-0001-5.

[8] N. El-Sheimy and X. Niu, “The promise of MEMS to the navigation community,” *Insid. GNSS*, vol. 2, pp. 46–56, Jan. 2007.

[9] S. Fingerman, “Introduction to GPS; the Global Positioning System, 2d ed,” *Sci-Tech News*, vol. 61, no. 1. 2007.

[10] Pamela Fox, “Geolocation.” .

[11] Wikipedia, “Six degrees of freedom.” .

[12] Rossi and Braindrain0000, “Depiction of Trilateration.” .

[13] Wikipedia, “Time of flight.” .

[14] P. Society, *IEEE Std 145-2013 (Revision of IEEE Std 145-1993)*, vol. 2013. 2014.

[15] Wikipedia, “Free-space path loss.” .

[16] L. E. Frenzel, *Principles of Electronic Communication Systems*, vol. 28, no. 10. 2015.

[17] I. H. Alshami, N. A. Ahmad, and S. Sahibuddin, “People effects on WLAN-Based IPS’ accuracy experimental preliminary results,” 2014, doi: 10.1109/MySec.2014.6986015.

[18] Z. Yun and M. F. Iskander, “Ray tracing for radio propagation modeling: Principles and applications,” *IEEE Access*, vol. 3. 2015, doi: 10.1109/ACCESS.2015.2453991.

[19] Analog Devices, “ADIS16448: Compact, Precision Ten Degrees of Freedom Inertial Sensor Data Sheet (Rev. H),” 2019. .

[20] Tom Mitchell, “Machine Learning textbook,” *McGraw Hill*, 1997. .

[21] A. C. Ian Goodfellow, Yoshua Bengio, *Deep Learning - Ian Goodfellow, Yoshua Bengio, Aaron Courville - Google Books*. 2016.

[22] S. Sharma, S. Sharma, and A. Athaiya, “ACTIVATION FUNCTIONS IN NEURAL NETWORKS,” *Int. J. Eng. Appl. Sci. Technol.*, vol. 04, no. 12, 2020, doi: 10.33564/ijeast.2020.v04i12.054.

[23] M. A. Cauchy, “Méthode générale pour la résolution des systèmes d’équations simultanées,” *C. R. Hebd. Seances Acad. Sci.*, vol. 25, no. 2, 1847.

[24] S. Jain, “An Overview of Regularization Techniques in Deep Learning (with Python code).” .

[25] L. Bottou, “Online Algorithms and Stochastic Approximation,” *Online Learning and Neural Networks*. 1998.

[26] D. P. Kingma and J. L. Ba, “Adam: A method for stochastic optimization,” 2015.

[27] N. Qian, “On the momentum term in gradient descent learning algorithms,” *Neural Networks*, vol. 12, no. 1, 1999, doi: 10.1016/S0893-6080(98)00116-6.

[28] J. Duchi, E. Hazan, and Y. Singer, “Adaptive subgradient methods for online learning and stochastic optimization,” *J. Mach. Learn. Res.*, vol. 12, 2011.

[29] Y. Lecun and Y. Bengio, “Convolutional Networks for Images , Speech , and Time Series Variable-Size Convolutional Networks : SDNNs,” *Processing*, vol. 3361, no. 10, 2010.

[30] S. Boyd, S. P. Boyd, and L. Vandenberghe, *Convex optimization*. Cambridge university press, 2004.

[31] J. Nocedal and S. Wright, *Numerical optimization*. Springer Science & Business Media, 2006.

[32] R. G. Parker and R. L. Rardin, *Discrete optimization*. Elsevier, 2014.

[33] P. J. M. van Laarhoven and E. H. L. Aarts, “Simulated annealing BT - Simulated Annealing: Theory and Applications,” P. J. M. van Laarhoven and E. H. L. Aarts, Eds. Dordrecht: Springer Netherlands, 1987, pp. 7–15.

[34] M. Dorigo, M. Birattari, and T. Stutzle, “Ant colony optimization,” *IEEE Comput. Intell. Mag.*, vol. 1, no. 4, pp. 28–39, 2006, doi: 10.1109/MCI.2006.329691.

[35] J. J. Moré and Z. Wu, “Global continuation for distance geometry problems,” *SIAM J. Optim.*, vol. 7, no. 3, 1997, doi: 10.1137/s1052623495283024.

[36] E. Hazan, K. Y. Levy, and S. Shalev-Shwartz, “On graduated optimization for stochastic non-convex problems,” in *33rd International Conference on Machine Learning, ICML 2016*, 2016, vol. 4.

[37] Autodesk, “Tinkercad,” 2011. .

[38] S. M. LaValle and J. J. Kuffner, “Randomized kinodynamic planning,” *Int. J. Rob. Res.*, vol. 20, no. 5, pp. 378–400, 2001, doi: 10.1177/02783640122067453.

[39] S. Karaman and E. Frazzoli, “Incremental sampling-based algorithms for optimal motion planning,” in *Robotics: Science and Systems*, 2011, vol. 6, doi: 10.15607/rss.2010.vi.034.

[40] MathWorks, “inflate Function,” 2019. .

[41] R. C. Coulter, “Implementation of the Pure Pursuit Path Tracking Algorithm,” *Communication*, no. January, 1992.

[42] A. W. Flux and V. Pareto, “Cours d’Economie Politique.,” *Econ. J.*, vol. 7, no. 25, 1897,

doi: 10.2307/2956966.

[43] "About Feature Scaling and Normalization – and the effect of standardization for machine learning algorithms," *Sebastian Raschka*, 2014. .

[44] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *J. Mach. Learn. Res.*, vol. 13, 2012.

[45] J. Wu, X. Y. Chen, H. Zhang, L. D. Xiong, H. Lei, and S. H. Deng, "Hyperparameter optimization for machine learning models based on Bayesian optimization," *J. Electron. Sci. Technol.*, vol. 17, no. 1, 2019, doi: 10.11989/JEST.1674-862X.80904120.

[46] N. D. Okamoto, J. Rhee, and N. J. Mourtos, "Educating students to understand the impact of engineering solutions in a global / societal context," in *8th UICEE Annual Conference on Engineering Education*, 2005, no. February.

[47] W. Wang and K. Siau, "Artificial intelligence, machine learning, automation, robotics, future of work and future of humanity: A review and research agenda," *J. Database Manag.*, vol. 30, no. 1, 2019, doi: 10.4018/JDM.2019010104.