FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO



A Machine Learning Model for Indoor Positioning of Bluetooth Receivers

Francisco Pêgo dos Santos Monteiro

Mestrado em Engenharia Eletrotécnica e de Computadores

Orientador: Nuno Miguel Cardanha Paulino

Co-orientador: Luís Manuel de Sousa Pessoa

July 28, 2023

© Francisco Monteiro, 2023

Abstract

One of the challenges being addressed in the past few years in the realm of Internet-of-Things (IoT) applications is the development of a standard Indoor Positioning System (IPS). While the Global Positioning System (GPS) has been globally used for many years with regard to outdoor environments, several approaches have emerged with the purpose of achieving similar accuracy in indoor environments. One such approach takes advantage of the Bluetooth Low Energy (BLE) 5.1 specification, which recently introduced a Direction Finding (DF) feature based on Angle-of-Arrival (AoA), giving room for new possibilities in the development of novel low-cost indoor localization systems.

This dissertation was developed in the context of an existing tracking system for forklifts and other indoor factory floor machinery, based on BLE technology and relying on AoA to perform position estimation. As existent state-of-the-art algorithms for AoA estimation are computationally demanding and have a low tolerance to noisy data, this creates the need for more robust methods of inferring the AoA. In this context, one of the most promising approaches to address this challenge is the use of Machine Learning (ML) techniques. In the last few years, the ML field has been actively researched and developed and its techniques are being increasingly applied to a wide range of scientific areas. One very interesting and powerful subset of ML techniques are the Neural Networks (NNs), which are a set of algorithms that mimic the operation of a human brain. NNs appear as a very promising solution to apply in AoA estimation, since, after being subject to proper training processes, they are able to produce very reliable output estimations based on noisy input data.

In this context, the main goal of this dissertation was to study and develop ML models to estimate the AoAs of transmissions related to the aforementioned tracking system for forklifts, based on data collected in earlier stages of the system's development. Three NNs and a K-Nearest-Neighbor (KNN) model were developed. The models were integrated in an existing Java based simulator framework, which evaluates the tracking system in question. The accuracy levels produced by the ML models, as well as their computational cost, were evaluated in comparison with the previously used algorithms, showing significant improvements.

Contents

1 Introduction **1**
- 1.1 Context . . . 1
- 1.2 Motivation . . . 2
- 1.3 Objectives . . . 2
- 1.4 Document Structure . . . 2

2 Related Work **4**
- 2.1 Background . . . 4
  - 2.1.1 Wireless Indoor Localization . . . 4
  - 2.1.2 Bluetooth Low Energy . . . 6
  - 2.1.3 Position Estimation via Angle-of-Arrival . . . 7
  - 2.1.4 Machine Learning . . . 9
- 2.2 State-of-the-Art . . . 16
- 2.3 Conclusions and Closing Remarks . . . 22

3 Proposed Approach **24**
- 3.1 Phase Difference Datasets for UCA with 8 Elements . . . 25
  - 3.1.1 Phase Difference Data . . . 25
  - 3.1.2 Dataset Features and Ground-Truth . . . 26
- 3.2 Machine Learning Framework . . . 28
- 3.3 AoA Estimation Models . . . 29
  - 3.3.1 Data Preprocessing . . . 29
  - 3.3.2 K-Nearest Neighbors . . . 30
  - 3.3.3 Neural Networks . . . 32
  - 3.3.4 Heuristic Algorithm . . . 36
- 3.4 Simulator . . . 36

4 Performance Evaluation **39**
- 4.1 Angle-of-Arrival Estimation . . . 39
  - 4.1.1 Experimental Setup . . . 39
  - 4.1.2 AoA Inference Results with KNN . . . 40
  - 4.1.3 AoA Inference Results with NNs . . . 41
  - 4.1.4 AoA Inference Results with Heuristic . . . 44
  - 4.1.5 Performance Comparison . . . 46
- 4.2 Integration with Simulator . . . 49
  - 4.2.1 Tracking Results with KNN AoA Estimation . . . 49
  - 4.2.2 Tracking Results with NN AoA Estimation . . . 50
  - 4.2.3 Tracking Results with CNN AoA Estimation . . . 52

4.2.4 Tracking Results with Heuristic AoA Estimation 53

5 Conclusion and Future Work **55**

References **57**

List of Figures

2.1 Angle of Departure and Angle of Arrival [1] . . . 7
2.2 Position estimation based on AoA, with two reference points [2] . . . 7
2.3 Estimation of position via received AoA and least-squares method . . . 8
2.4 AoA based localization with antenna array . . . 9
2.5 Estimation example of KNN used for regression, with k=3 . . . 11
2.6 Schema of a Decision Tree [3] . . . 12
2.7 Neural Network example [4] . . . 13
2.8 Example of a single-layer feedforward network [5] . . . 14
2.9 Example of a multi-layer feedforward network [5] . . . 15
2.10 Example of a feedback network [5] . . . 15
2.11 A simple Convolutional Neural Network (CNN) architecture . . . 16
2.12 Block diagram of the networks [6] . . . 18
2.13 The block diagram of the proposed CNN-based AOA estimation [7] . . . 19
2.14 The proposed CNN architecture [7] . . . 20
2.15 Block diagram of the proposed deep ensemble model [8] . . . 21

3.1 Aerial view of the area where the real-world experiment was performed [9] . . . 25
3.2 Diagram of the circular antenna array [10] . . . 26
3.3 Profile of phase difference values for all adjacent antenna pairs, as a function of four different AoAs [10] . . . 26
3.4 6 data points belonging to one of the generated datasets . . . 27
3.5 Four phase difference profiles generated with different standard deviations [9] . . . 28
3.6 Example scheme of the nearest neighbor search . . . 32
3.7 Summary of the first model structure . . . 33
3.8 Summary of the second model structure . . . 35
3.9 Summary of the CNN structure . . . 36
3.10 Example of position estimation via least-squares method . . . 37
3.11 Position tracking example . . . 38
3.12 Simulator operation diagram . . . 38

4.1 Mean Absolute Error (MAE) as a function of noise for the KNN model, using datasets with different number of features . . . 40
4.2 Scatter plots of the AoA estimations obtained with datasets of different noise (KNN) 41
4.3 MAE as a function of noise for the three NN models (8 Features) . . . 42
4.4 Scatter plots of the AoA estimations obtained with datasets of different noise (CNN) 42
4.5 MAE as a function of noise for the three NN models (16 Features) . . . 43
4.6 MAE as a function of noise for the three NN models (32 Features) . . . 44

4.7 MAE as a function of noise for the Heuristic model, using datasets with different number of features . . . 45
4.8 Scatter plots of the AoA estimations obtained with datasets of different noise (Heuristic) . . . 45
4.9 MAE as a function of noise for three different models (8 Features) . . . 46
4.10 MAE as a function of noise for three different models (16 Features) . . . 47
4.11 MAE as a function of noise for three different models (32 Features) . . . 48
4.12 Root Mean Squared Error (RMSE) as a function of noise for the KNN, using datasets with different number of features . . . 50
4.13 Evolution of the estimated trajectory with increasing noise, for the KNN model (8 feature data) . . . 50
4.14 RMSE as a function of noise for the second NN, using datasets with different number of features . . . 51
4.15 Evolution of the estimated trajectory with increasing noise, for the second NN model (8 feature data) . . . 51
4.16 RMSE as a function of noise for the CNN, using datasets with different number of features . . . 52
4.17 Evolution of the estimated trajectory with increasing noise, for the CNN model (8 feature data) . . . 53
4.18 RMSE as a function of noise for the Heuristic model, with different number of data features . . . 53
4.19 Evolution of the estimated trajectory with increasing noise, for the Heuristic model 54

List of Tables

2.1 Parameters of the five CNNs evaluated in [8] 21
2.2 Summary of the AoA estimation approaches 22

3.1 Hyper-parameters of the first model 35

4.1 Average training and testing times of the KNN model for each dataset type 41
4.2 MAE of each NN model for 8 feature datasets 43
4.3 MAE of each NN model for 16 feature datasets 43
4.4 MAE of each NN model for 32 feature datasets 44
4.5 MAE, training time and testing time of each NN model for the real world dataset 44
4.6 Average testing time of the Heuristic model for each dataset type 46
4.7 Average AoA inference time for each model (8 Features) 47
4.8 MAE of each model for the real world dataset 48
4.9 Average AoA inference time for each model (Real World Dataset) 48
4.10 Simulation Parameters 49

Abbreviations

AoA	Angle-of-Arrival
AoD	Angle-of-Departure
AI	Artificial Intelligence
ANN	Artificial Neural Network
BLE	Bluetooth Low Energy
CNN	Convolutional Neural Network
CSI	Channel State Information
CTE	Constant Tone Extension
DNN	Deep Neural Network
DF	Direction Finding
DL	Deep Learning
DL4J	DeepLearning4J
ESPRIT	Estimation of Signal Parameters via Rotational Invariance Techniques
GPS	Global Positioning System
IoT	Internet-of-Things
JVM	Java Virtual Machine
KNN	K-Nearest Neighbors
ML	Machine Learning
MAE	Mean Absolute Error
MSE	Mean Squared Error
MLP	Multi-Layer Perceptron
MUSIC	Multiple Signal Classification
NN	Neural Network
OFDM	Orthogonal Frequency-division Multiplexing
RFID	Radio Frequency Identification Device
RMSE	Root Mean Squared Error
RSSI	Received Signal Strength Indication
TDoA	Time-Difference-of-Arrival
ToA	Time-of-Arrival
UCA	Uniform Circular Array
ULA	Uniform Linear Array
UWB	Ultra-Wide-Band

Chapter 1

Introduction

1.1 Context

As the IoT becomes more prevalent, and more efficient telecommunications are implemented, there is a growing demand for smarter devices that can operate more independently from centralized systems, as well as more ambitious edge computing applications. Among those applications is real-time localization of devices or assets in specific indoor spaces [11]. While GPS has been extensively used with regard to localization systems for outdoor environments, when it comes to indoor positioning systems, there is still no standard solution offering similar levels of accuracy. In the past few years, several approaches have emerged, relying on different wireless sensor meshes and different positioning techniques, aiming to progressively improve the accuracy of Indoor Positioning Systems. In this context, the recent Bluetooth 5.1 version gave rise to new possibilities in the realm of real-time indoor localization systems, since it introduced a new DF feature based on AoA [11].

This dissertation is presented in the context of an existing tracking system for forklifts and other indoor factory floor machinery. The solution is based on Bluetooth beacons, which are small wireless devices that work based on BLE. The beacons are placed at fixed locations in a factory floor, and periodically send packets with their ID. The forklift is a moving vehicle, equipped with an antenna array, which receives the beacons and can determine its stationary position on the factory floor by knowing the relative AoAs of each incoming beacon. Previous work has implemented a Java based simulator [11] which evaluates the aforementioned solution using AoA data gathered in real-world experiments using a commercial antenna array receiver [9].

Briefly, the direction finding is performed by sampling a Constant Tone Extension (CTE) which is appended to the end of a Bluetooth packet. The phase of this tone can be sampled by each of the antennas that make up the array on the receiver, and the AoA of the signal can be calculated using the phase differences between the antennas [9]. In the simulation solution, the AoA is generated directly, and is not computed from the phase differences, which would be the real world scenario. Additionally, current AoA calculation methods are simple heuristics, which

can fail if the sampled antenna phase data is noisy. This creates a demand for more robust methods to infer the AoA from the phase differences of the antennas.

1.2 Motivation

In order to meet the demand of obtaining more accurate AoA estimations from the antennas' phase differences, the most promising approach to follow is the use of ML methods.

The ML field has been actively researched and developed in the last few years. This field includes a variety of techniques that are being increasingly applied to a wide range of scientific areas, with positive outcomes. More specifically, in the context of this work, the use of ML techniques appears as the most suitable solution, since ML algorithms are capable of handling noisy measurements and learning complex nonlinear relationships between the input and output values of a system.

Regarding the application of ML methods in AoA estimation, although there are some recent approaches [12], [13] that are attempting to evaluate its viability, there is still significant space for improvement as the existent solutions have not been largely exploited so far.

Benefits that a ML based solution may provide include lower computational cost relative to existing algorithms, leading to faster calculations of AoAs, which is a crucial step towards the construction of an efficient real-time localization system.

1.3 Objectives

This dissertation addressed two primary objectives. Firstly, we aimed to study ML models to estimate the AoAs of transmissions, based on data related to the phase of the signals that was captured by the antenna array of a receiver board. Preliminary work had already been conducted on deriving the AoA from phase difference measurements[1]. The accuracy and computational cost of the developed models were then evaluated and compared with an existing heuristic model, used in earlier stages of the system development. For this purpose, a set of synthetic datasets with different noise levels and also a dataset collected in real world experiments were used, aiming to evaluate the error of the AoA estimations as a function of different noise levels in the data.

Secondly, the ML models were integrated into the existing simulator framework, in order to evaluate the tracking accuracy produced by the AoA estimations of each model under different simulated conditions for the movement of the receiver, and the number of BLE beacons, among other parameters.

1.4 Document Structure

This chapter introduced the context in which this dissertation is presented, as well as its main motivations and objectives. The next chapters of this document are structured as follows: Chapter

[1] https://github.com/JJJJJJJJJJJJJJl/NNsIndoorPosition

2 explores the background technologies and techniques related to this work. It also details some existent approaches for AoA estimation in indoor tracking systems. In Chapter 3, the proposed approach to the development of the ML models for AoA estimation is detailed. Chapter 4 presents the results of the experimental tests carried out to evaluate the AoA estimations produced by the developed models. Chapter 5 presents the main conclusions drawn from this dissertation and points out the future work on this subject.

Chapter 2

Related Work

2.1 Background

In this section, the most relevant theoretical aspects related to this dissertation are presented. It is divided into four subsections. In subsection 2.1.1, a summary of the most popular technologies and techniques for wireless indoor localization is given. An overview of the Bluetooth Low Energy technology is presented in subsection 2.1.2, followed by an explanation of the Angle-of-Arrival technique in subsection 2.1.3. Finally, the core concepts behind ML are summarized in subsection 2.1.4, followed by a detailed description of the most relevant ML algorithms in the context of this dissertation.

2.1.1 Wireless Indoor Localization

The wide-scale proliferation of sensing-capable mobile devices and wireless communications has given rise to a wide range of services, including indoor localization. Indoor localization is the process of determining the position of a device within an interior space [14]. Over the last few decades, there has been an increased interest towards this subject, motivated by the wide range of real-world applications that may benefit from its use, for instance, tracking of factory floor machinery, location detection of items kept in a warehouse or finding of tagged equipment scattered throughout a plant [15]. An architecture for an accurate and low-cost IPS, scalable to any indoor space without special infrastructure restrictions is still an open challenge. Numerous indoor localization methods and systems have been proposed over the past years, making use of different wireless technologies and positioning techniques [16].

Among the most popular technologies used for indoor localization are:

- **WiFi** - WiFi access points can be also used for signal collection, allowing the construction of localization systems without the need for extra infrastructure. Additionally, most of current portable devices are WiFi enabled, which makes WiFi a good candidate for indoor localization. However, WiFi networks are usually built for communication, rather than localization, so novel algorithms are required in order to improve their localization accuracy [14].

- **Radio Frequency Identification Device (RFID)** - One of the basic types of RFID systems that is suitable for indoor localization is the Active RFID. Active RFIDs are connected to a local power source and can communicate with devices over long distances, via ID transmissions, making them appropriate for indoor localization. The main disadvantage of RFID lies in the fact that most portable devices do not support this technology. Moreover, it can not achieve sub-meter accuracy [14].

- **Ultra-Wide-Band (UWB)** - UWB technology has mainly been utilized for short-range communication systems. UWB has proven a particularly appealing technique for indoor localization due to its resistance to interference from other signals and low sensitivity to multi-path effects. The main drawbacks of this technology are the slow progress in the UWB standard development, the need for line-of-sight with the mobile receivers and the expensive cost of infrastructure[14].

- **Bluetooth** - Bluetooth based solutions for indoor localization are gaining momentum, particularly since the recent introduction of the BLE 5.1 version. This version introduced new DF features based on AoA. The emergence of this new feature allowed for the development of novel indoor localization systems. The BLE technology is of particular relevance in the context of this dissertation, so it is analysed in more detail in subsection 2.1.2.

There are several positioning techniques which can be used to derive the physical location of a device in an indoor environment. The techniques differ from each other based on which signal parameters corresponding to the wireless communication between the target and the emitter are measured. Four of the main techniques for indoor positioning are described below.

- **Received Signal Strength Indication (RSSI)** - This technique uses the signal's power to determine the distance between a receiver and a transmitter. As wireless signals propagate, their power decreases. If the initial signal strength is known, the received signal's power can be used to estimate its distance to the emitter, using the Friis free path loss equation. Although RSSI has various advantages, such as easy implementation and cheap hardware, its accuracy levels are relatively low, due to reflections, refraction, and the possibility of transmission power variations between different beacons/transmitters [10].

- **Time-of-Arrival (ToA)** - ToA is based on time-stamped signals. The distance from the target device to the emitter is directly proportional to the propagation time [15]. Knowing the signal's speed and the time-stamps at transmission and reception, a distance can be calculated. In order to obtain high accuracy levels, precise synchronization of clock frequencies between both the receiver and transmitter is required, which requires high implementation costs.

- **Time-Difference-of-Arrival (TDoA)** - Instead of using the absolute arrival time of ToA, the goal of TDoA is to identify the relative position of the mobile transmitter by computing the difference in the arrival time of several signals. It does not require synchronization

between receivers and transmitters. Instead, the transmitters are synchronized to simultaneously emit periodic signals [10]. To achieve precise localization, very high synchronization is required between transmitters. To achieve it, on-site dedicated cable installations are necessary, which makes this an expensive technique.

- **Angle-of-Arrival (AoA)** - In contrast with the previously mentioned techniques, AoA does not needs synchronization between receivers and transmitters, is not affected by variations in the RSSI, and works for an arbitrary number of transmitters. The localization accuracy improves with the number of emitters and the technique is robust to the failure of one or more devices [10]. Similarly to other techniques, AoA is affected by noise, especially multipath effects. Although it is possible to achieve high levels of accuracy, in comparison with other techniques, algorithms to compute the AoA are computationally complex [10]. As AoA is the focus of this dissertation, a more extensive characterization is given in subsection 2.1.3.

The wireless communication technologies and localization techniques can be combined to implement localization solutions. For instance, TDoA is employed in conjunction with UWB, while the use of the RSSI is frequent for WiFI or Bluetooth. The use of AoA is frequent for satellite and avionic applications, but only recently did the Bluetooth standard add DF features based on AoA, making new solutions based on this combination possible.

2.1.2 Bluetooth Low Energy

Bluetooth Low Energy (BLE) is a technology that emerged in Bluetooth version 4.0, in 2010, aiming to optimize wireless transmissions. It has been successfully used in the field of position detection, resorting to beacon communication [17]. This type of communication makes use of specific wireless devices or sensors, provided with the BLE technology, and placed in a structured environment. These devices are configured to emit radio signals so that listener devices can receive them and use their information to perform the position detection. BLE stands out for its good performance, low-energy consumption and widespread use [17].

Until recently, BLE proximity solutions and positioning systems used RSSI technique to estimate distance. However, the recent introduction of the BLE 5.1 standard included a DF feature to determine the direction of Bluetooth signal transmissions, allowing for new research possibilities in the indoor localization field. This new feature provides two different techniques for determining the direction a Bluetooth signal is coming from: the aforementioned AoA and the complementary Angle-of-Departure (AoD) [1]. Each of the methods requires that one of the two communication devices used on the localization system is equipped with an array of multiple antennas. When the AoD method is used, the antenna array is equipped in the transmitting device, while when using AoA, the array is included in the receiving device, as shown in Figure 2.1.

The DF process is performed relying on a CTE which is appended to the end of a BLE packet. A radio frequency device with an array of antennas, can sample this tone's phase at each antenna, and compute the AoA of the signal via the phase differences between antennas [9].

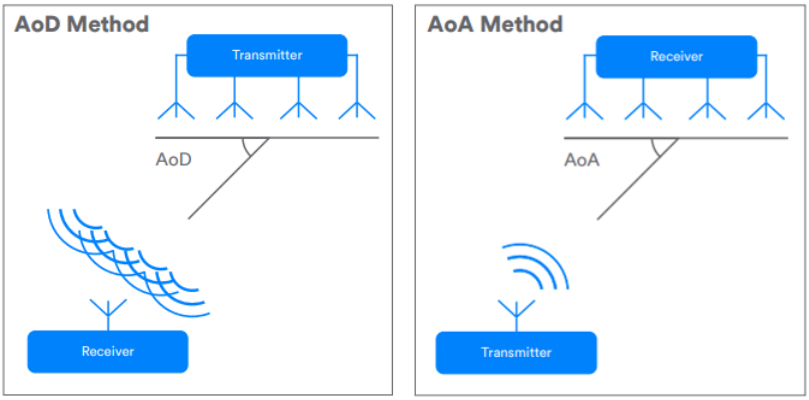


Figure 2.1: Angle of Departure and Angle of Arrival [1]

2.1.3 Position Estimation via Angle-of-Arrival

One of the many techniques used for position estimation in Indoor Positioning Systems is Angle-of-Arrival (AoA). In AoA, the position of the desired target can be determined by the intersection of several pairs of angle direction lines (also known as direction vectors), each formed by the circular radius between a beacon station and the mobile target [15]. The simplest topology which enables to compute the 2D location of a target is composed of as few as two measured angles, originating from two different reference points. In this topology, the estimated position is given by the intersection point of the two angle direction lines, as shown in Figure 2.2.

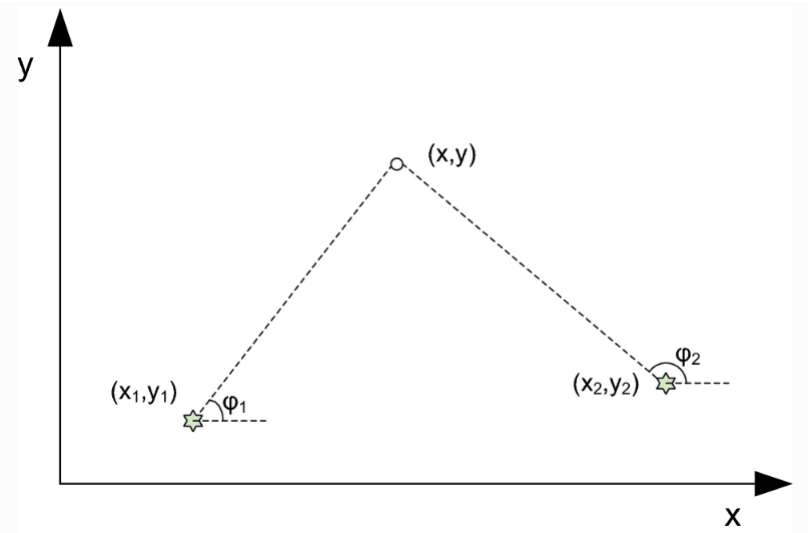


Figure 2.2: Position estimation based on AoA, with two reference points [2]

However, in real world scenarios, the measured angles are affected by errors, so one way to increase the accuracy of the estimated position is to use more reference points. This leads to more direction vectors, which makes it harder to compute a single intersection point. Instead, a candidate area is produced [11]. In this scenario, mathematical models are required to estimate the most likely position of the target inside the candidate area, as is the case of the least-squares

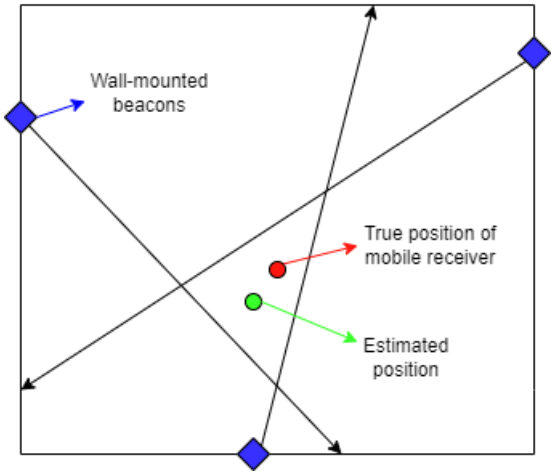


Figure 2.3: Estimation of position via received AoA and least-squares method

method, which can be used to compute the point that minimizes the distance to all lines [11]. Figure 2.3 shows an example of a 3 beacon topology, where three direction angle lines originate a candidate area for the target.

To perform the AoA estimation, usually referred to as DF, the receiver must be equipped with an antenna array [15]. A single pair of antennas is sufficient to obtain the AoA of a wavefront. By knowing the time difference of arrival between the antennas, as well as the physical distance separating the pair, simple trigonometry can be used to calculate the angle [10]. As the signal's constant frequency ω is known, the time difference t can be used to calculate the phase difference φ between adjacent antennas, as per eq. (2.1).

$$\varphi = t\omega \tag{2.1}$$

To finally calculate the AoA, given by θ, eq. (2.2) is used, where d is the physical distance between antennas and λ is the signal's wavelength.

$$\theta = \arccos\left(\frac{\varphi\lambda}{2\pi d}\right) \tag{2.2}$$

Figure 2.4 illustrates the relationship expressed by eq. (2.2). The incident angle is derived from the difference of distance, and therefore phase in function of time, that each antenna observes for the wavefront. The figure illustrates a uniform linear array with 3 elements. The precision of the calculation increases as more array elements are added. For this type of array, it is only possible to calculate the AoA from -90º to 90º (or 0º to 180º), since there is ambiguity in the incoming direction. In this work, a circular array is used, where AoA calculation is possible for a range of 0º to 360º.

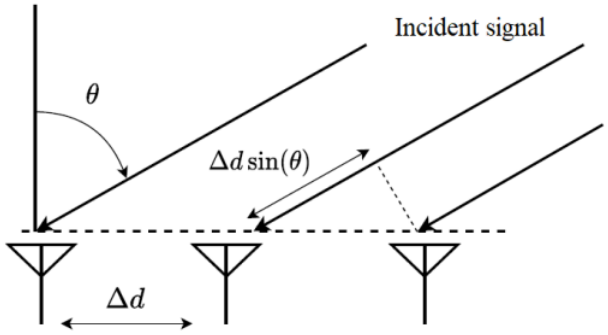


Figure 2.4: AoA based localization with antenna array

2.1.4 Machine Learning

ML [18] is a field within Artificial Intelligence (AI) that enables computer systems to learn and enhance their performance through experience, without the need for explicit programming. It has experienced rapid advancements in recent years, largely driven by notable progress in hardware technology. Consequently, it has become an extensively studied area and it is being applied in diverse scientific domains.

ML algorithms can automatically extract patterns, make predictions, and gain insights from large volumes of data, making them particularly valuable in data-driven decision-making processes. These algorithms learn from historical data to identify underlying patterns and relationships, enabling them to make accurate predictions or decisions on new, unseen data.

Regarding the different types of ML techniques, these can be classified in three main categories:

- **Supervised Learning** - In this paradigm [19], the input data is accompanied by corresponding target variables or labels, which serve as a guide for the learning process. By learning from these labeled examples, supervised learning algorithms aim to discover patterns and relationships in the data, enabling them to generalize and make accurate predictions on unseen or future data instances.

  The key objective of supervised learning is to create a mapping between input features and their corresponding target variables. This mapping can take different forms, such as regression, where the goal is to predict continuous numeric values, or classification, which involves assigning data points to predefined categories or classes. Through iterative training and optimization, supervised learning models strive to minimize the discrepancy between their predictions and the ground truth labels, ultimately improving their ability to generalize and make accurate predictions on new, unseen data.

- **Unsupervised Learning** - In contrast to supervised learning, unsupervised learning [20] algorithms do not require a ground truth associated to each set of inputs. Instead, this learning method relies on the identification of similarities and differences between data points to

group similar instances of data together, revealing underlying structures and associations. This can provide valuable insights into the natural organization of the data and help uncover previously unknown or unexpected patterns.

- **Reinforcement Learning** - Reinforcement learning [21] is a learning paradigm focused on decision-making in dynamic environments. It revolves around an agent interacting with an environment and learning to make sequential decisions to maximize a cumulative reward signal. Unlike supervised learning and unsupervised learning, reinforcement learning does not rely on labeled data or predefined patterns. Instead, it emphasizes learning through trial and error, using feedback from the environment to guide the agent's behavior.

Within the realm of supervised learning, there are two main types of algorithms: classification and regression. Classification models focus on assigning data points to predefined categories or classes, while regression models aim to establish a functional relationship between input features and the target variable, allowing for the estimation or prediction of numeric outcomes. Regression algorithms have found wide-ranging applications [22], such as stock market prediction, weather forecasting and medical diagnosis. In the context of this dissertation, these algorithms are particularly relevant. In the following subsections, a detailed description of some of the most relevant regression algorithms is performed.

2.1.4.1 K-Nearest Neighbors

One of the most straightforward ML algorithms based on supervised learning is the KNN [23]. It operates based on the principle of similarity between a new data point and the available training data and can be used in both classification and regression problems.

In the KNN algorithm, the value of k is a hyperparameter that determines the number of neighbors to consider. To classify a new data point, the algorithm calculates the distances between the new point and all the training data points based on a chosen distance metric, such as Euclidean distance. Then, in classification problems, the k nearest neighbors are selected and the majority class label of these neighbors is assigned to the new data point. In regression problems, after the selection of the k nearest neighbors, the average of the respective target values is computed to estimate the label of the new data point. Figure 2.5 is an example of how KNN can be used to perform regression. In this example, k is equal to 3, so the 3 closest neighbors of the new data point are selected, and then the average of their labels is computed and assigned to the new data point.

The KNN algorithm is simple to implement and does not make any strong assumptions about the underlying data distribution. One important consideration in k-NN is the choice of the value of k. A small value of k may lead to predictions that are more sensitive to local variations, potentially resulting in a model that overfits the training data and lacks generalization in regression tasks. On the other hand, a large value of k may yield smoother predictions but can overlook finer details and fail to capture complex patterns in the data. Selecting the optimal value of k in regression is

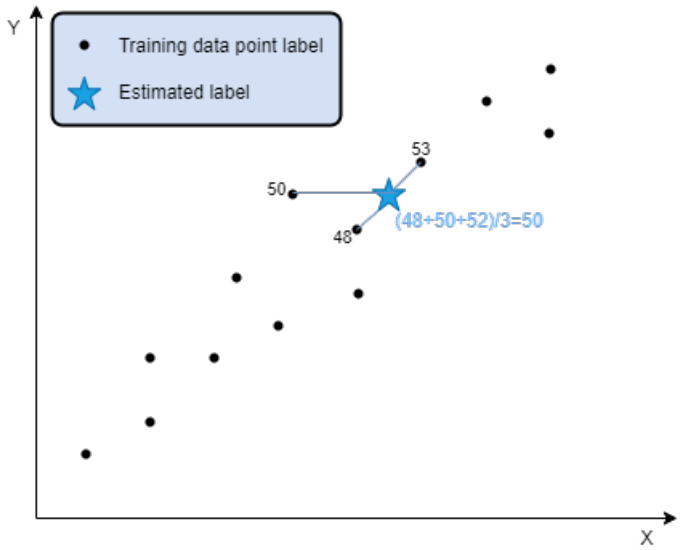


Figure 2.5: Estimation example of KNN used for regression, with k=3

often determined through cross-validation or other model selection techniques to strike a balance between capturing local patterns and avoiding overfitting the data.

An important disadvantage associated with the KNN algorithm is the computational cost, which can be expensive, especially for large datasets, as it requires calculating distances between the new point and all training points for each prediction. Various optimization techniques, such as k-d trees (k-dimensional trees), can be employed to improve its efficiency. As the datasets used in this dissertation are considerably large, binary search trees play an important role in making algorithms more efficient.

A k-d tree [24] is a binary search tree that partitions the feature space into axis-aligned hyperplanes. It facilitates efficient nearest neighbor search by dividing the space recursively into smaller regions. Each internal node of the k-d tree represents a splitting hyperplane based on a selected feature, and each leaf node represents a data point in the training set. By organizing the data in this hierarchical structure, k-d trees reduce the number of distance calculations needed during KNN prediction, making the algorithm more computationally efficient.

2.1.4.2 Decision Trees

Decision trees [25] are a particular type of ML algorithms whose fundamental concept is similar to that of k-d trees, although they are used in a slightly different context. Similarly to KNN, they are based on supervised learning and can be used for both classification and regression tasks.

A decision tree is a hierarchical structure composed of nodes and branches. The topmost node, called the root node, represents the entire dataset, and each subsequent node represents a subset of the data based on specific feature conditions. The nodes further down the tree are known as internal nodes, and the nodes at the bottom, known as leaf nodes or terminal nodes, contain the final predicted output or decision.

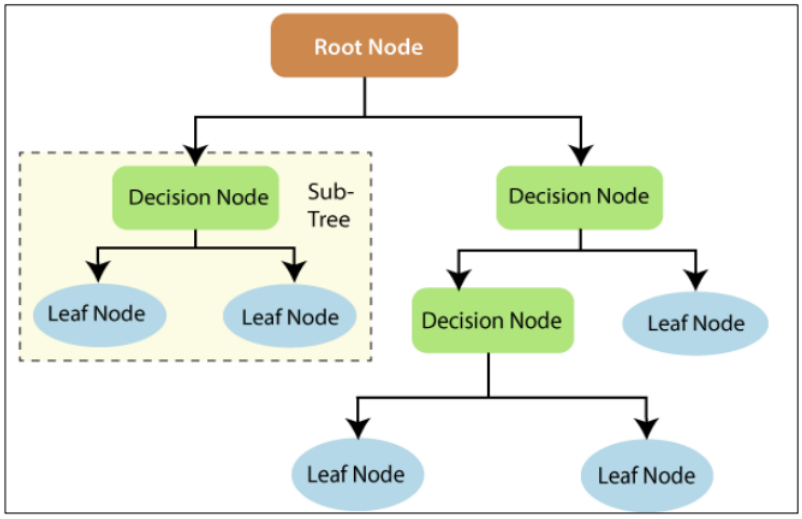


Figure 2.6: Schema of a Decision Tree [3]

The decision tree learning algorithm operates by recursively partitioning the data based on feature values. It selects the most informative feature at each internal node to split the data and create child nodes. The process continues until a stopping criterion is met, such as reaching a maximum tree depth or a minimum number of instances in a leaf node.

Regarding the process of spliting the data, different algorithms employ distinct metrics for evaluating the "optimal" solution. When it comes to regression tasks, some commonly used metrics include:

- **Mean Squared Error (MSE)** - It measures the average squared difference between the predicted and actual values within a subset. The feature and split point that result in the lowest MSE after the split are chosen.
- **Mean Absolute Error (MAE)** - It measures the average absolute difference between the predicted and actual values within a subset. The splitting criteria that minimize the MAE is selected.
- **Variance Reduction** - It quantifies the reduction in variance achieved by splitting the data based on a specific feature and split point. The feature and split point that result in the highest variance reduction are considered optimal.

Among the most relevant advantages that decision trees provide are:

- **Simple interpretation** - Decision trees provide a transparent and intuitive representation of the decision-making process. The rules inferred by the tree can be easily understood and explained in a human-readable manner.
- **Good performance with large datasets** - Standard computing resources can be used to analyse large amounts of data in a reasonable time.

- **Nonlinear relationships** - Decision trees can capture complex nonlinear relationships between input features and the target variable.

2.1.4.3 Neural Networks

Neural networks (NNs), commonly referred to as Artificial Neural Networks (ANNs) or Simulated Neural Networks (SNNs), are a subset of ML and are at the core of Deep Learning (DL) algorithms. Their name and working principle are inspired by the human brain, emulating the way biological neurons interact with each other.

ANNs are composed of several nodes, or artificial neurons, organized in sets of layers. Every ANN includes an input layer, one or more hidden layers and an output layer. All nodes are connected to each other and have an associated weight and threshold. If the output of a given node exceeds the value of the respective threshold, that node is activated and sends data to the next layer of the network.

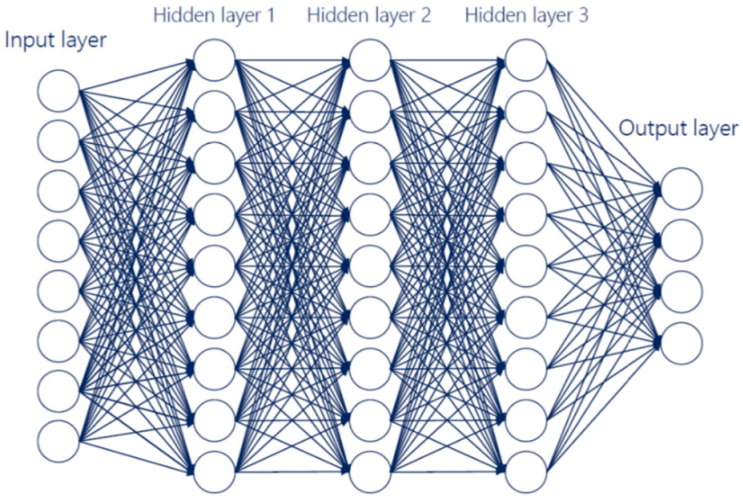


Figure 2.7: Neural Network example [4]

ANNs are a powerful tool when it comes to solving nonlinear problems. Their nature is very adaptable, as they are able to handle large amounts of noisy and incomplete datasets, making them especially useful in several problems related to engineering and sciences, where the environment can be potentially volatile and dynamic.

One of the most significant characteristics of ANNs is their capability of adapting from experience, or in other words, learning the relationship between different variables of the system and generalizing solutions for new inputs that have not been observed before. A training process is required in order to achieve this learning ability. The process of training a network consists in using a fraction of the system's input samples as a training set. The network uses the training set to adjust the weights and thresholds of each of the connections between nodes. Then, the unused fraction of the input samples serves as a test set, and is used to determine whether the obtained outputs are within an acceptable range or not. Regarding the structures and topologies used in ANNs

those can be associated with specific architectures, based on the number of layers and connection patterns between nodes. Those can be divided into two classes:

Feed-forward Networks This type of networks only allow the signals to propagate from input to output, in a strictly feed-forward direction, without feedback connections. This means that a layer is not affected by the outputs produced by subsequent layers. These type of networks have been widely employed to solve pattern classification and function approximation problems. Two of the most relevant feed-forward architectures, which are described next, are the single-layer feed-forward architecture and the multi-layer feed-forward architecture.

Single-layer feed-forward architecture This architecture is composed of an input layer and a single neural layer, which coincides with the output layer. The single-layer feed-forward architecture was at the basis of one of the earliest and most basic trainable learning machines. In Figure 2.8, an example of single-layer feed-forward architecture is given, composed of n inputs and m outputs. It is possible to see that the number of outputs is coincident with the amount of neurons figured in the network.

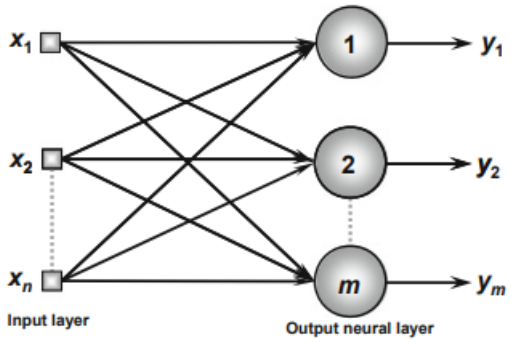


Figure 2.8: Example of a single-layer feedforward network [5]

Multi-layer feed-forward architecture This neural-network architecture is composed of one or more hidden neural layers. It is fully-connected, which means that the outputs of each layer's neurons are distributed to each neuron of the following layer, as Figure 2.9 illustrates. In this architecture, the number of hidden layers and its respective number of neurons are determined by the nature and complexity of the problem at hand, as well as the quality and quantity of available training data [5].

Feedback Networks Contrarily to feed-forward architectures, feedback networks allow for signals to propagate in different directions, and not only from input to output. In other words, an output of a given neuron can be used as feedback input for one or more neurons of precedent layers, as Figure 2.10 illustrates. This feedback characteristic provides networks with the capability

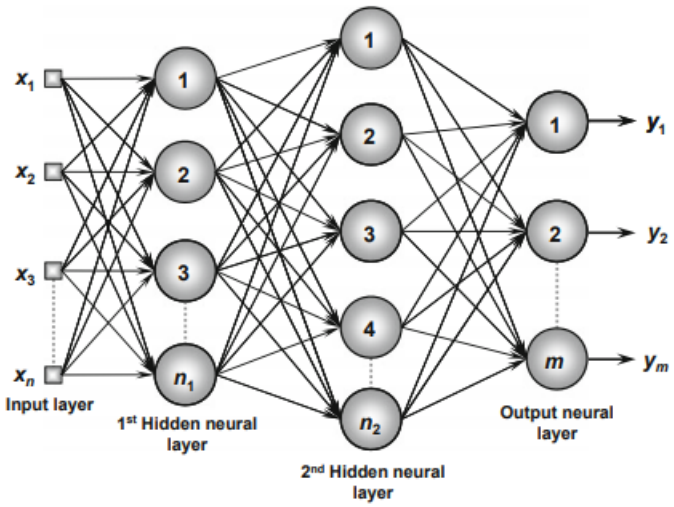


Figure 2.9: Example of a multi-layer feedforward network [5]

of dynamic information processing, allowing them to be employed on time-variant systems, such as time series prediction, system identification and optimization, process control, and so forth [5].

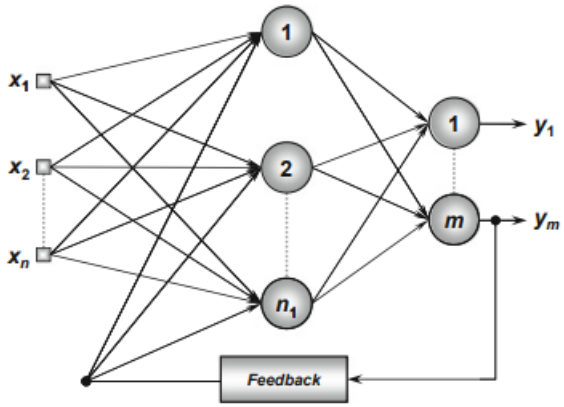


Figure 2.10: Example of a feedback network [5]

2.1.4.4 Convolutional Neural Networks

CNNs [26] are similar to traditional ANNs as they also consist of neurons that can self-optimize through learning. They are known for being particularly effective at image analysis and processing.

CNNs consist of three types of layers. These are convolutional layers, pooling layers and fully-connected layers. Once these layers are stacked (Figure 2.11), a CNN has been formed.

- **Convolutional layers** - These layers perform convolution operations, where small filters or kernels are slid over the input data. The filters extract features by computing dot products between the filter weights and corresponding data values. By applying multiple filters,

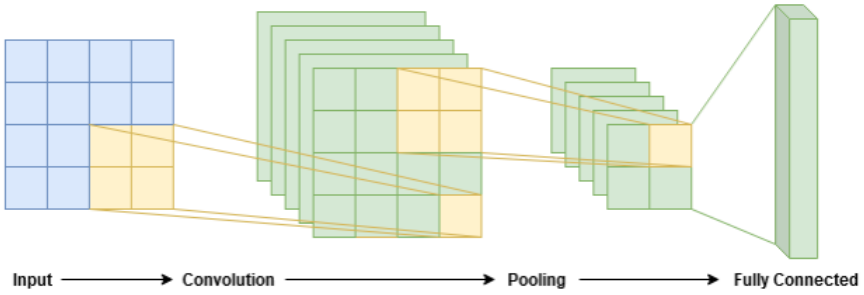


Figure 2.11: A simple CNN architecture

CNNs can capture different features at different positions within the input data. This process enables the network to learn hierarchical representations of the input.

- **Pooling Layers** - Pooling layers downsample the spatial dimensions of the feature maps obtained from the convolutional layers. Common types of pooling layers include max-pooling layers, which return the maximum value within a pooling window in each feature map, and average pooling layers, which return the average value in the pooling window. Pooling reduces spatial resolution, making the network robust to input variations and reducing the number of parameters. It also helps in extracting the most salient features.

- **Fully Connected Layers** - These layers serve the same purpose as the ones found on other types of ANNs and are placed at the end of the CNN, just before the output layer. They are fed with the output of the preceeding layers, flattened into a one dimensional vector, and make the final predicitons.

2.2 State-of-the-Art

Several topologies for indoor localization have emerged over the years, relying on different technologies, infrastructures and techniques. The approach contemplated by this dissertation is fundamentally based on AoA estimation. This section is dedicated to the analysis and discussion of existent AoA calculation approaches, not only based in ML but also other methodologies. Five specific works developed in the context of indoor localization based on AoA are reviewed below.

Multi-antenna Array-based AoA Estimation Using Bluetooth Low Energy for Indoor Positioning [27] This approach exploits a method combining non-linear recursive least squares (NRLS) and unscented Kalman filter (UKF) to improve the accuracy of AoA. The system's model consists of a BLE signal transmitter and a signal receiver with a multi-antenna array.

As pointed out by the authors, AoA-based methods are vulnerable to the effects of path loss, multipath, and noise. Additionaly, the use of multi-antenna arrays can cause errors on AoA measurements due to the receiver switching among single antennas. Under these assumptions, the

main goal of this approach is to try to overcome traditional AoA estimation algorithms in terms of accuracy, particularly the Multiple Signal Classification (MUSIC) [28] algorithm.

For context, the MUSIC algorithm is one of the most widely used algorithms in AoA calculation [10]. This algorithm exploits the orthogonality between the array vectors corresponding to the true angle of the received signals and the noise eigenvectors of the covariance matrix associated with the same signals [29].

The method proposed in this approach includes two pre-processing steps to improve the raw phase data sampled by the multi-antenna array. The idea behind AoA estimation in multi-antenna arrays is that the received signal will have a different phase at each receiving antenna. Because the distance between antennas is fixed and known to the receiver, in this approach the AoA is estimated through the formula shown in eq. (2.3), which is analogous to eq. (2.2).

$$\theta_k = \arcsin \frac{\lambda \Delta\phi_k[n]}{2\pi d_k} \tag{2.3}$$

Where θ_k is the phase difference. λ denotes the signal wavelength, $\lambda = c/f_c$, f_c is the carrier frequency, c is the speed of light, and d_k is the distance of antenna elements.

The accuracy of the AoA estimation is directly related to the quality of the phase data retrieved from the antennas. In this regard, the first pre-processing step contemplated by this approach is a non-linear recursive least squares technique based on curve fitting of the raw data on an expected model of the data to observe. Secondly, a Kalman filter attempts to mitigate the effect of various oscillator frequencies between devices, which may result in phase drift errors.

The authors conducted real-time experiments to compare the AoA estimations' accuracy among their proposed method, with and without pre-processing, and the MUSIC algorithm. The experimental results show that the average estimation error of the AoA calculated by the proposed method is reduced by 3.9 degrees when compared with MUSIC, and reduced by 7.1 degrees when compared with the method without pre-processing.

Decentralized Online Direction-of-Arrival Estimation and Tracking [30] In this work, a distributed online AoA estimation algorithm is proposed, which combines the state-of-the-art Estimation of Signal Parameters via Rotational Invariance Techniques (ESPRIT) [31] algorithm and a decentralized version of this algorithm (d-ESPRIT) proposed in a previous work, which avoids issues such as noise or bandwidth limitations.

Along with MUSIC, the ESPRIT algorithm is one of the most widely used algorithms to perform AoA estimation. The basic idea behind ESPRIT is to exploit the rotational invariance property of the received signals. It works by dividing the antenna array into two subarrays and exploiting the relationship between their signal subspaces. The algorithm estimates the AoA by solving a set of linear equations derived from the signal subspace.

Diferently from the d-ESPRIT algorithm, which is not suitable for tracking moving sources, the algorithm proposed in this work is an online approach, which updates the AoA estimations while the measurements are obtained, and thus is suitable for AoA tracking applications. To

achieve this, a modification based on a network of sub-arrays followed by a consensus algorithm is proposed. The authors introduce a temporal component to the calculation of the covariance matrix of the received signal, which involves using a *forgetting factor* to combine eigenvalues from different sub-arrays over a period of time.

For validation, a simulation setup consisting of six identically oriented sub-arrays, each being a Uniform Linear Array (ULA) with two elements, is used. The consensus algorithm is employed by separating the arrays into distinct subsets, and then an AoA is computed for each subset, with three different transmitters and 200 signal snapshots. The obtained performance is similar to that of ESPRIT but requires less computational power when compared with d-ESPRIT. While tracking two mobile sources, the MSE of the AoA estimations is 1.1°, which outperforms a similar method [32].

Deep Neural Network for Estimation of Direction of Arrival With Antenna Array [6] This work presents a novel AoA estimation approach based on Deep Neural Networks (DNNs), in which a nonlinear mapping that relates the outputs of the receiving antennas with their corresponding AoAs is learned by using a sequence of two DNNs.

As stated by the authors, in order to achieve significant accuracy improvements, a massively large amount of training data is generally required. To address this issue, the authors propose a detection network which may reduce the training set's size. The full architecture used in this approach consists of the detection network followed by the AoA estimation network, as shown in Figure 2.12.

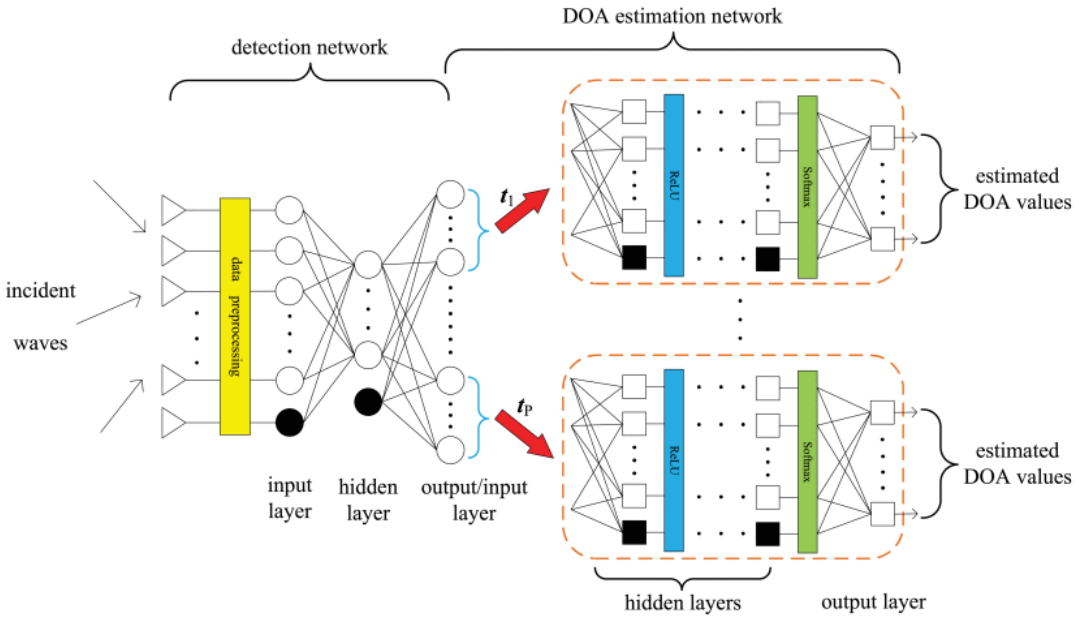


Figure 2.12: Block diagram of the networks [6]

The detection network intends to divide the antenna array's search region into different position sectors in order to detect signals coming from sources in each sector. To achieve it, the

authors used a fully-connected feedforward network, as it can handle inputs of any dimension without making assumptions about the input data distribution. The features fed to the detection network are extracted from the signal correlation matrix of the array's upper triangle.

Once the detection process is complete, the associated AoA networks are activated in order to estimate the signals' AoAs. Each DNN present in the estimation network corresponds to a different subset of antenna elements of the receptor. The formulation is thus a classification problem, where each class corresponds to a possible AoA value in the training set.

To evaluate the performance of the proposed method, several computer simulations were carried out, employing a ULA of 10 elements to receive signals from multiple sources. As no real-world measurements were conducted, a synthetic dataset was used for training and testing, where the AoA varied between -60º and 60º, with a step of 1º. The obtained angular accuracy was 1º. The method yields a RMSE around two times that of MUSIC. Nevertheless, it keeps the AoA inference time nearly constant at 15 ms, regardless of the volume of input data, contrary to MUSIC, which inference time increases linearly with the signal length.

Angle of Arrival Estimation in Indoor Environment Using Machine Learning [7] In this approach, the authors propose an AoA estimation framework for a multipath radio environment based on a CNN. This method is employed with the aim of learning the mapping between the eigenvectors of the spatial covariance matrix of received array signals and angles of arrival. The solutions' main motivation is to outperform the state-of-the-art MUSIC algorithm in regard to computational complexity.

This topology resorts to a ULA of antennas at the receiver, which collects Orthogonal Frequency-division Multiplexing (OFDM) multipath signals in an indoor environment. OFDM systems are characterized by data transmission over multiple sub-carriers, each with a different frequency.

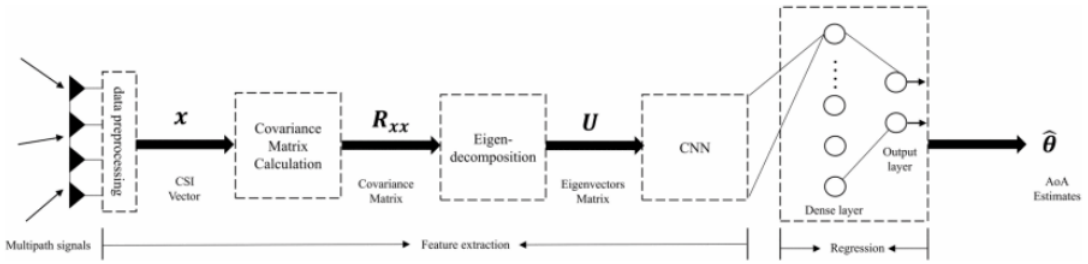


Figure 2.13: The block diagram of the proposed CNN-based AOA estimation [7]

The proposed CNN-based AoA estimation approach is presented as a block diagram in Figure 2.13. The first event is the reception of the multipath signal by the ULA. The signal is then processed into a Channel State Information (CSI) vector. The CSI comprises the complex attenuation and phase of all sub-carriers from all antennas and captures the channel variations caused by the multipath effect. It is represented by a 3D matrix of size $N \times N_{SC} \times N_S$, where N is the number

of antennas, N_{SC} is the number of sub-carriers and N_S is the total number of samples of channel estimate in the frequency domain. The ULA processes the multipath signal into a 1D vector representation of the CSI, defined by equation 2.4.

$$\vec{x} = [\vec{x}_1 \vec{x}_2 \dots \vec{x}_{Ns}] \tag{2.4}$$

Then, $\vec{x}$ is used to compute the covariance matrix $\hat{\mathbf{R}}_{xx}$, as represented in equation eq. (2.5).

$$\hat{\mathbf{R}}_{xx} = \frac{1}{N_s} \sum_{i=1}^{N_s} \vec{x}_i \vec{x}_i^{\rightarrow H} \tag{2.5}$$

Where $^{\rightarrow H}$ denotes the hermitian operation. The covariance matrix extracts features related to AoA, compresses the data and suppresses additive white gaussian noise.

The following event in the block diagram marks a difference from previous proposed approaches, which directly feed the $\mathbf{R}_{xx}$ to the ML model. Instead, in this work, the authors propose a further preprocessing step using the eigen-decomposition on the $\hat{\mathbf{R}}_{xx}$. The result is an eigenvectors matrix, which serves as input to a convolutional neural network-based regression function, which performs the AoA estimations. Figure 2.14 shows the proposed CNN architecture, consisting of an input layer, four 2D convolutional layers, 2 average pooling layers, a 32 neuron dense layer and an output layer.

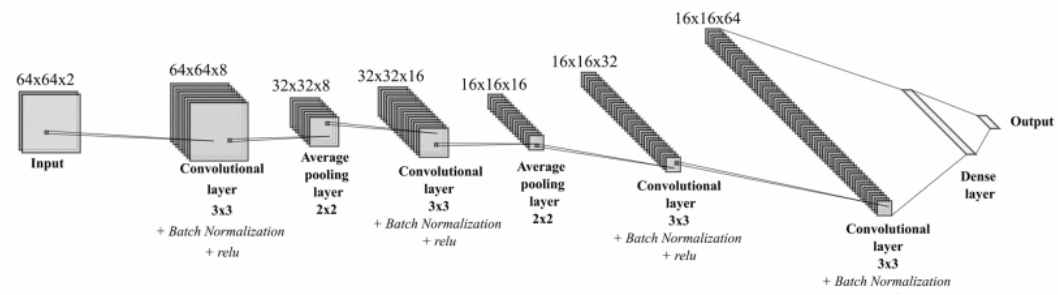


Figure 2.14: The proposed CNN architecture [7]

The simulations are performed on a ULA with four elements, two signal paths, and 16 subcarriers. Several levels of SNR are evaluated, but for 20 dB, the approach achieves a RMSE of only 2.4°, versus the 17.4° error achieved by MUSIC.

Two-Dimensional DOA Estimation via Deep Ensemble Learning [8] This approach proposes the use of a ML technique called ensemble learning, which uses multiple base learners to increase the predictive accuracy, to estimate both the azimuth and elevation AoAs. Contrary to the previously reviewed works, where the models target a ULA, this work uses a Uniform Circular Array (UCA) to receive the spatial signals.

The choice of this method is based on the theory that, according to the authors, the nonlinear relationship between the input data and the estimated output values is too complex to be captured by a single CNN with a limited dataset, leading to significant estimation errors. For this reason, the

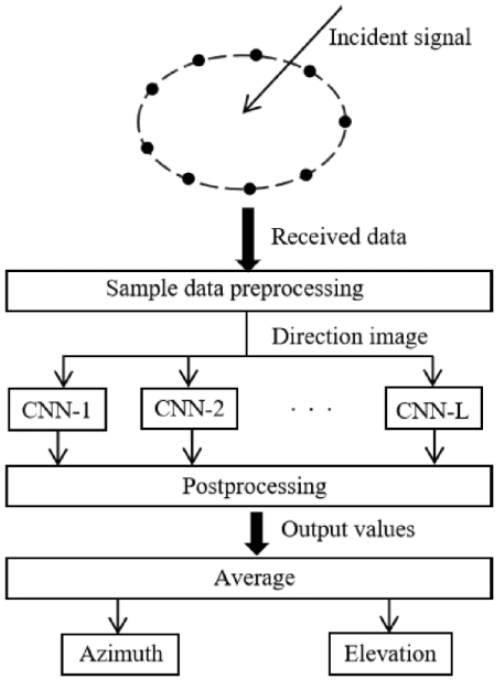


Figure 2.15: Block diagram of the proposed deep ensemble model [8]

proposed ensemble learning method aims to ensure that the most reliable and accurate prediction is made for AoA estimation.

Ensemble learning is a technique used to improve the performance of NNs by training multiple models independently and combining their outputs in a specific manner. This results in reduced prediction errors when compared with single network models. Methods of ensemble learning can be classified according to the component that is varied between networks, such as the base learner, the training data, and how predictions are merged.

In this approach, five CNNs with different number of convolutinal layers, different batch sizes, different learning rates and different initial random weights are trained on the same dataset. The configuration parameters of the five networks are given in Table 2.1.

Model	Architecture	Batch size	Learning rate	Standard deviation of weight
CNN-1	$128-128-128-128-128$ $(2-2-2-2-2)$	256	0.001	0.05
CNN-2	$128-128-128-128-128$ $(2-2-2-2-2)$	128	0.01	0.1
CNN-3	$128-128-128$ $(3-3-3)$	200	0.01	0.1
CNN-4	$128-128-128$ $(3-3-3)$	200	0.001	0.05
CNN-5	$128-128-128$ $(2-3-4)$	200	0.001	0.05

Table 2.1: Parameters of the five CNNs evaluated in [8]

For the training process, each CNN is trained independently with the same set of training data. During the testing, a new test dataset is applied to all CNNs simultaneously and the final prediction is calculated by taking the average of all the results.

For a UCA of nine elements and inputs composed of 1024 snapshots, the proposed ensemble model achieves a RMSE of 0.59º, taking only 3.3 seconds to process a test set of 10 thousand samples, far more efficient than MUSIC, which takes about a minute to process 100 samples.

Summary Table The key points related to the approaches described in this section are summarized in Table 2.2.

Work	Summary of Proposed Method	Algorithm	Array	RMSE(º)
[27]	NLRLS[a] + Kalman filter	$\theta = f(\Delta\phi)$	ULA-6	13
[30]	Sub-arrays + consensus algorithm	d-ESPRIT	ULA-2	1.1
[6]	2 DNNs[b] (signal detect + sub-ranges)	Classification	ULA-10	0.25
[7]	CNN[b] (6 layers, 2 outputs)	Regression	ULA-4	2.4
[8]	CNN Ensemble[b] (azimuth + elevation)	Regression	UCA-9	0.6

[a] Non-linear recursive least squares
[b] Co-variance matrix as feature

Table 2.2: Summary of the AoA estimation approaches

2.3 Conclusions and Closing Remarks

This chapter gave a brief presentation of the most used indoor localization technologies and techniques, with special focus on those related to this dissertation. Furthermore, it introduced the main concepts of ML and focused the most relevant algorithms in the context of this work. Lastly, an analysis of five works related to AoA estimation was performed.

Through the study of existent indoor localization methodologies, it was possible to conclude that the most promising topology in terms of accuracy, affordability and scalability is achieved by combining a Bluetooth infrastructure with an AoA position estimation technique. This was made possible by the recent introduction of the BLE 5.1 standard, which included a DF feature. Approaches relying on this topology have already outperformed many existent solutions based on different infrastructures and position estimation methods. However, one of the most determining factors in achieving high levels of accuracy concerns the algorithms used for the AoA estimation. In that regard, the performed study allowed to conclude that a large number of approaches still rely on state-of-the art algorithms such as the MUSIC. Although this algorithm is still the most widely-used for AoA estimation, it has several limitations, for instance, performance degradation in the lack of line-of-sight, and the assumption of ideal orthogonality between the sample and noise subspaces, which is false in real world conditions, due to multipath effects. Additionally,

the computational complexity of these algorithm is a major factor in allowing for faster response. All those factors contribute for an increasing demand for alternative methods of estimating AoA. ML-based solutions appear in this context. Although there are some recent approaches that are attempting to evaluate its viability, there is still significant space for improvement as the solutions have not been largely exploited so far. Concluding, the application of ML techniques is the way to follow when it comes to achieve better outcomes in indoor localization.

Chapter 3

Proposed Approach

The main goal of this dissertation was to study and develop ML models to perform AoA estimation based on phase difference data collected as part of the development of an indoor tracking system.

As mentioned in Chapter 1, this indoor tracking system is based on a topology that relies on an antenna array mounted on a moving platform, which recieves packets transmitted by Bluetooth beacons. The design of the receiver board was evaluated in a previous dissertation [33].

Prior to the development of this dissertation, several steps had already been taken with the aim of evaluating this topology. The first step was the implementation of a Java based simulator [11], which relies on AoA data gathered in real-world experiments performed with a commercial antenna array and a BLE transmitter. The evaluation of the AoA inference using the commercial board was performed in a previous dissertation [34], as well as the study of position tracking via simulation, using the data collected by the commercial board [35].

The second step was the design and construction of a circular antenna array with 8 antennas [9], which was used in two experimental evaluations for phase data collection: the first experiment was performed in an anechoic chamber and the second one in an outdoor area[1]. The obtained phase difference data was used to perform AoA estimation resorting to a heuristic algorithm [10]. The results yielded by this algorithm are the key comparison term for the ML algorithms developed in this dissertation.

In this chapter, the proposed approach to the development of the ML models for AoA estimation is detailed. In Section 3.1, the datasets used to feed the models are described. In Section 3.2, an overview of the software tools used in the models' development is given. Section 3.3 details the different ML techniques used. Lastly, in Section 3.4, the integration of the algorithms with the simulation framework is detailed.

[1] https://ieee-dataport.org/documents/dataset-phase-samples-using-8-element-uniform-circular-antenna-array-and-bluetooth-low

3.1 Phase Difference Datasets for UCA with 8 Elements

As mentioned, a real-world experiment performed in an outdoor area has produced a dataset of phase data, which is one of the datasets used to test the ML models. Figure 3.1 shows the aerial view of the area where this experiment was conducted. In this experiment, 4 Bluetooth transmitters were placed in each corner of the area outlined in red and a receiver was placed in 21 different positions within that area, receiving a total of 600 packets in each of these positions.

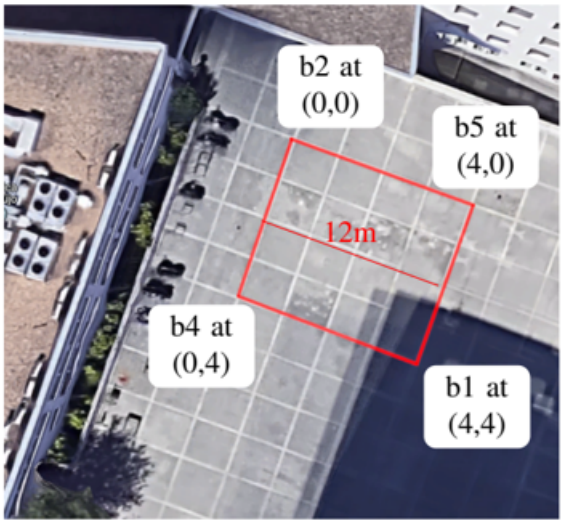


Figure 3.1: Aerial view of the area where the real-world experiment was performed [9]

However, this dataset is not sufficient to perform an extensive evaluation of the developed algorithms. For that reason, this work resorts to a dataset generator[2] which was developed in an earlier stage of the tracking system evaluation, and is able to generate phase difference profiles associated to a given AoA, with arbitrary levels of noise. In the next subsection, the concept of phase differences is explained. In subsection 3.1.2, the format of the data in the datasets is described.

3.1.1 Phase Difference Data

As previously mentioned, the AoA estimations performed in this work are based on phase difference data. The concept of phase difference is related to the reception of transmitted signals by each of the antennas that make up the circular antenna array, which is schematized in Figure 3.2.

When the antennas are receiving transmissions, each received packet has an associated CTE. The CTE is a sinusoidal wave which is suffixed at the end of every DF enabled BLE packet, and whose phase value is read on one antenna at a time. Knowing this value for each antenna, it is possible to calculate a phase difference value for each pair of adjacent antennas. To prevent phase ambiguities, neighbour antennas must be placed at a distance no lesser than half the wavelength. As the antenna array used in this topology consists of eight antennas, there is a total of eight

[2] https://codeocean.com/capsule/0771129/tree/v1

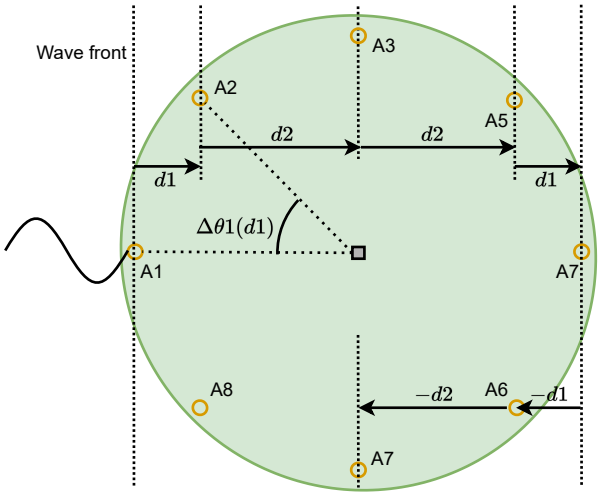


Figure 3.2: Diagram of the circular antenna array [10]

possible adjacent antenna pairs, resulting in eight phase differences. An example of four phase difference profiles is represented in Figure 3.3. Each phase profile is obtained from a different incident angle of the wavefront. These phase difference profile examples were computed using phase values per antenna which were generated by the mentioned dataset generator. The data was generated without noise. The phase profile is a sinusoid because the antenna array we consider is circular, and all the antennas are evenly spaced along its perimeter. This means that there are always pairs of phase difference values with the same absolute value but opposite sign.

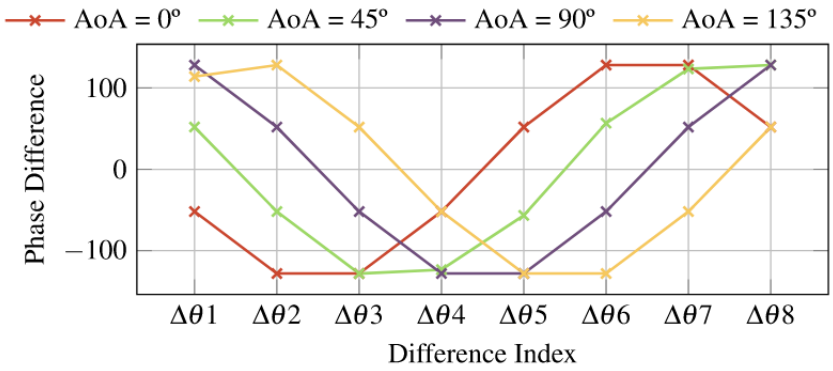


Figure 3.3: Profile of phase difference values for all adjacent antenna pairs, as a function of four different AoAs [10]

3.1.2 Dataset Features and Ground-Truth

In the ML problems contemplated in this dissertation, the phase difference values are the input features and the associated incident angle is the ground truth. The goal is to train the ML models so that when they are fed with a set of phase differences, they are able to estimate an AoA value as close as possible to the ground truth.

The dataset generator is implemented as an Octave script and allows to generate datasets in *.csv* format, with the desired number of data points and an arbitrary value of noise. Essentially, a single data point generated by the script is a one dimensional tensor with nine columns, where the first eight are the features and constitute the eight phase differences from each adjacent antennas pair, and the last one is the label, or ground truth, i.e, the AoA matching the generated phase difference profile. Figure 3.4 shows a table with 6 data points (the first 3 data points and the last 3 data points) belonging to one of the datasets generated for this work.

Features (Phase Differences)								Ground Truth (Real AoA)
-86.853	-110.882	-116.185	-71.949	70.647	108.784	134.512	20.126	1
-67.615	-110.531	-129.516	-74.717	53.370	110.457	131.405	74.798	1
-46.175	-122.869	-125.588	-55.354	63.898	130.658	106.961	56.960	1
...								
-71.480	-145.210	-91.638	-69.367	56.402	138.036	129.482	34.401	360
-70.340	-99.841	-154.706	-46.791	54.313	94.424	146.245	-51.375	360
-40.159	-140.331	-146.524	-30.960	61.196	99.762	153.947	51.676	360

Figure 3.4: 6 data points belonging to one of the generated datasets

Additionaly, the script also allows to generate a larger value of phase differences for each data point. This is done by considering not only a sampling of the entire perimeter of the receiver board (shown in Figure 3.2), but multiple samplings of each antenna. This is possible when the CTE is configured for a lower duration (up to a maximum of 320us). Assuming that the sample and switching times are the same (4us for our case study), then multiple periods of the phase profile wave (examples shown in Figure 3.3), can be extracted for a single CTE. This will increase data redundancy and potentially increase resiliance to noise effects. This is studied in Chapter 4.

In order to train and test the ML models several datasets were generated, with 8, 16 and 32 features. The noise induced in the datasets follows a Gaussian distribution with mean 0º and a standard deviation ranging from 10º to 90º, with a step of 10º. Figure 3.5 shows an example of four different phase difference profiles obtained with different standard deviation values of the Gaussian distribution associated with the noise level. It can be seen that with increasing noise the profiles lose their sinusoidal characteristic.

A total of 27 datasets were generated, each with 72000 data points (200 datapoints dedicated to all possible angle integer values [0º..360º]). The real-world dataset follows the same format as the generated datasets and has 12600 datapoints, each consisting of 32 phase differences plus a ground truth.

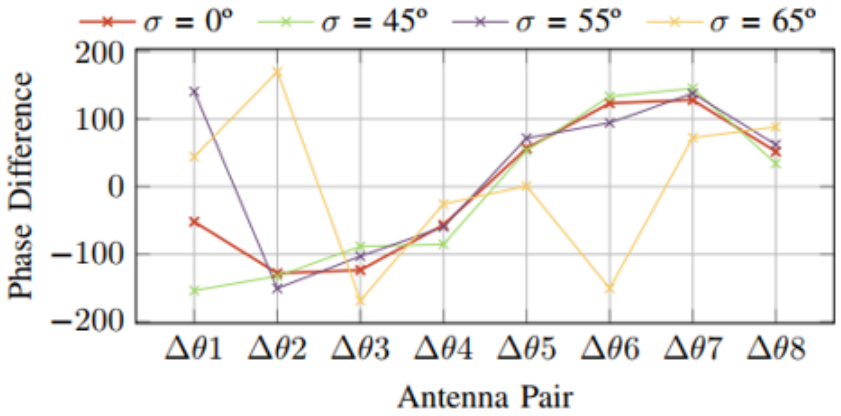


Figure 3.5: Four phase difference profiles generated with different standard deviations [9]

3.2 Machine Learning Framework

Since, as explained in Chapter 1, the developed ML algorithms were meant to be integrated in the existent simulation framework, which is coded in Java, to facilitate the integration, the ML models were also written in Java. As Java is not the most common programming language to develop ML algorithms, an appropriate ML framework compatible with Java was required. In this context the chosen framework to support the development of the work was Eclipse DeepLearning4J (DL4J).

Eclipse DL4J [36] is an open-source, Java Virtual Machine (JVM)-based toolkit for building, training, and deploying NNs. It was created to serve the Java and Scala communities and is user-friendly and stable, including multiple libraries, each with its own set of features:

- **DataVec** - DataVec is a library used to preprocess data before feeding it into NNs. The primary purpose of DataVec is to transform raw data into vector-like data, appropriate for training a NN. Current supported formats include raw text, CSV, images, and more. Additionally, DataVec can be used for feature engineering, data cleaning, scaling, and normalization.

- **Deeplearning4J** - DL4J is the primary tool used for creating NNs. It provides the components to configure, train, and evaluate NNs and supports a wide range of NN architectures, including feedforward and convolutional networks. It performs both supervised and unsupervised learning, enabling tasks like classification, regression, and generative modeling.

- **ND4J** - The numerical processing library for DL4J is called ND4J. The key features of ND4J include linear algebra / signal processing functions, multiplatform functionality with GPU support, and versatile n-dimensional array object. This Java API is comparable to that of Numpy and scikit-learn in Python. It provides the functionality for the loss funcitons, optimization algorithms, and updaters.

- **Arbiter** - Arbiter is a library used for tuning hyperparameters of NNs. Due to the size of the hyperparameter space, this is frequently a computationally demanding challenge. Arbiter

currently supports two hyperparameter optimization methods: GridSearch and RandomSearch. These optimization methods search over different combinations of hyperparameters, like the learning rate and the batch size, in order to identify the optimal combination.

3.3 AoA Estimation Models

Prior to selecting the ML models to be employed in this dissertation, the problem at hand had to be analysed in order to select the most suitable algorithms. As mentioned in Chapter 2, when we want to establish a functional relationship between input features and a ground truth, in order to estimate numerical outcomes, we are facing a regression problem. To address this type of problem, ML models for regression are required. Chapter 2 already described some of the most relevant ML algorithms to perform regression. In this section, the implementation of each of the models used in this work is detailed. Additionaly, subsection 3.3.1 gives an overview of the data preprocessing performed prior to its use.

3.3.1 Data Preprocessing

As already explained in Chapter 2, there are two main steps related to the use of a ML model: the training and the testing. The goal of the training is to make the model learn and identify patterns, relationships, and correlations within the training data that enable it to make accurate predictions on unseen test data.

This way, all the datasets used in this work have to be divided into two different datasets: one for training the models and another one to test them. For this purpose, before developing the models, an algorithm for pre-processing the data had to be developed.

In addition to splitting the datasets, there is another important preprocessing operation: data normalization. The first reason why data normalization is important is because in the datasets, the range of numerical values in the features is different from that of the labels. While the value of each of the phase differences varies between -180º and 180º, the associated label value ranges from 0º to 360º. To improve the convergence of the ML algorithms, as well as to create more reliable models, it is important that all the data is in the same range.

Based on this information, the pre-processing algorithm performs three main tasks: first, for a given dataset, it normalizes the whole data to values between 0 and 1. Secondly, it shuffles the data points, since these are originally arranged in ascending order of the ground truth value, e.g., 200 samples of 1º, followed by 200 samples of 2º, etc. In this way, it is ensured that the models do not learn patterns based in the original order of the data points. In fact, the shuffling is more important when using a batch learning method, since we want to sample batches of points that correspond to different ground truths while performing learning. Lastly, it divides the dataset into two datasets for training and testing by means of a chosen percentage, e.g., 60% for training and 40% for test.

3.3.2 K-Nearest Neighbors

One of the chosen ML algorithms to use was KNN. The motivation behind this choice has to do with the concept on which KNN is based: similar data points tend to have similar output values. This concept makes sense in the context of the datasets used in this work, as the similarity between phase difference values of the training data and each of the test instances can be exploited in order to generate an accurate estimation of the AoA.

The operation of the KNN algorithm is based on the computation of the distances between the features of a given training set and the features of a single test instance. Depending on the chosen numerical value for the K parameter, the K nearest datapoints to the test instance are selected and the average value of their labels is calculated. This average value is then assigned to the test instance label.

The main problem related to the development of a KNN model in this work is that the datasets are considerably large. The traditional KNN algorithm performs nearest neighbors search linearly, i.e, the algorithm calculates the distance between the test instance and each of the training samples. As there are a large amount of training samples, too many distances have to be computed, and that results in a high computational cost (i.e., training and inference times), which is the opposite of what is intended.

A good alternative to the linear search is the k-d tree (K-Dimensional Tree). A k-d tree is a data structure that is commonly used for efficient nearest neighbor search in K-dimensional spaces. It is a binary tree where each node represents a training data point in the K-dimensional space and partitions the space into two regions. The k-d tree was used in the development of the KNN model, aiming to overcome the high computational cost of the traditional linear search.

The following steps detail the development of the model:

- **Creation of the k-d tree** - The first step was to build a k-d tree using all the data points from the training set. The construction of the k-d tree is a recursive process that proceeds as follows:

    1. Firstly, the first data point of the training set is defined as the root node of the tree.
    2. Secondly, a splitting axis is chosen considering the current number of levels (depth) of the tree. The root node represents a depth of 0, so the chosen splitting axis for this node is defined as the modulus operation between the depth and the feature vector length (8), resulting in a value of 0. This means that the criterion for the creation of the left and right subtrees will be based on the value of the first feature (0 index) of all the remaining feature vectors of the dataset.
    3. After the splitting axis is chosen, the dataset is sorted in ascending order along the determined feature index and after that the median feature index from the sorted dataset is retrieved.

4. Then, two sub-datasets are created by splitting the sorted dataset at the median index. The left dataset contains the instances with indexes from 0 to medianIndex - 1, and the right dataset contains instances from medianIndex + 1 to the end.
5. Finally, the method recursively calls itself to build the left and right subtrees. It passes the left and right sub-datasets along with an incremented depth value. The recursive process is stopped when an empty sub-dataset is passed, which means that a leaf node was reached.

When the process is completed, we have a binary tree where each node contains the value of the median features that were retrieved in each recursive call of the method.

- **Nearest neighbors search** - The second step is the development of a nearest neighbors search method which receives the features of a test data point. This is also a recursive method which traverses the k-d tree in search of the the most similar nodes, and is organized in the following steps:

  1. Firstly, the method computes the euclidean distance between the features of the test data point and the features of the current node being examined.
  2. Then, the method checks if the computed distance is smaller than the distance of the farthest neighbor in a list which stores the feature values of the k nearest neighbors. When the method is first called, that list is empty, so the current node features are simply added to the list.
  3. After updating the list, the method determines the splitting axis based on the depth of the node. It calculates the difference between the test data point's feature value on the axis and the node's feature value on the same axis.
  4. The method recursively calls itself to search the left or right subtree based on the sign of the calculated difference. If the difference is less than 0, which indicates that the test data point's feature is smaller than the node's feature on the splitting axis, the left subtree is explored; otherwise, the right subtree is explored.
  5. The method continues this recursive process, traversing the tree based on the splitting conditions and updating the nearest neighbors' list until it has examined all relevant nodes.

  By the end of the recursive process, the list contains the k nearest neighbors to the test data point.

- **AoA estimation** - Lastly, the AoA estimation is performed by calculating the average value of the labels associated with the k feature vectors present in the list of k nearest neighbors. The obtained value is returned has the predicted AoA value for the test data point.

In Figure 3.6, an example scheme of the decision process at the different levels of the tree is given. For simplicity, each node has only 3 features. The blue nodes represent the "path" followed

by the test data point until it reaches the leaf node, which is the one whose features are the most similar to the test features.

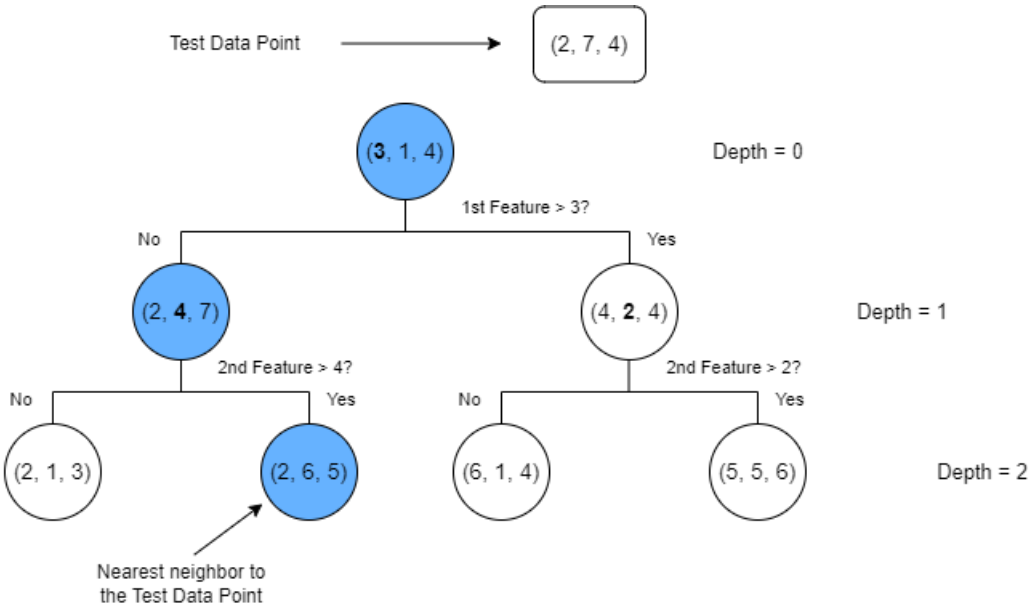


Figure 3.6: Example scheme of the nearest neighbor search

3.3.3 Neural Networks

The state-of-the-art research performed in the initial phase of this dissertation allowed to conclude that most of the approaches that evaluated ML based methods to perform AoA estimation have resorted to NNs. As these approaches yielded significant improvements in the accuracy and efficiency of the estimations, the use of NN models in this work emerged as one of the most appealing approaches to follow. None of the approaches that resorted to NNs used phase difference values as input features, making this a novel approach in AoA estimation.

For these reasons, the development of NN models was chosen as one of the approaches to follow in this dissertation. However, as the research about NNs performed in Chapter 2 allowed to conclude, NNs are an extensive subset of ML and there are several different network classes and architectures, so it is important to understand which NN models best fit our problem.

Three different NN models were developed. As a starting point, a very simple NN model based on the Multi-Layer Perceptron (MLP) architecture was chosen. Our first intention was to understand if NNs could capture the relation between phase difference data and AoA with sufficiently good quality. Therefore, the reason for choosing the MLP was the simplicity and versatility of this architecture, together with the fact that MLPs can handle large datasets and are capable of learning complex nonlinear relationships between features and labels.

The second model constitutes a more complex version of the first model. It is based on the exactly same architecture as the first model but contains a larger number of hidden layers. The

purpose of this model development was to evaluate the influence of a larger number of hidden layers in the quality of the obtained estimations.

The last model is a CNN. This network was developed with the main objective of achieving better robustness to noise, compared with the previous models. CNNs are commonly used for image-related tasks, but they can also be applied to regression problems with tabular data, being able to capture relevant patterns and spatial relationships between data features.

The models' development was supported by the DL4J framework, which has multiple libraries for NN construction. The following subsections describe each of the NN models developed.

3.3.3.1 First Model

The first model is a simple feedforward NN with fully connected layers. This model is based on the MultiLayerNetwork class provided by the DL4J library, which allows for flexible configuration of the network architecture, as well as definition of the model parameters.

The network structure is composed of three layers: one input layer with the same number of nodes as the input vector dimension (8), a single hidden layer with 256 nodes, and an output layer with a single output node, representing the estimated AoA. The structure of this model is summarized in Figure 3.7.

```
=================================================================
LayerName (LayerType)   nIn,nOut   TotalParams   ParamsShape
=================================================================
layer0 (DenseLayer)     32,256     8 448         W:{32,256}, b:{1,256}
layer1 (OutputLayer)    256,1      257           W:{256,1}, b:{1,1}
-----------------------------------------------------------------
            Total Parameters:  8 705
        Trainable Parameters:  8 705
           Frozen Parameters:  0
=================================================================
```

Figure 3.7: Summary of the first model structure

After defining the network architecture, the next step is to choose the most appropriate values for the network hyper-parameters, such as:

- **Activation function for the hidden layer** - Activation functions are a key parameter in introducing non-linearity in the learning process of NNs. Without them, a NN would simply be performing linear transformations and would not be able to learn complex patterns and relationships in the data. There are several popular activation functions which could be applied to our problem, such as Tanh, Softmax and ReLU. All those functions were tried on the model, with the ReLU producing slighly more accurate results. For that reason, the ReLU was chosen as the activation function for the hidden layer.

- **Activation function for the output layer** - In contrast to the hidden layer, in the ouput layer it is not desirable to perform non-linear transformations to the data, since we are predicting a single continuous value. For that reason, the chosen activation function for the output layer

was the linear activation function, which allows the network to directly output a continuous value without any non-linear transformation.

- **Weight initialization** - The weight initialization parameter serves to set the initial values of the weights of the network's connections, which is important because if all the weights are initialized with the same value, that results in symmetric neurons which compute the same output, making it difficult for the network to learn distinct features. Several popular weigh initialization methods were tried, such as Random initialization, He, and Xavier, with the latter producing the best results, and therefore being chosen for this model.

- **Weight updater** - Also known as optimizer, the weight updater is responsible for adjusting the weights of the network during the training process, in a way that minimizes the error between the predicted outputs and the ground truth values. The updaters that best fit our model are the Adam and the Nesterovs. The updaters also perform a dynamic adjustment of the learning rate during the training process, so it is important to select the best initial learning rate depending on the updater used. For this model, the Nesterovs optimizer were chosen.

- **Loss function** - The loss function is used to evaluate the offset between the output values and the ground truth, throughout the learning process. It is important because it serves as a guide for the optimizer to adjust the network's parameters during the training. For regression tasks, one of the most commonly used loss functions is the MSE and so this was the chosen loss function for this model.

- **Initial learning rate** - This hyperparameter defines the value of the learning rate at the beginning of the training process. In our model, the initial learning rate value that best adapts to the Nesterovs updater is 0.1.

- **Batch size** - The batch size determines the number of training samples to propagate through the network before updating the network's connections weights. It is important to define a balance between the batch size and the total number of training samples in order to avoid problems such as overfitting and slow convergence. Through a process of experimentation and evaluation of different batch sizes, the most accurate estimations were produced for a batch size of 600.

- **Number of epochs** - The number of epochs determines the number of times that the entire dataset is propagated through the network. By going through multiple epochs, the network has the opportunity to learn more from the data, potentially improving its performance. In this case, 20 epochs were found to be sufficient for the model to converge to the best possible value.

Table 3.1 summarizes the choosen hyper-parameters for this model.

Hyper-parameters	Values
Activation Function (Hidden Layer)	ReLU
Activation Function (Output Layer)	Identity
Weight initialization	XAVIER
Weight updater	Nesterovs
Loss Function	MSE
Initial learning rate	0.1
Batch size	600
Number of epochs	20

Table 3.1: Hyper-parameters of the first model

3.3.3.2 Second Model

The second model does not need an extensive description as it uses the exact same hyper-parameters as the first model. As previously mentioned, the difference between the models lies in their architecture. While the first model has a single hidden layer, this model has three hidden layers. The input and output layers do not change and all the layers are also fully connected.

With regard to the number of nodes of each hidden layer, after several experiments, it was concluded that a decreasing pattern for the number of neurons in successive hidden layers produced the most accurate results. Based on this assumption, the number of nodes chosen for each hidden layer was: 256 nodes for the first layer, 128 for the second layer and 64 for the third layer.

```
=================================================================
LayerName (LayerType)   nIn,nOut   TotalParams   ParamsShape
=================================================================
layer0 (DenseLayer)     32,256     8 448        W:{32,256}, b:{1,256}
layer1 (DenseLayer)     256,128    32 896       W:{256,128}, b:{1,128}
layer2 (DenseLayer)     128,64     8 256        W:{128,64}, b:{1,64}
layer3 (OutputLayer)    64,1       65           W:{64,1}, b:{1,1}
-----------------------------------------------------------------
            Total Parameters:  49 665
        Trainable Parameters:  49 665
           Frozen Parameters:  0
=================================================================
```

Figure 3.8: Summary of the second model structure

3.3.3.3 Third Model

The third and last NN model is a CNN. It is composed of three convolutional layers followed by the same three hidden layer structure applied to the second model. In this model, the Adam optimizer combined with a learning rate of 0.001 produced better results than the Nesterovs optimizer. The other hyper-parameters remained the same.

The network starts with a one dimensional convolutional layer defined by the "Convolution1DLayer" class of the DL4J library, followed by a pooling layer ("Subsampling1DLayer") for

most-important data features extraction. Subsequently, two more convolutional layer pairs (Convolutional1DLayer + Subsampling1DLayer) complete the convolutional structure of this NN. Finally, the same layer structure developed in the second model is appended to the last convolutional layer.

```
==========================================================================
LayerName (LayerType)         nIn,nOut   TotalParams   ParamsShape
==========================================================================
layer0 (Convolution1DLayer)   32,32      3 104         b:{1,32}, W:{32,32,3,1}
layer1 (Subsampling1DLayer)   -,-        0             -
layer2 (Convolution1DLayer)   32,64      6 208         b:{1,64}, W:{64,32,3,1}
layer3 (Subsampling1DLayer)   -,-        0             -
layer4 (Convolution1DLayer)   64,64      12 352        b:{1,64}, W:{64,64,3,1}
layer5 (Subsampling1DLayer)   -,-        0             -
layer6 (DenseLayer)           64,256     16 640        W:{64,256}, b:{1,256}
layer7 (DenseLayer)           256,128    32 896        W:{256,128}, b:{1,128}
layer8 (DenseLayer)           128,64     8 256         W:{128,64}, b:{1,64}
layer9 (OutputLayer)          64,1       65            W:{64,1}, b:{1,1}
--------------------------------------------------------------------------
            Total Parameters:  79 521
        Trainable Parameters:  79 521
           Frozen Parameters:  0
==========================================================================
```

Figure 3.9: Summary of the CNN structure

3.3.4 Heuristic Algorithm

Although it is not a ML algorithm, the heuristic algorithm is important because the results it produces serve as a comparison for the ML models. As mentioned in Chapter 2, the heuristic algorithm was already used in earlier stages of the tracking system development, so there was no need to develop it from scratch. However, the existing implementation was coded in Matlab, so a Java model of the heuristic algorithm had to be created.

This algorithm assumes that the phase difference values that make up the features of the dataset can be represented as a wave, similarly as represented in Figure 3.3. As each period of a wave corresponds to a full rotation of the circular antenna array, it is possible to know the variation of the incident AoA as a function of the phase shift (i.e., horizontal displacement) of a given profile. Since each period has 8 phase difference samples, an offset of 8/360º on the x-axis is equivalent to 1º of AoA variation. With this in mind, and knowing that the wave profile of an incident AoA of 0º intersects the y-axis when x=4.5, that value can be used as a reference to calculate the "deviation" of any other wave relative to the reference, and from there obtain the estimated AoA.

3.4 Simulator

The integration of the developed ML models with the Java-based simulator had the goal of assessing how well the models behaved tracking the position of a moving receiver.

Before integrating the models into the simulator, it is important to understand the principles of its operation. In this context, this section presents the functional description of the simulator.

The simulator models a two-dimensional map with adjustable width and height, which contains a configurable number of beacon objects at any map position and a single mobile receiver object which follows a trajectory defined by a parametric function. The map state is evaluated in time steps of 1 ms. Events generated by beacons and receiver are assessed at every time step and the simulation ends after a specified time interval.

During the simulation process, estimation of position from received angles is based on the assumption that the receiver is aware of the exact location of each transmitter, and that each received packet has a beacon identifier attached to it. To perform position estimation, the simulator resorts to a state-of-the-art approximate least-squares method. The simulator invokes this method once a given configurable number of packets, i.e., AoA vectors, have been gathered. The least-squares method interprets each AoA as an angle direction vector originating from the respective beacon. The least-squares method computes a single point that minimizes the distance between all the angle direction lines. This point corresponds to the estimated position. Figure 3.10 shows an example of position estimation via AoA and the described least-squares method for 3 AoAs from distinct wall-mounted beacons in a short 12x4 m corridor.

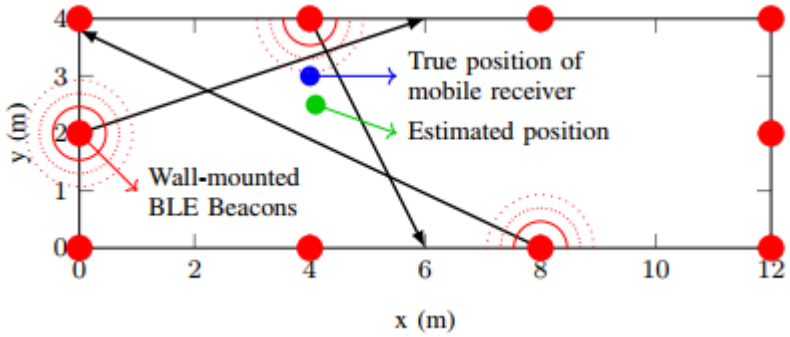


Figure 3.10: Example of position estimation via least-squares method

Previously to the development of this dissertation, the simulations resorted to AoA data gathered in real-world experiments performed with a commercial antenna array and a BLE transmitter, to achieve more realistic results. To estimate a measured angle, one of the thousand data points from the real-world dataset corresponding to that same true angle is sampled.

Another important feature of the simulator is the employment of a first-order Kalman Filter to "smooth" the estimations obtained via the least-squares method. Figure 3.11 shows an example of a simulation test, where a cosine trajectory is generated in a 4x100 meter room with 64 beacons uniformly distributed along the walls, with transmission period of 100 ms. The real trajectory is shown in blue, estimated position is shown in red and filtered position is shown in green.

To test the tracking performance of the developed ML models, these had to be integrated in the simulator in such a way that the AoAs are no longer sampled from the real-world dataset but are, instead, sampled directly from the models. As the receiver knows the true angle of transmissions, to obtain an estimation, a data point from a test dataset with a ground truth value as close as

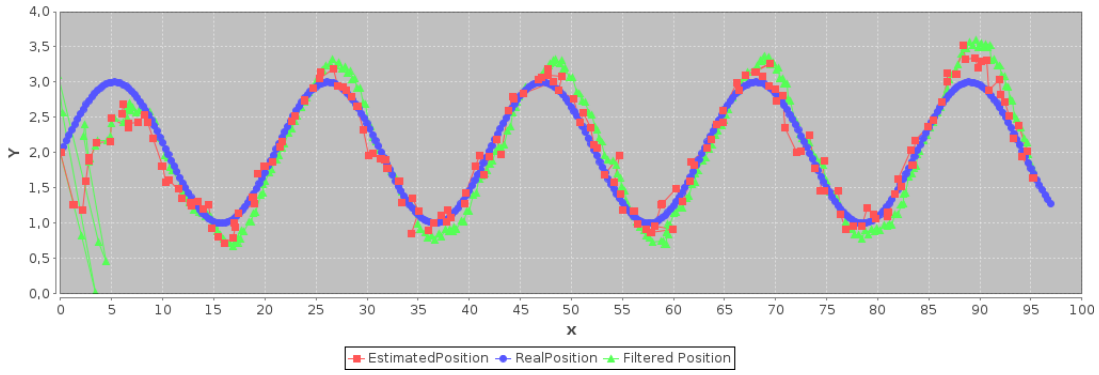


Figure 3.11: Position tracking example

possible to the true angle has to be fed to one of the ML models in order to obtain the desired estimation. This demands that the ML models are already trained previously to the integration.

In addition to the visual representation of the real trajectory versus estimated trajectory, the simulator also provides a measure of the RMSE between the position estimations and the real positions. Figure 3.12 summarizes the simulator's operation through a block diagram.

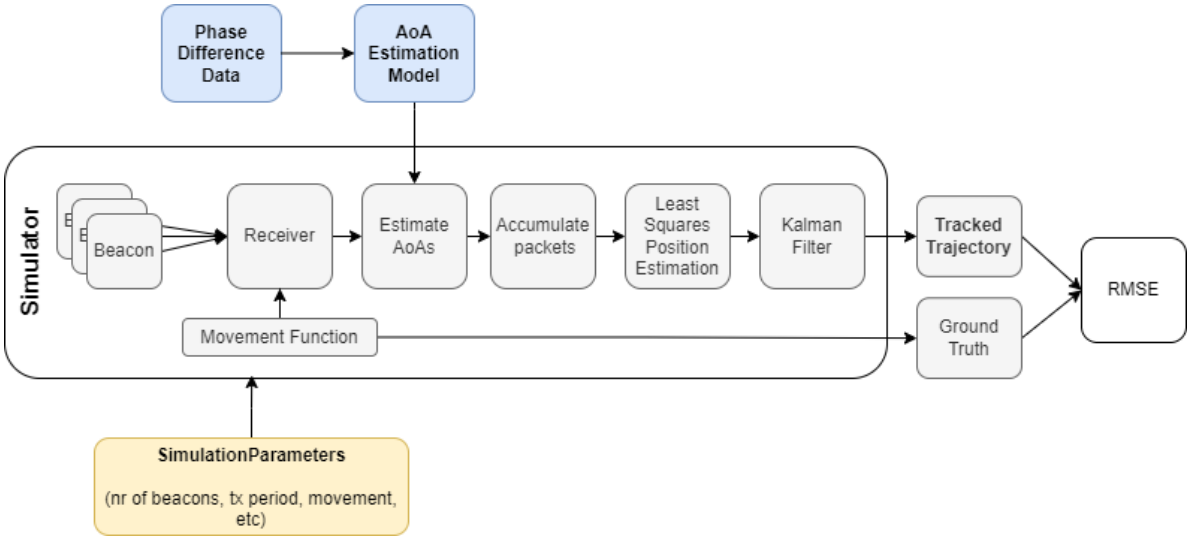


Figure 3.12: Simulator operation diagram

Chapter 4

Performance Evaluation

After the development of the models described in the previous chapter, the next step was to evaluate their performance in the estimation of the AoA.

The experimental tests were divided in two phases. The first phase aimed to train and test the models with all the generated datasets and with the real world dataset. The accuracy of the AoA estimations, as well as the training and testing times of each model were compared resorting to error metrics and time measurements, respectively.

In the second phase, the models were integrated in the Java simulator and several simulations were performed in order to evaluate the tracking accuracy produced by the AoA estimations of each model.

4.1 Angle-of-Arrival Estimation

The goal of the first experimental phase was to establish comparisons between the performance of all the developed models for AoA estimation. A critical factor in obtaining consistent comparisons is to make sure that all the models are trained and tested under the same exact datasets. In the next subsection, the experimental setup is described. In subsections 4.1.2, 4.1.3 and 4.1.4, the results produced by each model when trained and tested with each of the available datasets, both in terms of accuracy and computational cost, are presented. Finally, in Subsection 4.1.5, a comparison between the performance of each model is given.

4.1.1 Experimental Setup

As already explained in Section 3.1, a total of 27 datasets, with 72000 data points each, were generated in order to evaluate the models, plus the real world dataset, which has 12600 data points.

To train and test each of the models, every dataset was normalized and divided into two subdatasets: one for training and one for testing. The chosen ratio was 50:50, so each subdataset has 36000 data points (6300 in the case of the real world dataset).

To evaluate the models' performance, the MAE was used as the error metric of the estimations. After each test, the values of the deviations between the estimated AoAs and the associated ground

truths for each dataset were summed up and divided by the total datapoints in order to produce the MAE.

4.1.2 AoA Inference Results with KNN

In the KNN model, the training fraction of each dataset was used to construct the k-d tree. To perform the AoA estimations with the test set, the chosen value for the k parameter was 5.

After training and testing all the generated datasets, the obtained MAE between the estimated values and the real values was used to evaluate how the model performed as a function of an increasing noise level. Additionaly, the MAE has also served to evaluate the performance variation as a function of the datasets' number of features (8, 16 and 32). The obtained results are synthesized in Figure 4.1.

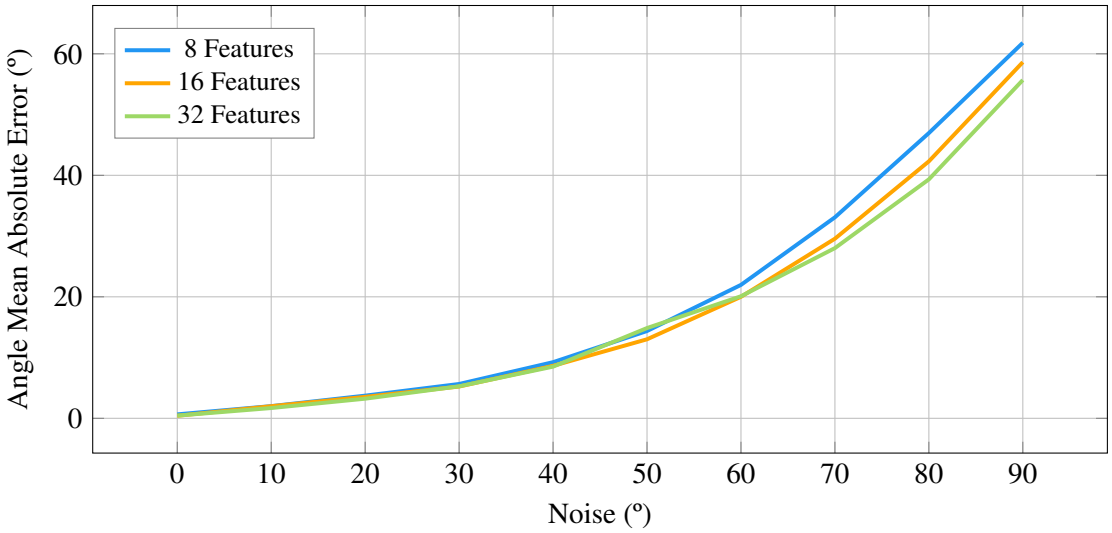
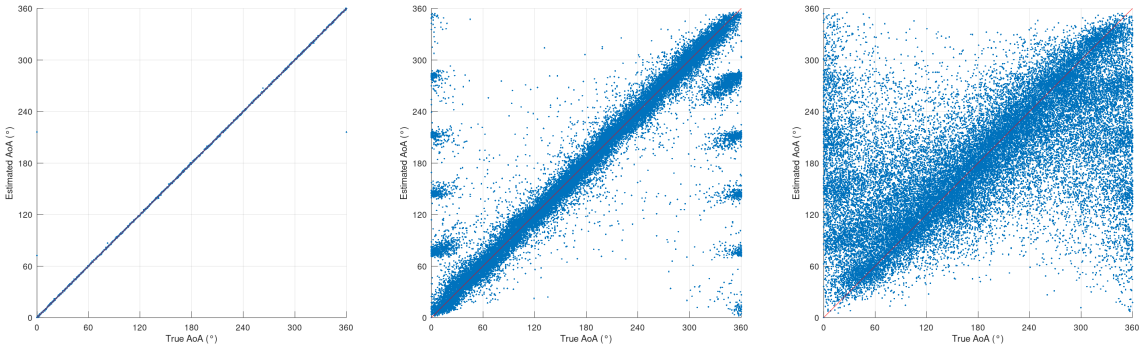


Figure 4.1: MAE as a function of noise for the KNN model, using datasets with different number of features

With regard to the performance as a function of the noise level, it is possible to verify that the model behaves in the same manner for all the 3 types of datasets. The MAE grows exponentially with the noise level increase. From noise levels with a standard deviation around 50º, the datasets' number of features starts to be an influential factor in the model performance. For high noise levels, the model generates better estimations using datasets with a larger number of features. Regarding the dataset with real world data, the obtained MAE was 53.84º.

To give a better perspective of the decrease in estimation precision as a function of increasing data noise, three scatter plots are shown in Figure 4.2, each of them representing a dataset with different noise level. The plots represent the regression obtained between the real AoA, which is the independent variable, and the estimated AoA, which is the dependent variable. All the 36000 test datapoints are featured in each of the plots.



(a) $\sigma = 0$ **(b)** $\sigma = 60$ **(c)** $\sigma = 90$

Figure 4.2: Scatter plots of the AoA estimations obtained with datasets of different noise (KNN)

In the first plot, which concerns the dataset without noise, the regression obtained is practically ideal, which means that the model was able to estimate AoAs very close to the real angles. With a dataset with a noise standard deviation of 60º, a greater dispersion of the points around the ideal regression line is observed and with the dataset with a noise standard deviation of 90º there is an even greater dispersion, which proves that the accuracy of the inferences decreases as a function of the increase in the noise of the datasets.

With respect to the temporal measures obtained in the training and testing of this model, Table 4.1 shows the obtained values. In this model, the training time is the equivalent to the time it takes to build the k-d tree. The averages presented are relative to all the different noise levels corresponding to a given number of features (8, 16 or 32). This does not apply in the real world data, since it consists of a single dataset.

Dataset	Average Training Time (s)	Average Testing Time (s)
8 Features	24.18	54.57
16 Features	24.69	56.89
32 Features	25.03	55.33
Real World	4.07	8.26

Table 4.1: Average training and testing times of the KNN model for each dataset type

4.1.3 AoA Inference Results with NNs

As three NN models were developed, it is important to understand which of the three yields better results. With that purpose, the same analysis was performed as for the KNN model. Figure 4.3 shows the produced MAEs using datasets of 8 features.

The same exponential increase in the MAE as the KNN is verified for the 3 models. The first model behaves considerably worse for all noise levels, when compared with the other two. The second model and the CNN model yield similar results.

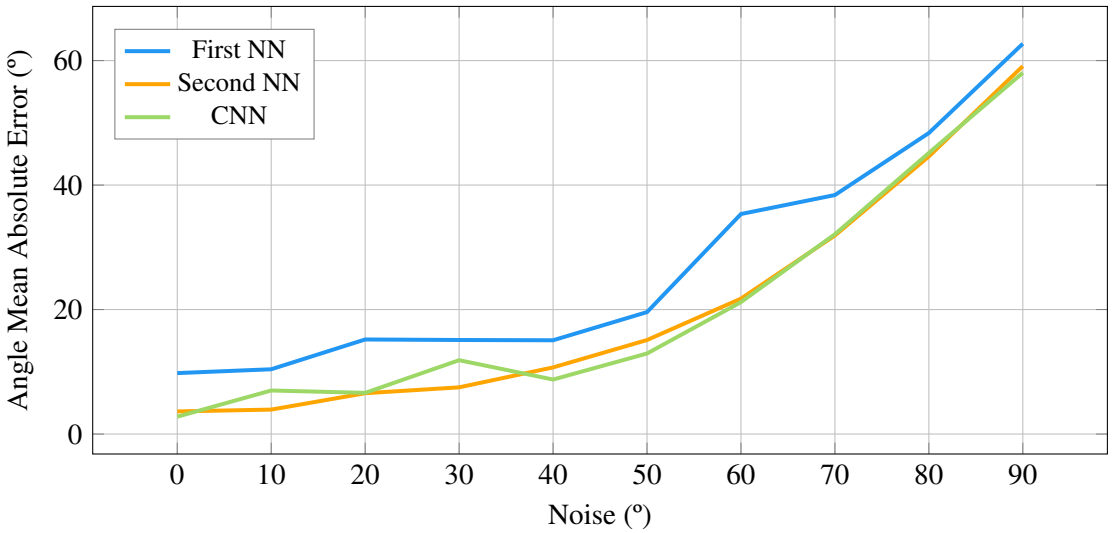
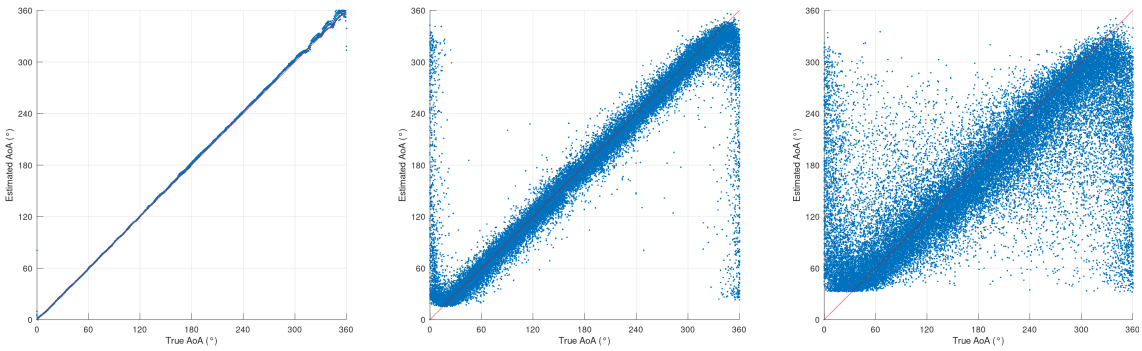


Figure 4.3: MAE as a function of noise for the three NN models (8 Features)

Figure 4.4 shows 3 scatter plots of the AoA estimations obtained with the CNN model. Each plot concerns a dataset with different noise level. For the dataset without noise, a practically ideal regression was obtained, similarly to the KNN model. With the increase in noise, the points appear more scattered around the ideal regression line, proving the increase in estimation error as a function of noise.



(a) $\sigma = 0$ **(b)** $\sigma = 60$ **(c)** $\sigma = 90$

Figure 4.4: Scatter plots of the AoA estimations obtained with datasets of different noise (CNN)

Regarding the training and testing times of each model, the obtained values are presented in Table 4.2.

It can be seen that the computation time is directly related to the complexity of the model. The first model, the simplest, requires the shortest computation times, while the CNN model, the most complex, requires the longest computation times.

Figure 4.5 shows the obtained MAEs for the three models when trained and tested with 16 feature datasets.

Model	Average Training Time (s)	Average Testing Time (s)
First NN	8.43	0.12
Second NN	14.83	0.17
CNN	37.36	0.40

Table 4.2: MAE of each NN model for 8 feature datasets

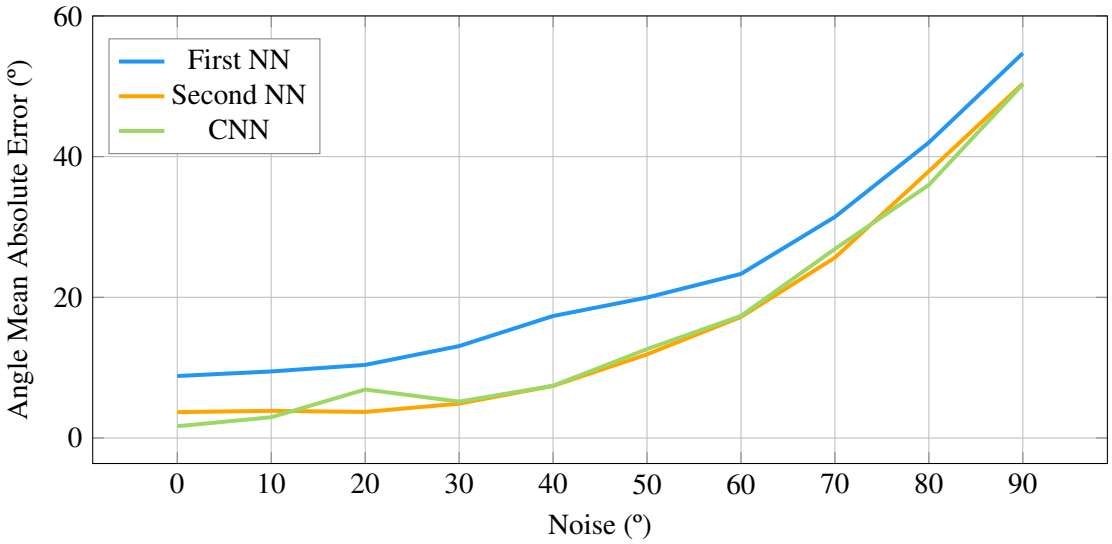


Figure 4.5: MAE as a function of noise for the three NN models (16 Features)

The relationship between the results produced by each model remains similar even though the feature increase generated an improvement for all models estimations, especially at high noise levels. The average training and testing times for each model are presented in Table 4.3.

Model	Average Training Time (s)	Average Testing Time (s)
First NN	8.80	0.11
Second NN	16.69	0.17
CNN	36.48	0.41

Table 4.3: MAE of each NN model for 16 feature datasets

All the obtained time measures are very similar to the ones obtained for 8 feature datasets, leading to the assumption that the number of features in the datasets has no influence in the models' computational cost.

Figure 4.6 shows the obtained MAEs using the datasets with 32 features, which followed the exact same tendency as the previous number of features. Table 4.4 shows the obtained time measurements, which were also very similar to those obtained with different feature sizes.

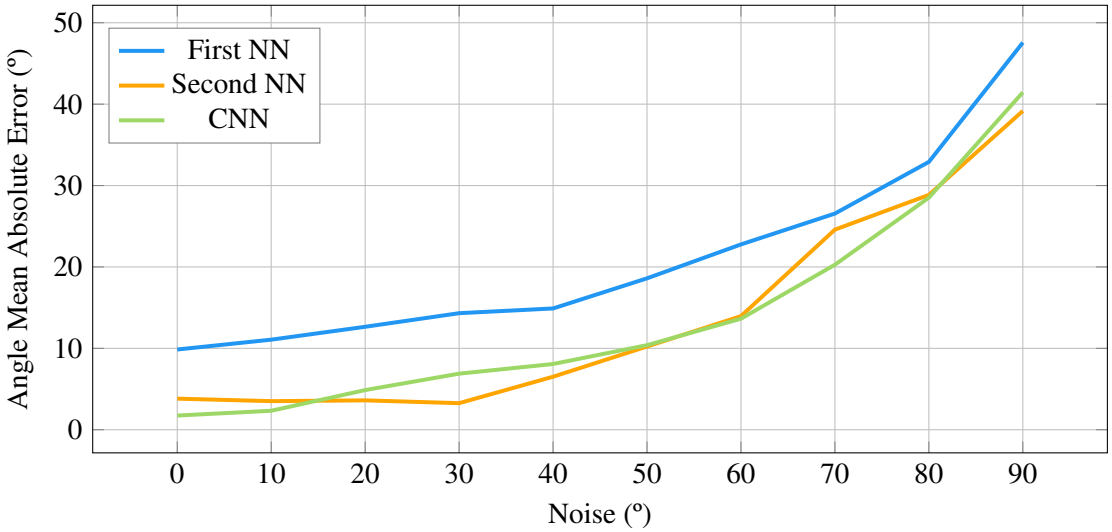


Figure 4.6: MAE as a function of noise for the three NN models (32 Features)

Model	Average Training Time (s)	Average Testing Time (s)
First NN	9.29	0.10
Second NN	15.42	0.17
CNN	37.67	0.42

Table 4.4: MAE of each NN model for 32 feature datasets

With respect to the real world dataset, the obtained MAE for each model is presented in Table 4.5, together with the training and testing times. While for all the generated datasets the second NN model and the CNN produced similar MAEs, for the real world dataset, the CNN model produced significantly better estimations than both the first and the second NN models, with the difference between the MAEs of the second model and the CNN model being greater than 10º.

Model	Mean Absolute Error (º)	Training Time (s)	Testing Time (s)
First NN	57.11	2.49	0.03
Second NN	53.02	5.38	0.05
CNN	43.27	9.46	0.11

Table 4.5: MAE, training time and testing time of each NN model for the real world dataset

4.1.4 AoA Inference Results with Heuristic

Contrary to the developed ML models, the heuristic model does not include a training phase, so the test datasets are directly used to evaluate the models' performance. The obtained MAE as function of the noise is presented in Figure 4.7, for the generated datasets. In this model, the number of features in the dataset has almost no influence in the accuracy of the estimations. For the real world dataset, this model produced a MAE of 237.65º.

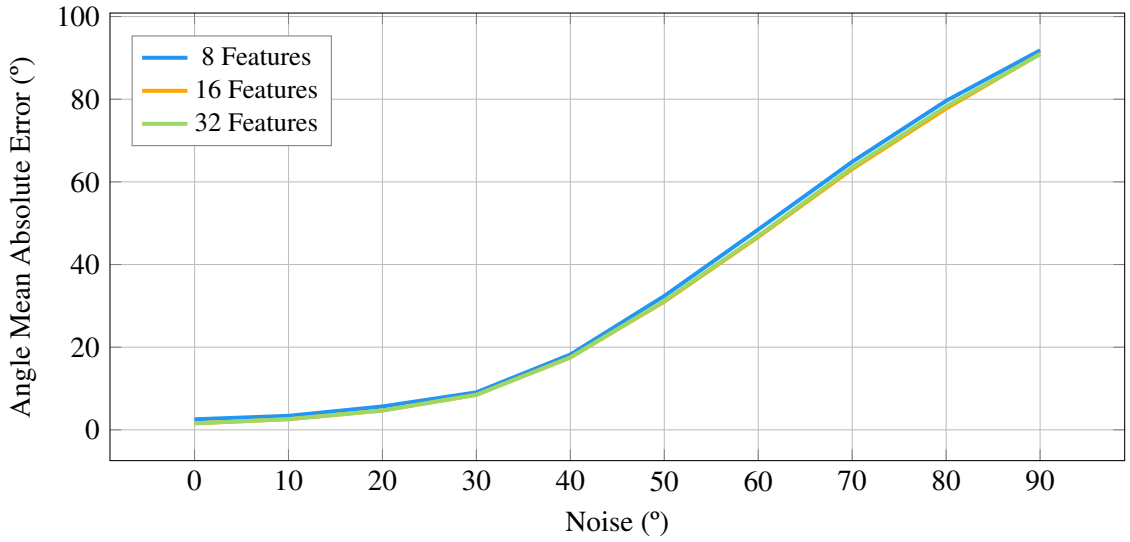
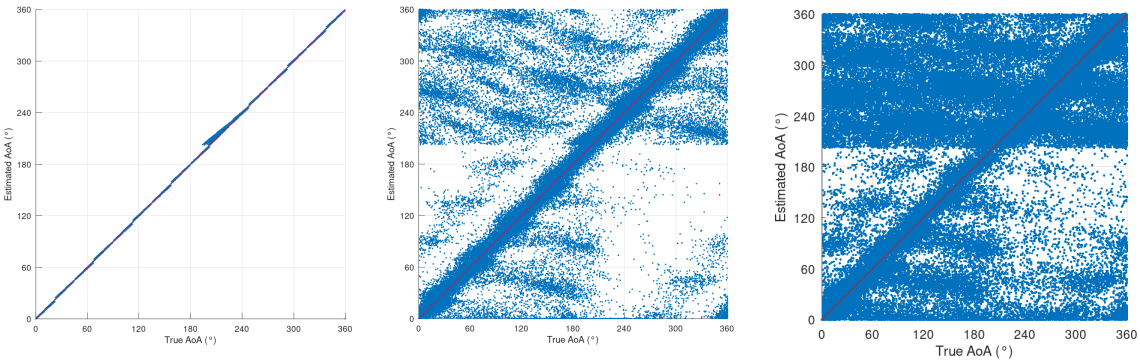


Figure 4.7: MAE as a function of noise for the Heuristic model, using datasets with different number of features

Three scatter plots of the estimations obtained with three different datasets are presented in Figure 4.8. It can be seen that, similarly to the ML models, for the noise-free dataset, the Heuristic model was also able to produce a near-optimal regression. However, the dispersion of points observed in the plots with a noise standard deviation of 60º and 90º is higher compared to that of the ML models, which proves that the Heuristic model has a lower estimation accuracy than the ML models, when subjected to increasingly noisy data.

When in comes to computation times, the measured average values are presented in Table 4.6.



(a) $\sigma = 0$ **(b)** $\sigma = 60$ **(c)** $\sigma = 90$

Figure 4.8: Scatter plots of the AoA estimations obtained with datasets of different noise (Heuristic)

Dataset	Average Testing Time (s)
8 Features	0.048
16 Features	0.051
32 Features	0.046
Real World	0.019

Table 4.6: Average testing time of the Heuristic model for each dataset type

4.1.5 Performance Comparison

This subsection uses the results presented in the previous subsection to establish a comparison between the performance of the different models. The first NN model is left out of this comparison since it has shown to have a significantly inferior performance when compared to the other two NN models.

4.1.5.1 8 Feature Datasets

In terms of the MAE, for 8 feature datasets, the developed models present similar values to the Heuristic model up to noise standard deviations around 30º. From there on, the developed models have much better noise robustness, as can be seen in Figure 4.9.

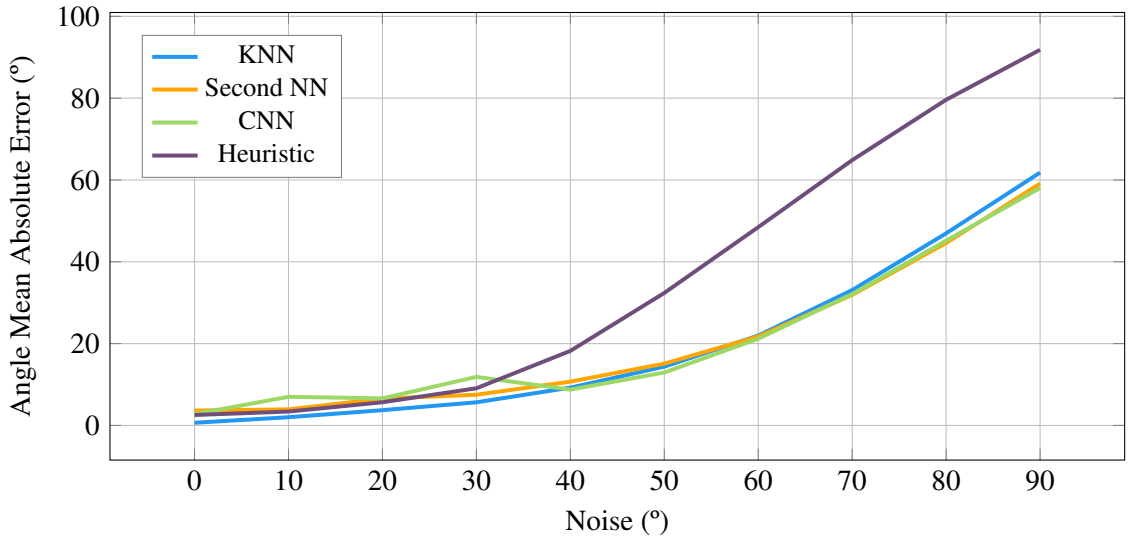


Figure 4.9: MAE as a function of noise for three different models (8 Features)

With regard to the AoA inference time, the same tendency does not apply, with the ML models having higher computing times compared to the Heuristic model. Even so, the NN models inference times are relatively close to the Heuristic's. On the other hand the KNN model has proven unable to compete with the other models in terms of computational cost, as can be seen in Table 4.7

Model	Average AoA Inference Time (μs)
KNN	1520
Second NN	4.72
CNN	11.1
Heuristic	1.33

Table 4.7: Average AoA inference time for each model (8 Features)

4.1.5.2 16 Feature Datasets

With 16 feature datasets, the developed models show an even more significant improvement over the Heuristic, with the two NN models also presenting a visible improvement in relation to the KNN model, for noise standard deviations from around 50, as shown by Figure 4.10.

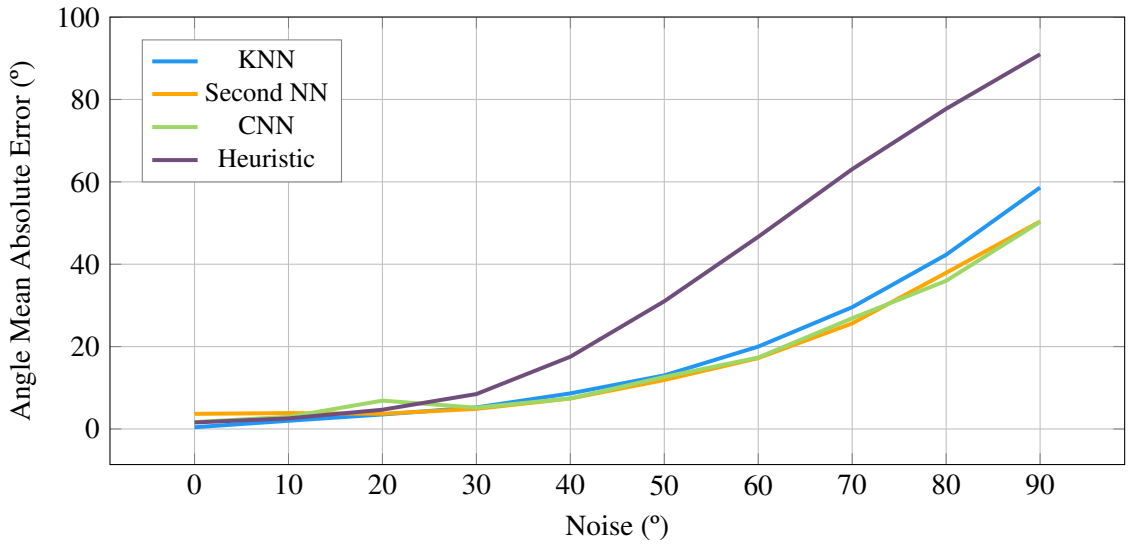


Figure 4.10: MAE as a function of noise for three different models (16 Features)

With regard to computational cost, none of the models has shown variations related to the number of features of the datasets, so the results obtained for 16 features are basically the same as for 8 features.

4.1.5.3 32 Feature Datasets

The observed tendencies when the number of features increased from 8 to 16 are also visible with the increase from 16 to 32 features, with even more pronounced differences as shown in Figure 4.11. For the reason mentioned above, the computational costs are the same as for the 8 feature datasets.

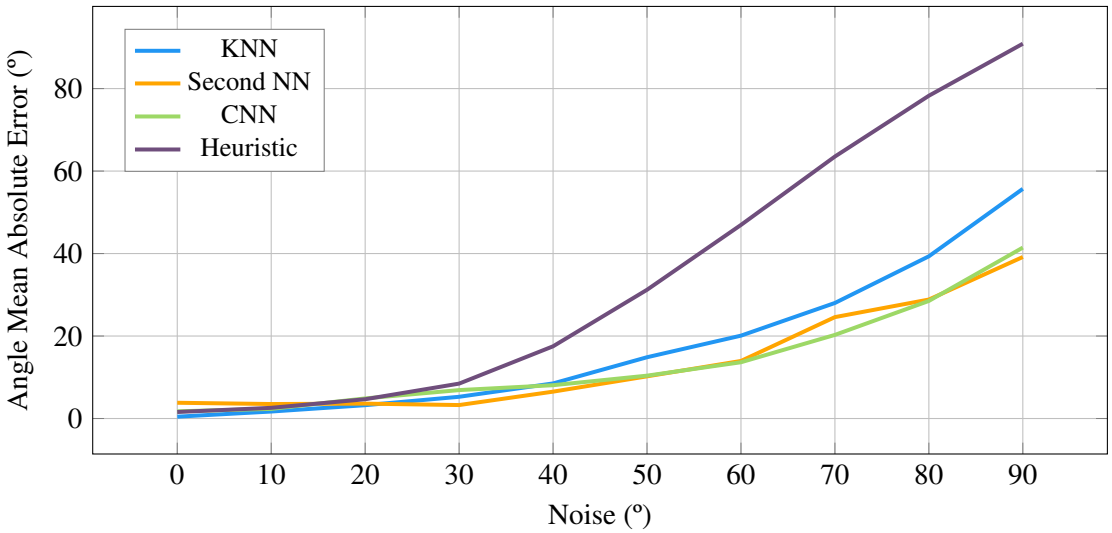


Figure 4.11: MAE as a function of noise for three different models (32 Features)

4.1.5.4 Real World Dataset

When tested with the real world dataset, the difference between the performances of the ML models and the Heuristic model was even more pronounced, with the Heuristic model being completely unable to perform accurate estimations under the real world dataset. The KNN and the second NN model presented very simmilar MAE values and the CNN model stood out as the most accurate model when tested with the real world dataset. Table 4.8 shows the MAEs obtained by each model.

Model	Mean Absolute Error (°)
KNN	53.84
Second NN	53.02
CNN	43.27
Heuristic	237.65

Table 4.8: MAE of each model for the real world dataset

Regarding the computational cost, the tendency is the same as for the generated datasets. However, since the real world dataset has less datapoints than the generated datasets, the obtained AoA inference times are shown in Table 4.9.

Model	Average AoA Inference Time (μs)
KNN	1310
Second NN	7.94
CNN	17.46
Heuristic	3.02

Table 4.9: Average AoA inference time for each model (Real World Dataset)

4.2 Integration with Simulator

To assess the performance of the AoA estimations in position tracking of a moving receiver, the ML models were integrated into the simulator. To obtain consistent comparisons between the models, all simulations were performed using the same trajectory. Thus, the simulator was configured such that the receiver travels in a sinusoidal trajectory in a 100 m long corridor of 4 m in width. The remaining simulator parameters remained constant and are presented in Table 4.10. For each model, a simulation was performed with each of the 27 generated datasets, plus the real world dataset. To evaluate the tracking performance, the error measure used to compare the offset between the real and estimated positions was the RMSE, since the simulator already implements a method that provides this measurement for each simulation. In the following subsections, the simulation results for each model are presented. For each model, a graphical representation of the RMSE as a function of noise is given. Furthermore, to give a better perpective of the tracking accuracy produced by data with different noise levels, several 2D representations of the real trajectories versus the estimated trajectories are shown.

Parameter	Value(s)
Map dimensions	100×4 m
Packet generation period of beacons	100 ms
Number of beacons on the map	64
Simulation time	35 s
Number of packets used in LSQ method	6

Table 4.10: Simulation Parameters

4.2.1 Tracking Results with KNN AoA Estimation

The obtained RMSE values as a function of the noise level of the datasets are shown in Figure 4.12. With respect to the real world dataset, the obtained RMSE was 6.22º.

As expected, the decreasing performance of AoA estimations as a function of noise is also reflected in the position tracking accuracy. The higher the noise, the more inaccurate are the estimated positions. However, while in the previous section we saw that increasing the number of features in the datasets yields better AoA estimates, especially for higher noise, this tendency was not observed in position tracking.

In Figure 4.13, a visual representation of the estimated positions in contrast with the real positions is presented through four subfigures, each with a different level of noise. The degradation of accuracy is noticeable. While for a noise standard deviation of 0º the estimations match almost exactly the real positions, for a high noise level the estimations do not preserve any trace of the sinusoidal shape.

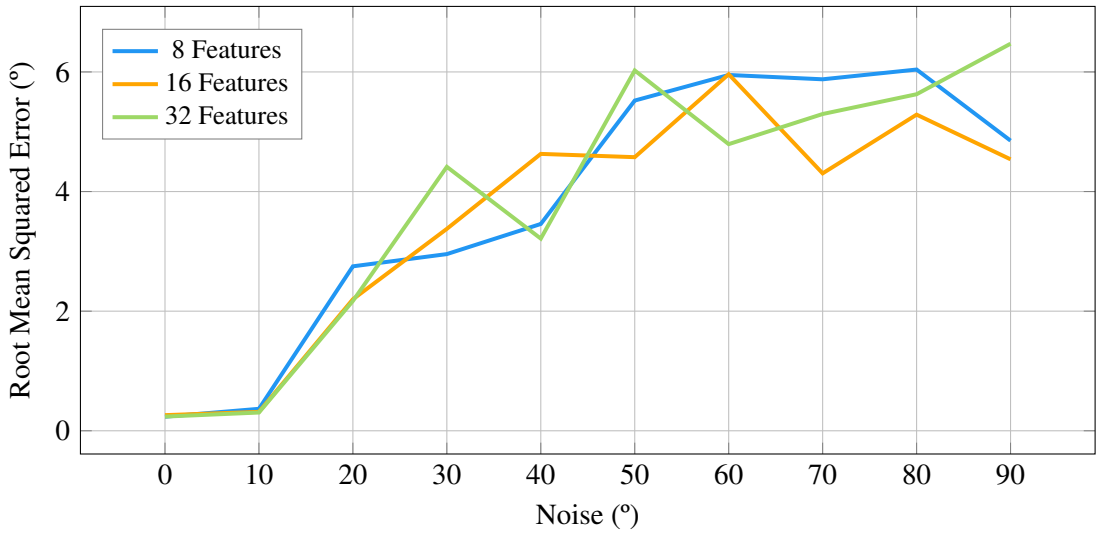
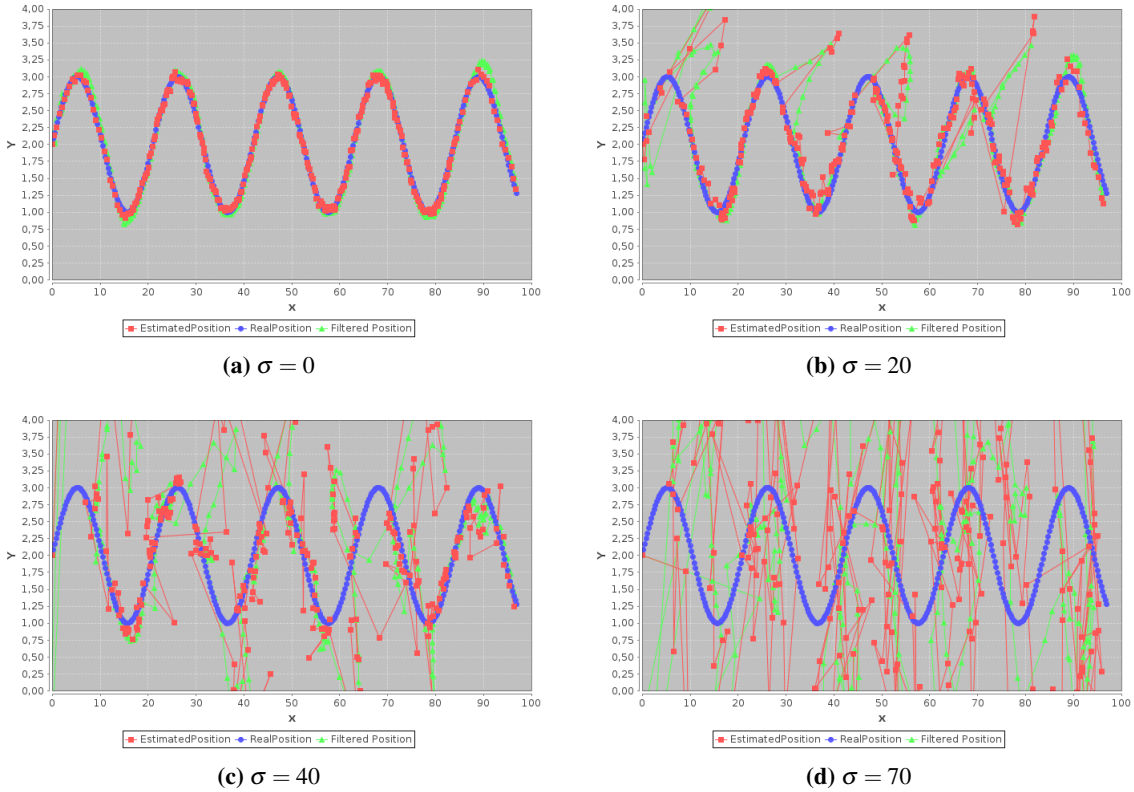


Figure 4.12: RMSE as a function of noise for the KNN, using datasets with different number of features



(a) $\sigma = 0$ **(b)** $\sigma = 20$

(c) $\sigma = 40$ **(d)** $\sigma = 70$

Figure 4.13: Evolution of the estimated trajectory with increasing noise, for the KNN model (8 feature data)

4.2.2 Tracking Results with NN AoA Estimation

Regarding the second NN model, the RMSE variation as a function of increasingly noisy data is shown in Figure 4.14. When using the real world dataset, the RMSE was 6.17º.

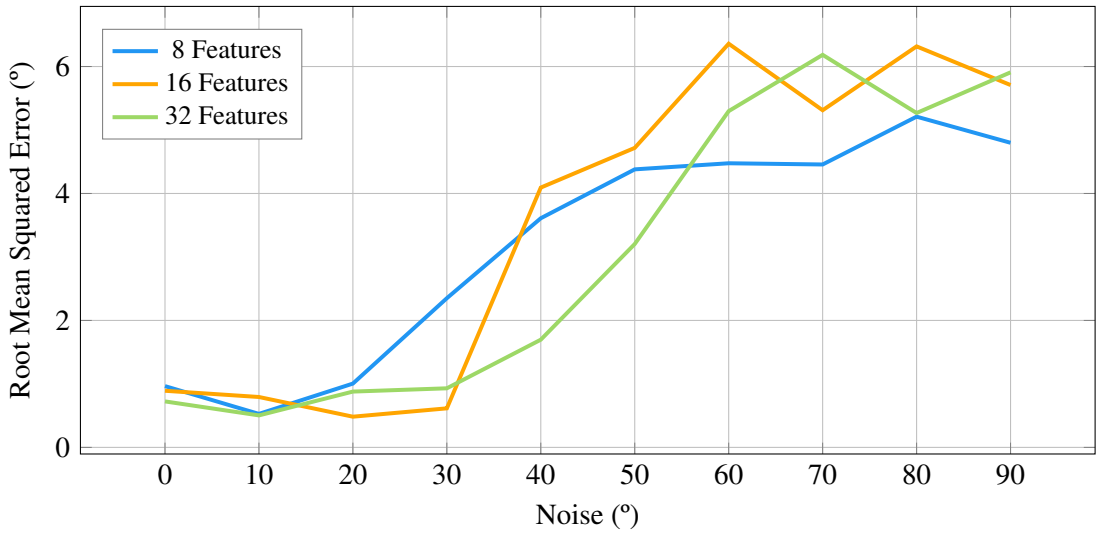
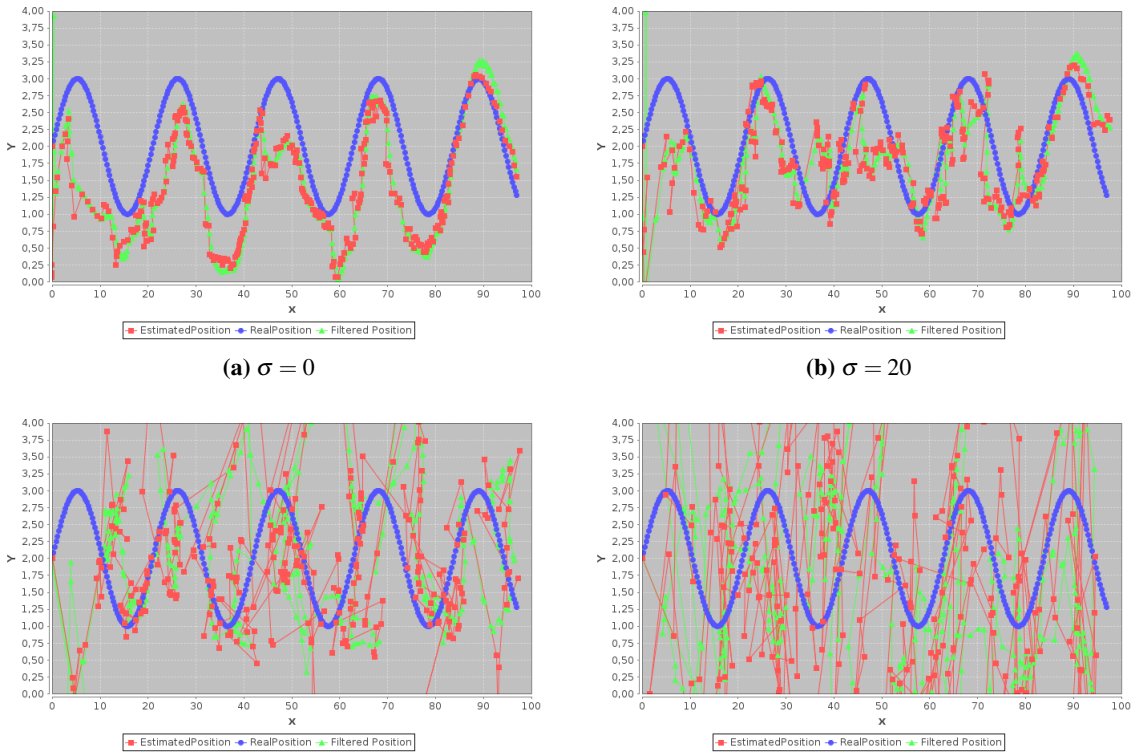


Figure 4.14: RMSE as a function of noise for the second NN, using datasets with different number of features

To obtain a better perspective of the aforementioned behaviors, several 2D representations of the tracked trajectories are shown in Figure 4.15.



(a) $\sigma = 0$ **(b)** $\sigma = 20$

(c) $\sigma = 40$ **(d)** $\sigma = 70$

Figure 4.15: Evolution of the estimated trajectory with increasing noise, for the second NN model (8 feature data)

Compared to the KNN model, the results obtained for the second NN model show that it

behaves worse under a noise standard deviation of 0º, but is able to maintain a consistent level of accuracy up to a noise standard deviation of 20º. From then on, the RMSE grows in a similar manner to the KNN model, until it stabilizes at values very similar to those of that model.

4.2.3 Tracking Results with CNN AoA Estimation

The obtained results in terms of RMSE for the CNN are shown in Figure 4.16. Compared with the previous NN model, there is a slight improvement in the results obtained for the 32 feature dataset, which stay below a RMSE of 2º up to noise standard deviations of 50º. There is also a very slight improvement when using the real world dataset, which generated a RMSE of 5.95º. Apart from this improvements, there is no significant change in the results obtained by CNN, when compared to the other NN.

Overall, comparing the simulation results of the NNs with those of the KNN, it is noticeable that both achieve similar RMSE values when subjected to the highest noise levels. However, the most significant difference is that the NNs reach maximum RMSE values from noise standard deviations around 60/70º, whereas the KNN reach those values from around 50º of standard deviation.

In Figure 4.17, the evolution of the estimated trajectory is shown, for different noise values.

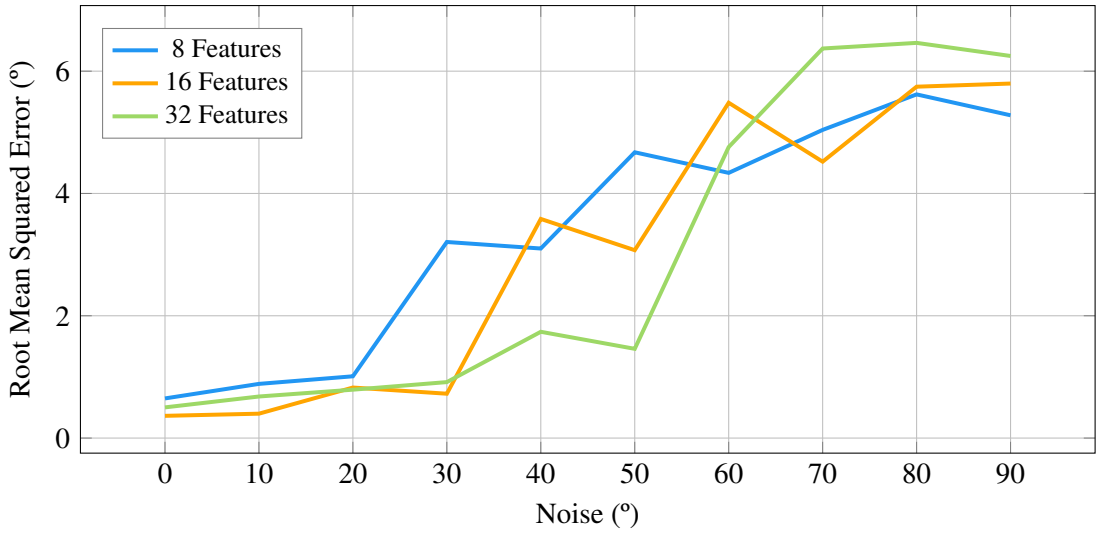
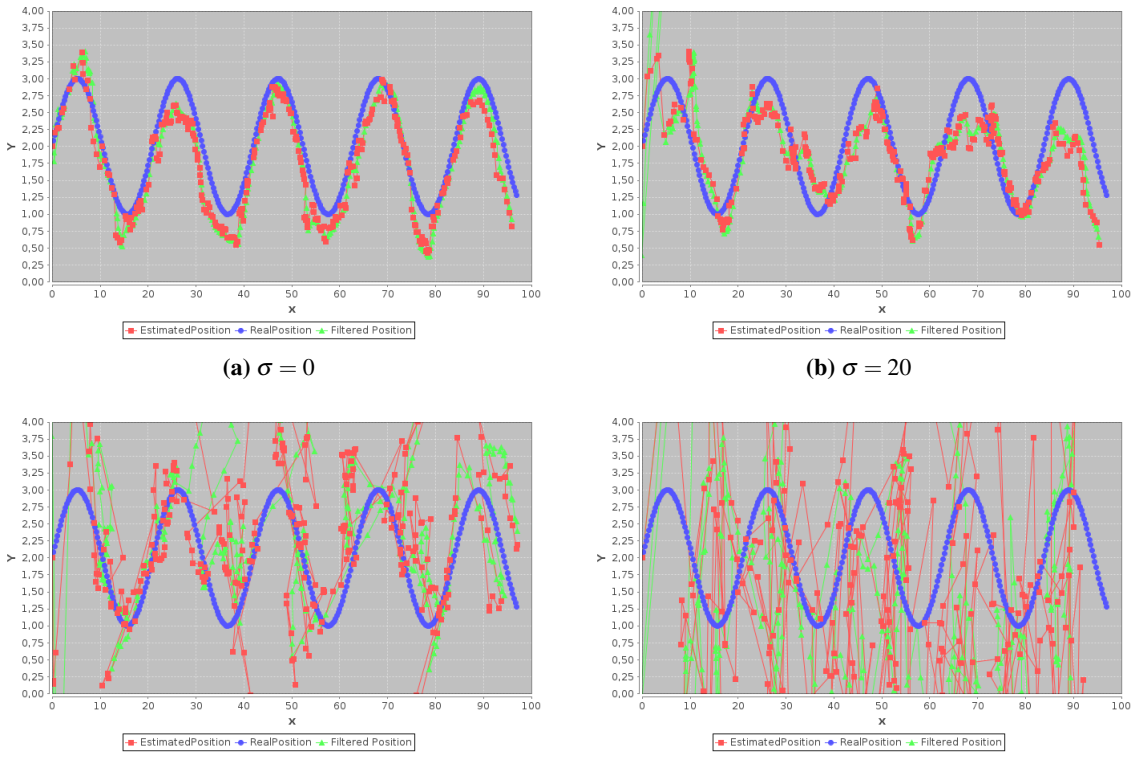


Figure 4.16: RMSE as a function of noise for the CNN, using datasets with different number of features



(a) $\sigma = 0$ **(b)** $\sigma = 20$

(c) $\sigma = 40$ **(d)** $\sigma = 70$

Figure 4.17: Evolution of the estimated trajectory with increasing noise, for the CNN model (8 feature data)

4.2.4 Tracking Results with Heuristic AoA Estimation

The simulations performed with AoA estimations obtained from the Heuristic model generated the RMSE values shown in Figure 4.18 and the tracking trajectories shown in Figure 4.19.

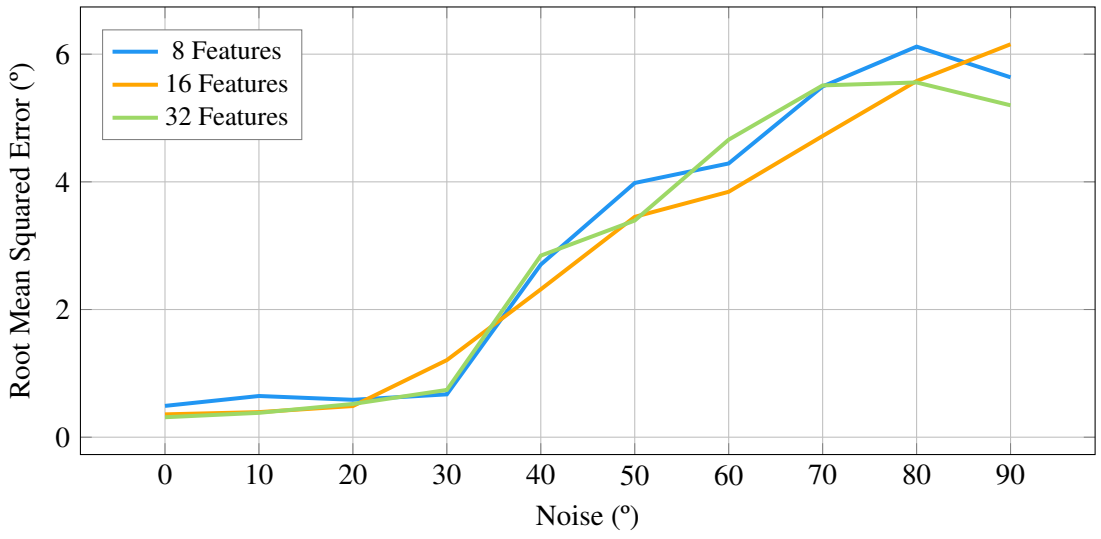
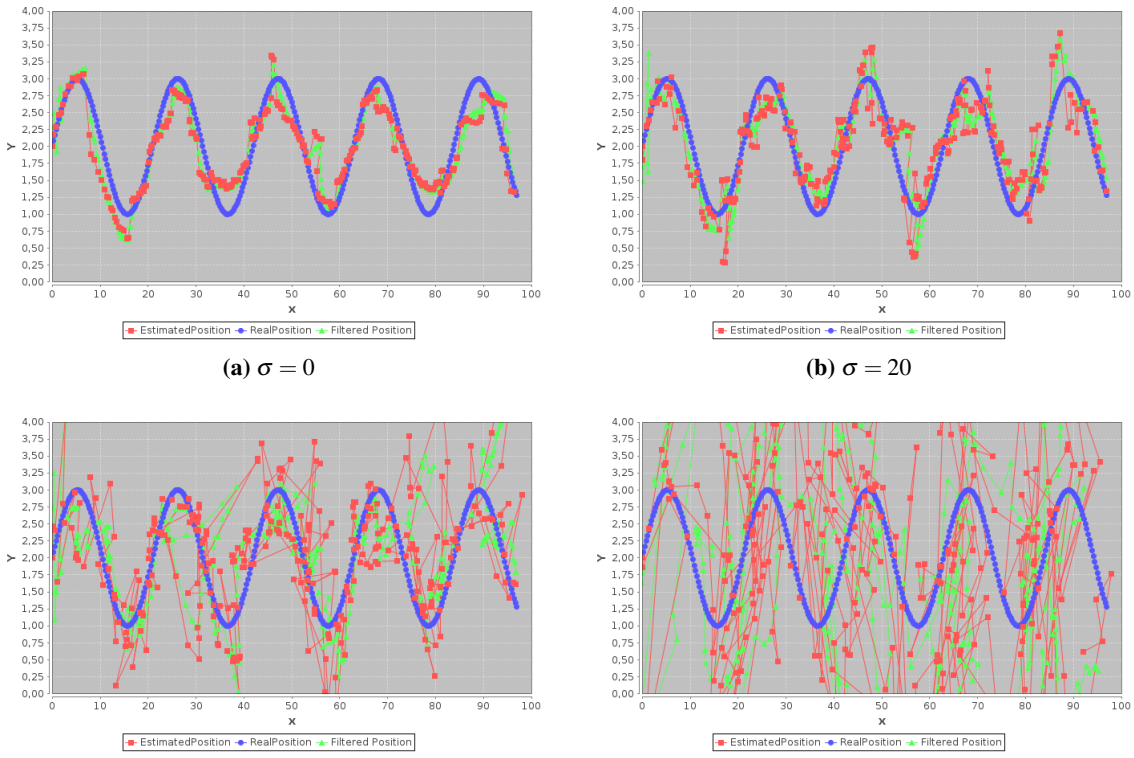


Figure 4.18: RMSE as a function of noise for the Heuristic model, with different number of data features



(a) $\sigma = 0$ **(b)** $\sigma = 20$

(c) $\sigma = 40$ **(d)** $\sigma = 70$

Figure 4.19: Evolution of the estimated trajectory with increasing noise, for the Heuristic model

In the performance evaluation of the AoA estimations it became clear that the Heuristic model was considerably less accurate compared to the ML models. With regard to tracking performance with the generated datasets, the accuracy achieved by the Heuristic model is, overall, slightly better than that obtained with the ML models. However, with respect to the accuracy obtained when using estimations from the real world dataset, the Heuristic model generated a much worse RMSE (15.97º).

Chapter 5

Conclusion and Future Work

This dissertation addressed two primary objectives. Firstly, we aimed to develop ML models to infer the AoA from phase difference data, aiming to outperform the accuracy of an existent heuristic method. Secondly, we integrated the developed models in an existent simulation framework, aiming to evaluate the quality of the AoA estimations produced by the models in a position tracking scenario.

A KNN model and three NNs were developed and their performance in AoA inference were evaluated against the heuristic model, resorting to the phase difference data of 27 synthetic datasets with different noise levels, plus a dataset with phase difference data collected in a real world experiment. The first goal of this dissertation was fully accomplished, with all the developed ML models showing to be effective in estimating the AoA from phase difference data. For data with low noise levels, the ML models showed a very similar accuracy to the heuristic model. However, for data with intermediate and high noise levels, both the KNN and the NNs outperformed the Heuristic by a wide margin, proving to be much more resistant to noise. For the real world data, where the noise values are random and unknown, the ML models outperformed the Heuristic by an even greater margin. Regarding the average AoA inference time, the computation time for the NNs were very close to those of the heuristic model, whereas the KNN has proven to be unable to compete with the other models in terms of computational cost. Regarding the evaluation of the tracking performance of the models, although the AoA estimations generated by the ML models were able to track trajectories with a good degree of accuracy regarding low noise datasets, the tracking quality of the heuristic model was not surpassed for estimations from the synthetic datasets. Yet, for the estimations produced by the real world dataset, the ML models showed significantly better tracking accuracy compared to the Heuristic.

In what concerns the simulation and tracking of trajectories, there are still a myriad of test scenarios that can be explored in order to obtain a better perspective of the tracking potential of the ML models. First of all, in this work, all the simulations were performed under the same sinusoidal trajectory and all the simulator parameters remained constant throughout all simulations. Future work can explore the variation of simulation performance as a function of, for instance, the number of beacon objects in the map, the packet transmission period of the beacons, and the variation of

the mobile receiver trajectories. Regarding the evaluation of the AoA estimation accuracy of the ML models, with the synthetic datasets, there is also an interesting test scenario that can be evaluated in future works. It consists of testing the models with different noise data than that used for training, e.g., train a model with a 60º noise dataset and test it with a 30º noise dataset. This would give a more realistic perspective of the models, since in real world scenarios the noise levels are unknown. Lastly, since this dissertation has proven that the use of ML techniques can outperform the heuristic method in terms of AoA inference accuracy, future works may also explore the development of more complex ML algorithms, with the goal of outperforming those developed in this work.

References

[1] Martin Woolley. Bluetooth core specification v5.1 feature overview. 2020.

[2] Qiang Wang, Binghao Li, and Chris Rizos. Dilution of precision in three dimensional angle-of-arrival positioning systems. *J. Electr. Eng. Technol.*, 14(6):2583–2593, November 2019.

[3] Bahzad Charbuty and Adnan Abdulazeez. Classification based on decision tree algorithm for machine learning. *Journal of Applied Science and Technology Trends*, 2(01):20 – 28, Mar. 2021. doi:10.38094/jastt20165.

[4] Massimo Merenda, Carlo Porcaro, and Demetrio Iero. Edge machine learning for ai-enabled iot devices: A review. *Sensors*, 20(9):2533, Apr 2020. URL: http://dx.doi.org/10.3390/s20092533, doi:10.3390/s20092533.

[5] I.N. da Silva, D.H. Spatti, R.A. Flauzino, L.H.B. Liboni, and S.F. dos Reis Alves. *Artificial Neural Networks: A Practical Course*. Springer International Publishing, 2016. URL: https://books.google.pt/books?id=DL_mDAAAQBAJ.

[6] Min Chen, Yi Gong, and Xingpeng Mao. Deep neural network for estimation of direction of arrival with antenna array. *IEEE Access*, 8:140688–140698, 2020. doi:10.1109/ACCESS.2020.3012582.

[7] Aysha Alteneiji, Ubaid Ahmad, Kin Poon, Nazar Ali, and Nawaf Almoosa. Angle of arrival estimation in indoor environment using machine learning. In *2021 IEEE Canadian Conference on Electrical and Computer Engineering (CCECE)*, pages 1–6, 2021. doi:10.1109/CCECE53047.2021.9569205.

[8] Wenli Zhu, Min Zhang, Pengfei Li, and Chenxi Wu. Two-dimensional doa estimation via deep ensemble learning. *IEEE Access*, 8:124544–124552, 2020. doi:10.1109/ACCESS.2020.3005221.

[9] Nuno Paulino, Luís M. Pessoa, André Branquinho, and Edgar Gonçalves. Design and experimental evaluation of a bluetooth 5.1 antenna array for angle-of-arrival estimation. In *2022 13th International Symposium on Communication Systems, Networks and Digital Signal Processing (CSNDSP)*, pages 625–630, 2022. doi:10.1109/CSNDSP54353.2022.9907908.

[10] Nuno Paulino and Luís M. Pessoa. Self-localization via circular bluetooth 5.1 antenna array receiver. *IEEE Access*, 11:365–395, 2023. doi:10.1109/ACCESS.2022.3233130.

[11] Nuno Paulino, Luís M. Pessoa, André Branquinho, and Edgar Gonçalves. Evaluating a novel bluetooth 5.1 aoa approach for low-cost indoor vehicle tracking via simulation. In *2021 Joint European Conference on Networks and Communications & 6G Summit (EuCNC/6G Summit)*, pages 259–264, 2021. doi:10.1109/EuCNC/6GSummit51104.2021.9482525.

[12] Aftab Khan, Stephen Wang, and Ziming Zhu. Angle-of-arrival estimation using an adaptive machine learning framework. *IEEE Communications Letters*, 23(2):294–297, 2019. doi:10.1109/LCOMM.2018.2884464.

[13] Mi Yang, Bo Ai, Ruisi He, Chen Huang, Zhangfeng Ma, Zhangdui Zhong, Junhong Wang, Li Pei, Yujian Li, and Jing Li. Machine-learning-based fast angle-of-arrival recognition for vehicular communications. *IEEE Transactions on Vehicular Technology*, 70(2):1592–1605, 2021. doi:10.1109/TVT.2021.3054757.

[14] Faheem Zafari, Athanasios Gkelias, and Kin K. Leung. A survey of indoor localization systems and technologies. *IEEE Communications Surveys & Tutorials*, 21(3):2568–2599, 2019. doi:10.1109/COMST.2019.2911558.

[15] Hui Liu, Houshang Darabi, Pat Banerjee, and Jing Liu. Survey of wireless indoor positioning techniques and systems. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 37(6):1067–1080, 2007. doi:10.1109/TSMCC.2007.905750.

[16] Dragan Stojanovic and Natalija Stojanovic. Indoor localization and tracking: Methods, technologies and research challenges. facta universitatis. *Series: Automatic Control and Robotics*, 13:57–72, 01 2014.

[17] Jacopo Tosi, Fabrizio Taffoni, Marco Santacatterina, Roberto Sannino, and Domenico Formica. Performance evaluation of bluetooth low energy: A systematic review. *Sensors*, 17(12):2898, Dec 2017. URL: http://dx.doi.org/10.3390/s17122898, doi:10.3390/s17122898.

[18] Issam El Naqa and Martin J. Murphy. *What Is Machine Learning?*, pages 3–11. Springer International Publishing, Cham, 2015. doi:10.1007/978-3-319-18305-3_1.

[19] Pádraig Cunningham, Matthieu Cord, and Sarah Jane Delany. *Supervised Learning*, pages 21–49. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008. doi:10.1007/978-3-540-75171-7_2.

[20] Zoubin Ghahramani. *Unsupervised Learning*, pages 72–112. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004. doi:10.1007/978-3-540-28650-9_5.

[21] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press Ltd, 2018.

[22] Jui-Chan Huang, Kuo-Min Ko, Ming-Hung Shu, and Bi-Min Hsu. Application and comparison of several machine learning algorithms and their integration models in regression problems. *Neural Computing and Applications*, 32(10):5461–5469, 2020. doi:10.1007/s00521-019-04644-5.

[23] Sadegh Bafandeh Imandoust, Mohammad Bolandraftar, et al. Application of k-nearest neighbor (knn) approach for predicting economic events: Theoretical background. *International journal of engineering research and applications*, 3(5):605–610, 2013.

[24] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, September 1975. doi:10.1145/361002.361007.

[25] Carl Kingsford and Steven L Salzberg. What are decision trees? *Nature Biotechnology*, 26(9):1011–1013, 2008. doi:10.1038/nbt0908-1011.

[26] Keiron O'Shea and Ryan Nash. An introduction to convolutional neural networks. *CoRR*, abs/1511.08458, 2015. arXiv:1511.08458.

[27] Shuai He, Hang Long, and Wei Zhang. Multi-antenna array-based aoa estimation using bluetooth low energy for indoor positioning. In *2021 7th International Conference on Computer and Communications (ICCC)*, pages 2160–2164, 2021. doi:10.1109/ICCC54389.2021.9674235.

[28] R. Schmidt. Multiple emitter location and signal parameter estimation. *IEEE Transactions on Antennas and Propagation*, 34(3):276–280, 1986. doi:10.1109/TAP.1986.1143830.

[29] So-Hee Jeong, Byung-kwon Son, and Joon-Ho Lee. Asymptotic performance analysis of the music algorithm for direction-of-arrival estimation. *Applied Sciences*, 10(6):2063, Mar 2020. doi:10.3390/app10062063.

[30] Yufan Fan, Cemil Emre Ardic, Minh Trinh-Hoang, and Marius Pesavento. Decentralized online direction-of-arrival estimation and tracking. In *2022 IEEE 12th Sensor Array and Multichannel Signal Processing Workshop (SAM)*, pages 6–10, 2022. doi:10.1109/SAM53842.2022.9827864.

[31] R. Roy and T. Kailath. Esprit-estimation of signal parameters via rotational invariance techniques. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 37(7):984–995, 1989. doi:10.1109/29.32276.

[32] *Decentralized Direction Finding Using Partly Calibrated Arrays*. Zenodo, September 2013. doi:10.5281/zenodo.43740.

[33] Catarina Marques. Indoor bluetooth low energy direction finding via circular antenna array. Master's thesis, Instituto Superior de Engenharia do Porto, 2022.

[34] Francisco Pimenta. Indoor location based on aoa and bluetooth low energy. Master's thesis, Faculdade de Engenharia da Universidade do Porto, 2020.

[35] Telmo Soares. Vehicle tracking in warehouses via bluetooth beacon angle-of-arrival. Master's thesis, Faculdade de Engenharia da Universidade do Porto, 2021.

[36] Eclipse Deeplearning4j Development Team. Deeplearning4j: Open-source distributed deep learning for the jvm, apache software foundation license 2.0. URL: https://deeplearning4j.konduit.ai/.