Indoor Positioning and Machine Learning Algorithms

Raavi Uttarwar and Julián Valentín
ra3878ut-s@student.lu.se,ju4508va-s@student.lu.se

Department of Electrical and Information Technology
Lund University

Supervisor: Meifang Zhu, Peter Karlsson

Examiner: Fredrik Rusek

June 11, 2021

© 2021
Printed in Sweden
Tryckeriet i E-huset, Lund

Abstract

This master thesis focuses around improving the efficiency and accuracy of existing indoor positioning systems with the help of Machine Learning (ML). Our work is based on Bluetooth Low Energy (BLE) v5.1. Position estimation is currently being carried out using the Least-Squares (LS) method in the framework. Introducing Machine Learning to position estimation can reduce computation time and increase accuracy of the system because of the additional "learning" in the form of Machine Learning models that is done by the system. An attempt has been made to extract information from the Direction-finding feature that BLE v5.1 presents, and combine it with ML to potentially improve the current position estimation.
In order to test the efficacy of these ML algorithms, a wide range of data has been used for these experiments. This included data from different simulated indoor environments and from measurements done physically in a real office environment.
We have experimented with three different Machine Learning algorithms for classification and regression: Random Forest, Support Vector Machine and k Nearest Neighbors. Each algorithm has shown impressive results with centimetre-level accuracy, indicating that it can be rewarding to explore ML even more for the purpose of indoor positioning.

Keywords: Indoor Positioning, Machine Learning, Bluetooth Low Energy

Popular Science Summary

Indoor positioning comes into play when satellite-based systems such as Global Positioning System (GPS) are not able to provide accurate information. This happens because these systems generally require a direct Line-Of-Sight (LOS) to communicate effectively, or the device to have access to as many satellites as possible. This is not always the case indoors. GPS systems do not work well inside closed structures, especially those built primarily of concrete, because these signals are too weak to be able to penetrate solid structures, which result in high loss in received signal power [1].
In these situations, short-range systems such as Bluetooth and WiFi work very well, where a high level of accuracy cannot be met using GPS. Indoor positioning systems are useful in scenarios where tracking and location-based information of objects are required. This includes Internet of Things (IoT) - e.g. home automation systems - airports, factories, warehouses, places providing healthcare, and transportation facilities [2]. There is a large variety of areas in day-to-day life where positioning using Bluetooth can prove to be helpful, and can make tasks easier. Bluetooth positioning is able to bring the positioning error down to sub-metre accuracy [3]. Indoor positioning with BLE works best when multiple BLE beacons are working together. This usually increases position accuracy. Bluetooth devices carry out positioning using Received Signal Strength Indicator (RSSI), Angle of Arrival or Angle of Departure (AoD).
We have mainly worked on improving the position estimates of a Bluetooth tag provided by four anchor points in an office environment. One of the key features of BLE v5.1 is direction finding, so Bluetooth beacons can provide information about the direction of the incoming signal. We have used the AoAs and RSSI provided by four anchor points to give a better estimate of the position of the Bluetooth device in question using Machine Learning algorithms.

Table of Contents

1 Introduction **1**
- 1.1 Background and Motivation . . . 1
- 1.2 Methodology . . . 2
- 1.3 Literature Survey . . . 2
- 1.4 Thesis Organization . . . 3

2 Theoretical Background **5**
- 2.1 Multiple Antenna Systems . . . 5
- 2.2 Multipath Propagation . . . 5
- 2.3 Bluetooth Low Energy v5.1 . . . 9
- 2.4 Introduction to Machine Learning . . . 10
- 2.5 Machine Learning Algorithms . . . 11
- 2.6 Least-Squares . . . 15

3 Positioning Estimation with Machine Learning **17**
- 3.1 Arranging Testbench Data . . . 17
- 3.2 Parameter Optimization . . . 19
- 3.3 Estimations in Different Indoor Environments . . . 24
- 3.4 Cross-prediction with Simulated Indoor Environments . . . 33
- 3.5 Estimations using Measurement Data . . . 35
- 3.6 Position Estimation Results . . . 40
- 3.7 Discussions and Future Work . . . 48

4 Conclusions **51**

References **53**

List of Figures

2.1 Scattering 6
2.2 Reflection 6
2.3 Diffraction 6
2.4 Refraction 6
2.5 Angle of Arrival and Angle of Departure [8] 8
2.6 Azimuth and Elevation [10] 8
2.7 Examples of Classification and Regression. [12] 11
2.8 k Nearest Neighbors [15] 12
2.9 Random Forest [16] 13
2.10 Support Vector Machine [20] 13

3.1 Bluetooth positioning scheme performed at u-blox. 18
3.2 Scheme of a Machine Learning process. 18
3.3 Different Parameters in the Simulated Scenario: Regression 21
3.4 Different Parameters in the Real (Measured) Scenario: Classification. 22
3.5 Different Parameters in Real Scenario: Regression. 23
3.6 Floor Plan of an Office Environment. 25
3.7 Simulated Environments. 25
3.8 LOS Distribution for High Concrete/Wood. 26
3.9 LOS Distribution for Mid Concrete/Wood. 26
3.10 LOS Distribution for Low Concrete/Wood. 26
3.11 RSSI for Concrete Scenarios. 27
3.12 RSSI for Furniture Scenarios. 27
3.13 LOS vs. Furniture Scenario. 27
3.14 LOS vs. Concrete Scenario. 27
3.15 Error Distribution for RF. 30
3.16 Error distribution for SVM. 30
3.17 Error Distribution for kNN. 31
3.18 Error Distribution for LS. 31
3.19 Comparison of Error Distribution in a Concrete Scenario. 32
3.20 Comparison of Error Distribution in a Furniture Scenario. 32
3.21 Cross-Prediction with RF Regressor. 34
3.22 Cross-Prediction with SV Regressor. 34
3.23 Cross-Prediction with kNN Regressor. 35

3.24 Floor Plan of a Real-Life Office Environment. 36
3.26 CDF of Positioning Error from Regression. 38
3.27 CDF of Positioning Error from Classification. 38
3.28 X-coordinate: RF 39
3.29 Y-coordinate: RF 39
3.30 X-coordinate: SVM 39
3.31 Y-coordinate: SVM 39
3.32 X-coordinate: kNN 39
3.33 Y-coordinate: kNN 39
3.34 Proposed ML System. 40
3.35 True and Estimated Path of the tag using LS Technique. 41
3.36 Position Estimations using RF in High Concrete Scenario. 41
3.37 Position Estimations using SVM in High Concrete Scenario. 42
3.38 Position Estimations using kNN in High Concrete Scenario. 42
3.39 Position Estimations using RF in High Furniture Scenario. 43
3.40 Position Estimations using SVM in High Furniture Scenario. 43
3.41 Position Estimations using kNN in High Furniture Scenario. 44
3.42 Position Estimation using RF Classifier in Office Environment. 45
3.43 Position Estimation using SV Classifier in Office Environment. 45
3.44 Position Estimation using kNN Classifier in Office Environment. . . . 46
3.45 Position Estimation using RF Regressor in Office Environment. . . . 46
3.46 Position Estimation using SV Regressor in Office Environment. . . . 47
3.47 Position Estimation using kNN Regressor in Office Environment. . . . 47
3.48 Mechanism of the Kalman Filter [29]. 48

List of Tables

2.1 Computational Complexity for each algorithm. [22] 14

3.1 Parameter values used for Optimization. 19
3.2 Optimized parameters. 19
3.3 Training and Testing Times: Random Forest. 21
3.4 Training and Testing Times: SV Regressor. 22
3.5 Training and Testing Times: kNN Regressor. 23
3.6 Dielectric Properties of Wood and Concrete. 24
3.7 MAE for x and y using RF. 28
3.8 MAE for x and y using SVM. 28
3.9 MAE for x and y using kNN. 29
3.10 Mean Euclidean Distance. 29
3.11 Average Positioning Error for Measurement Data. 37

Chapter 1

Introduction

1.1 Background and Motivation

Due to the rapid increase in smartphones and the spread of many other different wireless devices over the last decade, IoT is becoming an area of tremendous interest for different kinds of industries. Traffic monitoring, wearables, healthcare, and energy saving are some of the many different application areas for IoT. The addition of location-based systems (LBS) to such a developing field provides richer and more accurate service to the end users.
Location-based technologies such as GPS and Galileo have proven to be of great importance and reliance for outdoor scenarios. Nevertheless, in indoor scenarios, where the positioning must be more accurate as compared to the metre to sub-metre precision achieved by outdoor LBS, multipath propagation of the information transmitted plays a crucial role in the system performance. GPS and Galileo have been discarded for indoor purposes due to their low performance. Therefore, it is time for technologies such as Wi-Fi and BLE to deal with indoor positioning, which faces inconveniences caused by multipath propagation due to objects present in every room.
Indoor positioning has been a field of study for over 30 years and yet, new technologies are in constant development. Depending on the signal and the environment, some existing indoor positioning technologies give better results. System requirements such as power consumption and computational complexity also play an important role at the point of deciding which technology should be used for indoor localization.
The motivation behind introducing ML to position estimation is its unparalleled ability to adapt to different scenarios, which makes it desirable in so many different applications, may it be engineering, economics, or even medicine. This is because, essentially, ML is statistics and numbers, which are present in every aspect of life in the form of data. Using this data to enhance the understanding of a system on how to generate an output proves to be beneficial because not only is the system adapting to the data, but is also being taught how to learn, process, and use the data through a large variety of ML methods that we know of today.
Combining ML with applications in almost all cases simply leads to better results and enhanced efficiency of the system.

1.2 Methodology

The project revolves around comparing the performance of different ML models in estimating the position of a Bluetooth tag.
Along with real-time measurement data, we have also worked with 7 different simulated indoor scenarios. We studied the error performance of three ML algorithms: Random Forest (RF), Support Vector Machine (SVM), and k-Nearest Neighbors (kNN), to compare them to the currently used Least-Squares (LS) estimation.
We plan to propose brief theoretical approaches using the estimates from the selected ML model, the LS method and the Kalman Filter (KF) to show different ways of performing position estimation and tracking of the tag. This will be done in the Discussions and Future Work section of Chapter 3.
The differences between the sizes of the datasets that had been created through simulations and that obtained by physical measurements make it somewhat difficult to propose just one ML model. Hence, our work is a comparative study of some ML algorithms. A limitation that we faced was that the data provided to us which was obtained from measurements had sparse number of positions, making it slightly insufficient for ML training. However, experiments were performed to observe the functioning of ML models with such kind of data.

1.3 Literature Survey

The field of indoor positioning is continuously growing and has a lot to explore.The thesis finds its core in the basics of Wireless Communication channels and Bluetooth. This included academic papers about different propagation mechanisms, how 5G systems are different from those belonging to previous cellular generations [4], and the working of and concepts involved in the new version of Bluetooth, Bluetooth Low Energy v5.1. Herein, direction-finding was crucial to be kept in mind as the basic mechanism in order to understand the positioning. Direction-finding in BLE v5.1 is done using Angle of Arrival (AoA) or Angle of Departure (AoD) [3].
We then moved to reading papers about Bluetooth indoor positioning, also including papers that discussed implementation of ML algorithms to improve indoor positioning estimation with Bluetooth. These papers were very helpful in giving us a starting point since they were in line with our thesis topic, and gave us good insight to ML combined with indoor positioning.
To get better acquainted to ML, we decided to study some of the most important concepts related to our tasks. For this, we read some specific chapters from the book *Introduction to Machine Learning* by Ethem Alpaydin.These chapters explained the basics of supervised Machine Learning, which is what we have used in this thesis.
We referred to a paper which talked about combining Kalman Filter and Machine Learning to understand KF for our theoretical proposal [5].

1.4 Thesis Organization

This thesis document consists of four chapters, starting with the Introduction where we give a brief background about the thesis topic, the reasons or motivation why this topic is of interest, the methodology followed, and the reading that we carried out during the thesis as part of research.
Chapter 2 delves deeper into the theoretical concepts that we have used in the thesis to explain each component before we dive into our practical work, which is contained in Chapter 3. We have mainly described our experiments with different ML algorithms in different indoor scenarios and laid out our results.
In the last chapter of the thesis, we elaborate on the observations we make in the previous chapter. We discuss some theoretical approaches in addition to or as modifications of the models we have already created that we think are good options for improving position estimation accuracy using learning algorithms.

Chapter 2

Theoretical Background

2.1 Multiple Antenna Systems

The knowledge of multiple antenna systems has been a topic of major interest over the last three decades [6]. In order to take full advantage of such a scarce resource that the spectrum is, investigation on improving crucial factors in every wireless communication system such as capacity or coverage, led to the utilization of multiple antennas, usually at both sides of the communication link.
Multiple antenna systems offer such a possibility for many different scenarios, and indoor positioning is no exception. Specifically, in every indoor communication link, several inconveniences arise, which are explained in the following sections. With regard to location estimation, knowledge of the DOA, especially for the Line-Of-Sight component, will considerably improve with an increase in the number of antennas.

2.2 Multipath Propagation

Unlike outdoor positioning, where satellite communication is being used and therefore, the location of the users on the ground are in LOS paths to the satellite and are free from scattering, indoor positioning must deal with multipath propagation experienced by any signal transmitted. These different paths that the signal follows are due to different kinds of propagation that arise when waves encounter obstacles such as walls, furniture or even people present in any environment.
Wireless communication channel models commonly take into account different propagation mechanisms which can individually influence the LOS path and the strength of the signal. Therefore, each of them needs to be considered separately for a proper analysis of how the signal fades from the transmitter to the receiver via the wireless channel.
The propagation mechanisms are the following:

- Scattering (Figure 2.1), meaning the signal is spread out in many different directions.
- Reflection (Figure 2.2), referring to how the signal is reflected off a surface.
- Diffraction (Figure 2.3), where the wave is influenced by passing an aperture or an edge.

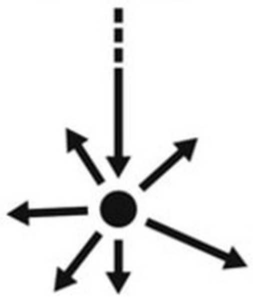


Figure 2.1: Scattering

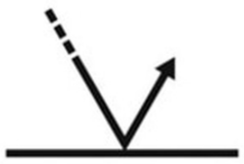


Figure 2.2: Reflection

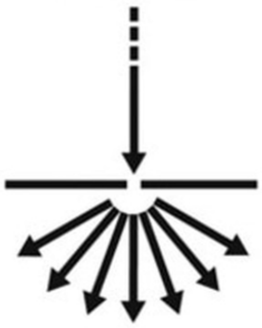


Figure 2.3: Diffraction

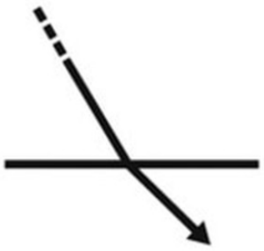


Figure 2.4: Refraction

- Refraction (Figure 2.4), in which the transmission of the signal goes through a different medium at a different angle.

The combination of the different propagation mechanisms mentioned above is what causes the phenomenon known as small-scale fading of the information signal. Multipath propagation forces the signals to echo within the channel. Each of them interacts in various different ways with the environment and spread in different dimensions. These dimensions are delay spread, Doppler spread and angular spread. These have a big impact on the signal and will be explained in the following subsections.

2.2.1 Delay Spread

According to [7], in a multipath environment, the received signal rays arrive in clusters. Several delayed and scaled versions of the signal transmitted arrive at the receiver in different clusters. Within each cluster, each version of the signal decays exponentially in power. In order to be able to distinguish between fading components, and therefore differentiate various clusters, it is crucial to obtain a measure of the coherence bandwidth. The coherence bandwidth is defined by Equation 2.1:

$$B_c \approx \frac{1}{\tau_{RMS}}, \tag{2.1}$$

where τ_{RMS} is the RMS delay spread.
Depending on whether the coherence bandwidth is lower than the bandwidth of the signal, the channel is said to be frequency selective. Hence, different frequency components of the signal transmitted suffer uncorrelated fading.

2.2.2 Doppler Spread

The Doppler spread refers to the movement of the transmitter and/or the receiver and the scatterers' presence in the room, the carrier frequency can spread over a finite spectral bandwidth. If this were the case, the coherence time, T_c in the equation below, provides us with a measure of how fast the channel is changing in time and therefore, be aware of how fast the wireless channel fluctuates. The coherence time is defined by Equation 2.2:

$$T_c \approx \frac{1}{\nu_{RMS}}, \tag{2.2}$$

where ν_{RMS} is the Doppler spread

2.2.3 Angular Spread

Finally, the spatial location of the antenna also matters. Space-selective fading causes the signal power to depend on the spread in AOAs of the MPCs at the receive antenna array or the spread in AODs that arrive at the receiver. Therefore, the coherence distance, or spatial separation, will give a measure of how distant each of the antennas should be from the other in order to be statistically independent. The coherence distance is defined by Equation 2.3:

$$D_c \propto \frac{1}{\theta_{RMS}}, \tag{2.3}$$

where θ_{RMS} is the angular spread.

2.2.4 Azimuth and Elevation (ϕ,θ)

Bluetooth direction-finding is done mainly using two methods: Angle of Arrival and Angle of Departure. Both these methods need one of the two end terminals to have an antenna array.
In the AoA method, which is used in this project, the receiver end includes an antenna array whereas, it is the transmitter end consisting of an antenna array in the AoD method. This is clearly because the AoA and AoD are observed at the receiver and transmitter, respectively. These techniques make use of the phase shifts that occur due to the multiple antenna elements [8]. Figure 2.5 portrays both Bluetooth direction-finding methods.
In this thesis, the AoA information has been used for position estimation. This is because one of the main applications of AoA is the geolocation of Bluetooth devices, such as cellphones. AoA can be measured by the direction of signal transmission which is incident on the antenna array at the receiver, or by the highest received signal strength by rotating the antennas [9]. AoD on the other hand, is

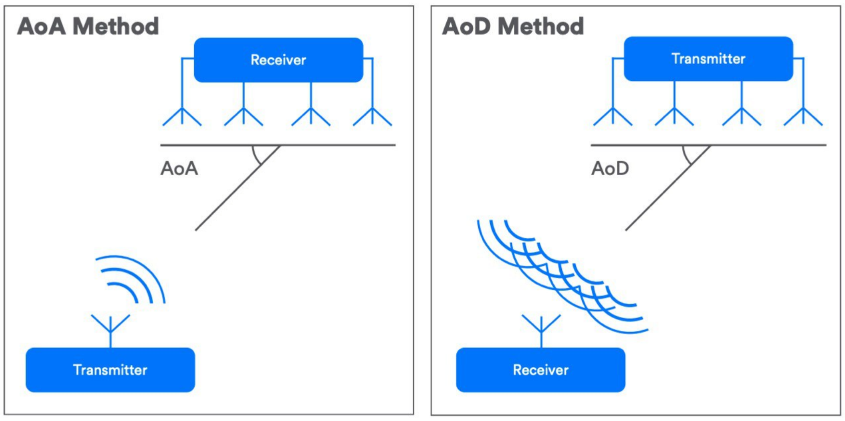


Figure 2.5: Angle of Arrival and Angle of Departure [8]

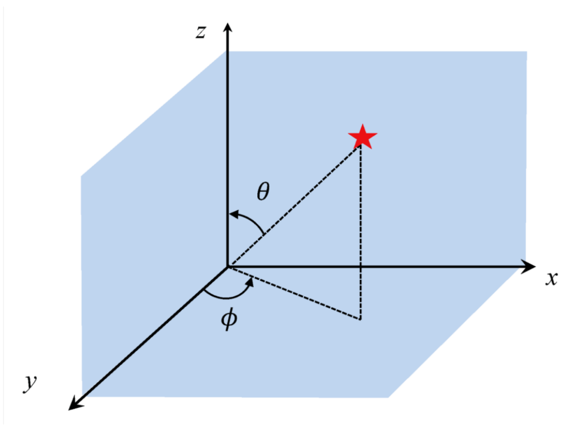


Figure 2.6: Azimuth and Elevation [10]

measured with an antenna array at the transmitter using the phase differences between them.
Both angles actually contain the information about the azimuth and the elevation angles. Azimuth (ϕ) is the angle measured clockwise between North and the observer's horizon (in this case, the observer is an anchor point). Elevation (θ) is the vertical angular distance between the tag and the anchor point's horizon. Essentially, azimuth tells us which direction to look and the elevation tells us how high to look. Figure 2.6 represents both angles in relation to the Cartesian coordinates. We have used both angles to obtain position estimates.

2.2.5 Line-of-Sight

The simulation data that we have used to obtain position estimates was created based on different indoor environments, specifically an office environment with solid structures. These structures and objects have different orientations and sizes. This indicates towards whether there is anything which obstructs the signal transmission between the tag and anchor point in question, which in turn determines if the signal propagation is in Line-Of-Sight (LOS) or Non-Line-Of-Sight (NLOS). LOS propagation means that there is a direct path between the transmitter and receiver. NLOS implies that there are obstructions in the direct LOS path. This implies that the signal propagation has occurred through an indirect and partially-obstructed path, such as through reflections from the various surfaces in the room.

2.2.6 Received Signal Strength Indicator

In addition to the AoA explained above, a correct estimation of the position can be achieved by capturing the power of the received signal. This parameter is known as Received Signal Strength Indicator (RSSI). It is measured in decibels (dB) and it will serve as another input to the ML system.
The indication of how much power the received signal has at every snapshot is a useful measure of how far the position of the user is. Depending on the intensity, a higher RSSI will mean a closer distance and a lower RSSI will mean a further distance between the receiver and the location of the user.
According to [3], distance estimation with RSSI is deficient. Distance estimation has not been attempted during this thesis. However, the more inputs we provide about the position of the tag to the ML system, a more accurate prediction of the final coordinates of the user will be achieved.

2.3 Bluetooth Low Energy v5.1

Bluetooth technology has been used for various types of LBS for many years [3]. In 2019, a new version of Bluetooth called Bluetooth Low Energy (BLE) v5.1 was released, which not only includes previous advantages such as low cost and energy efficiency, but additional new abilities as well.
The new features in the most recent release of this Bluetooth specification mainly focus on the enhancement of indoor positioning capabilities. In the previous versions of Bluetooth, positioning mainly relied on the RSSI in order to provide a proper estimate of the location of a user device, giving a metre-level accuracy. However, BLE v5.1 now also takes into account the new and important direction finding, therefore improving the accuracy of positioning.
Starting with v5.1, an optional feature allows detecting signal direction. To achieve this, it utilizes one of the next two methods: Angle of Arrival (AoA) at the receiver or Angle of Departure (AoD) at the transmitter. In the case of AoA, it is primarily used for Real-Time Location Services (RTLS) (Figure 2.5). On the other hand, indoor positioning systems make use of AoD. However, this thesis attempts to explore the usage of AoA for indoor positioning.
With this new feature, the accuracy of positioning achieved reaches the centimetre level.

2.4 Introduction to Machine Learning

The applications of Machine Learning are found everywhere and are manifold. It is a branch of statistics and computer science in which data is collected and processed in a way in which it can identify patterns and inter-dependencies in the data. The unique thing about ML is that to obtain a result, the system only needs to learn these patterns according to a certain ML algorithm, and there is no specific computation required to do so. The data is generally split into two, the better part of which is usually used to train the system and the remaining part for testing the model that was implemented.
Machine Learning is broadly classified into the following techniques:

- Supervised Machine Learning
- Unsupervised Machine Learning
- Reinforcement Learning

Supervised learning is one in which the system is given a set of data as well as the expected output. This allows the system to have an idea about what the output is supposed to look like. The methods that are most often used are Regression and Classification. In this project, mainly regression has been used. Both methods will be discussed in detail in the following subsections.
Unsupervised learning on the other hand, does not require users to specify dependencies and how inferences are to be drawn from the data. Instead, the system itself performs the tasks of observing patterns in the data, while having some or no knowledge of what the result is expected to be. Examples of unsupervised Machine Learning are Clustering and Principal Component Analysis (PCA).
In reinforcement learning, the system works on maximizing the "reward". Training is not required. Instead, a reinforcement agent decides how the given task can be done. Based on the performance, the system is either penalized or rewarded. In this way, the system makes a chain of decisions based on trial and error to find the best solution.

2.4.1 Regression

Regression is a method which determines the dependencies of a given data with its outcomes. Here, we study the relation between a dependent variable on one or more independent variables. This explains why the main uses of regression are predicting and forecasting. In both cases, we try to predict the state of a particular variable, such as the weather, given that we have information about the factors that the weather depends on, such as temperature, moisture content, etc. It has consistently been a valuable tool in economics and business. For example, annual food expenditure can be predicted based on at least one variable income. It is known that as incomes rise, food expenditure also increases (for clarification, by Engel's law, the expenditure on other things also increases with increase in income, so the total expenditure on food as a share of the total income declines) [11]. Hence, this can be modelled as a linear regression problem. Increasing the number of independent variables makes it a multiple linear regression problem.
Figure 2.7 shows an example of linear regression and explains how regression works

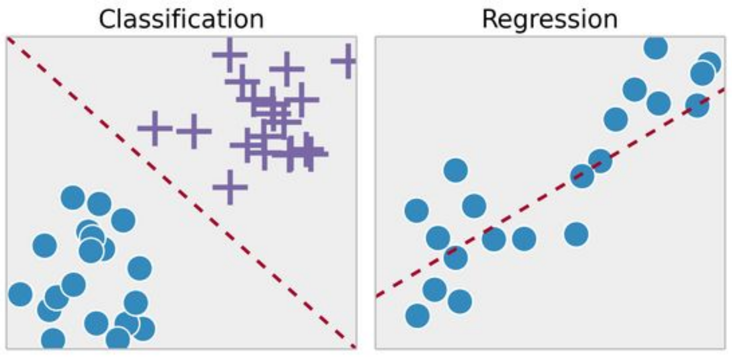


Figure 2.7: Examples of Classification and Regression. [12]

as a concept, simply a line that fits all data points in the best way possible. Regression is categorized based on various factors such as the number of independent and dependent variables, and nature of the regression plot. Popular types of regression are: logistic, polynomial, Bayesian linear, and Ridge.

2.4.2 Classification

Classification works on the idea of grouping things into categories. Based on the dataset provided, the system learns how to classify new observations into predefined categories. It does not exactly perform prediction of any outcomes like regression, but instead, states what the outcome is, and is most useful when there are distinct categories in the data. It is widely used in image, pattern and language recognition, and in scenarios where labelling is required such as categorizing books into genres. Classification is further divided into binary, use cases having two categories, and multi-class, use cases having more than two distinct categories. Often, multi-class problems are a group of binary classifiers.
Figure 2.7 gives us an example of a simple classification problem. A decision boundary exists which decides which class every datapoint gets sorted into.

To elaborate more on this, the data which we worked with during this thesis has decided what kind of supervised Machine Learning we chose. Some parameters showed some correlation with others but the scenario is not one where there were distinct classes into which data could be categorized into. This made it more of a regression problem than a classification problem and hence, we have mainly chosen different regression algorithms.

2.5 Machine Learning Algorithms

2.5.1 k-Nearest Neighbors

The algorithm k-Nearest Neighbors (kNN) a non-parametric (a branch of statistics which does not work only on probability distributions [13]) ML algorithm which was developed as a classification algorithm but is actually used for

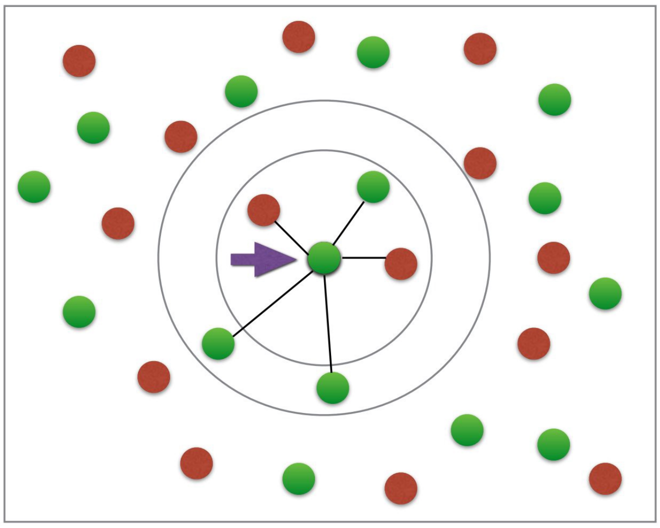


Figure 2.8: k Nearest Neighbors [15]

regression problems as well [14].
This algorithm takes k of the nearest training data samples as input and outputs a class that the observation has been put into if the model is a classifier, or the average of the k nearest data samples if the model is a regressor. As an example, in Figure 2.8 k=5, meaning the data point selects the 5 closest neighbors. The estimated class of that specific data point is determined by the dominant class, green colour in this example. We decided to explore kNN based on [1].

2.5.2 Random Forest

Random Forest (RF) or Random Decision Forest is an interesting ML method which is used for both regression and classification. The principle behind RF is the creation of a specified number of decision trees, a predictive modelling technique in which information about an observation is contained in the "branches" of a tree and the target value in the "leaves". When the target variable takes on discrete values, it implies that the trees are being used for classification, and for regression when the target variable can take on continuous values. Figure 2.9 is a graphic explanation of the algorithm.
While creating a random forest, we generally specify the number of decision trees and the depth until which we want each tree to split into branches. Higher the depth, more the tree will split, and more amount of data will be captured. From [17], we decided to attempt the decision tree approach using RF, especially for classification.

2.5.3 Support Vector Machine

Based on the Vapnik-Chervonenkis theory, Support Vector Machine (SVM) is

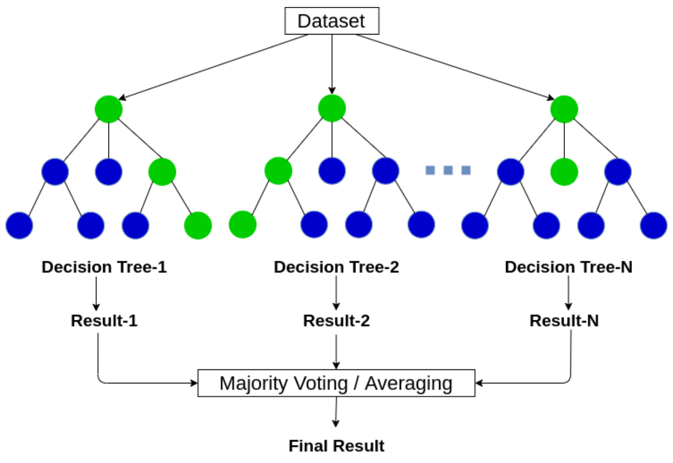


Figure 2.9: Random Forest [16]

a robust ML technique. It is used for a variety of use cases, such as regression, linear and non-linear classification, and even clustering in spite of primarily being a supervised Machine Learning algorithm [18].
SVM uses kernels, which define how the input is read and transformed into a specific form. Kernels help us find a hyperplane in high-dimensional data. This hyperplane separates two data classes in case of Support Vector Classification and creates a decision boundary in case of Support Vector Regression [19]. The widely used types of kernels used in SVM are: linear, radial basis function (RBF), polynomial, and sigmoid.The principle of SVM is graphically explained in Figure 2.10. It shows the hyperplane separating two classes of data with the help of support vectors.

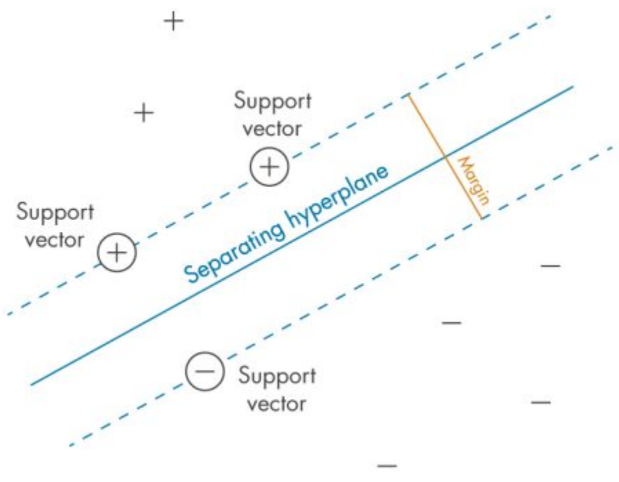


Figure 2.10: Support Vector Machine [20]

2.5.4 Computational Complexity

The term complexity in the Machine Learning context tells us how many iterations are required to run the training and how much time these iterations take based on the number of training samples and number of features. Similarly, it also tell us how long the prediction takes based on the parameters that are passed to the algorithm.
Ensemble type learning methods use multiple algorithms, hence the term "ensemble" [21]. An example of this is Random Forest. Let us say that the original complexity is x. Because of multiple decision trees (say n), this complexity is multiplied by the number of trees and becomes nx.
For Support Vector Machine, the complexity follows a similar path except for the role of the kernel being used [22].
The training in k Nearest Neighbors involves calculating the distance between the data point in question and all others in the dataset. If there are n samples in the dataset, the number of iterations required for training will also be n.
Just as the most suitable algorithm changes from application to application, so does the acceptable time and computational complexity. Some require fast results, thus requiring low complexity algorithms. Often in some applications, generally where the size of data and complexity of the application itself is more, the high time and complexities are excused in order to obtain accurate results. On the other hand, in scenarios where highly accurate results are not expected, we can afford sacrificing some of the complexity and accuracy.
Computational complexity is commonly denoted by the Big O notation. It tells us how many resources, such as time and space, are required to run a particular algorithm. Take for example a list with n elements. The worst-performing search algorithm would be one which runs through every element once, which would also be the simplest search. This means that the complexity of this algorithm is $O(n)$. To explain it further, say the element we are searching for is the first element of the list. This gives us the best-case scenario $O(1)$ [23]. Expressed using the Big O notation, in Table 2.1 we can find the computational complexities of the three algorithms we have experimented within this thesis. Note that n is the number of samples, p is the number of features, t is the number of trees (for RF), and n_sv is the number of support vectors (for SVM).

Algorithm	**Training**	**Prediction**
Random Forest	$O(n^2pt)$	$O(pt)$
Support Vector Machine	$O(n^2p+n^3)$	$O(n_{sv}p)$
k Nearest Neighbors	$O(1)$	$O(np)$

Table 2.1: Computational Complexity for each algorithm. [22]

2.6 Least-Squares

The positioning engine that is being used at the company currently is based on the Least-Squares method, as mentioned before. This method is a popular and effective way of minimizing the error in linear regression models, where it minimizes the sum of the squares of the difference between the true and predicted value, as shown in Equation 2.4:

$$S = \sum_{i=1}^{n} (p_i - \hat{p_i})^2, \tag{2.4}$$

where p_i and $\hat{p_i}$ are the actual value and the value predicted by the model, respectively. The outcome S refers to the sum of the squares of the mentioned difference. However, the positioning engine does not use linear regression. Instead, the best-fit is computed using the Moore-Penrose inverse. This is essentially the pseudoinverse of a matrix and is more formally defined as the generalization of an inverse matrix, i.e., the square matrix $\mathbf{P}$ is invertible if there is another matrix $\mathbf{Q}$ that fulfills the criterion. Equation 2.5 shows the above mentioned mathematically:

$$\mathbf{PQ} = \mathbf{QP} = \mathbf{I}_n, \tag{2.5}$$

where n is the dimension of both these square matrices, and I_n is an identity matrix of n x n dimensions.
In other words, if the matrix is invertible, the pseudoinverse equals the matrix inverse. The Moore-Penrose inverse is useful for solving a large number of least squares in a system [24]. In Equation 2.6

$$\hat{p}_{LS} = (\mathbf{H}^T\mathbf{H})^{-1}\mathbf{H}^T\mathbf{B}, \tag{2.6}$$

the matrices $\mathbf{H}$ and $\mathbf{B}$ contain information about the direction of the target with respect to the local coordinate system of each anchor and the location of the target.Local coordinate system refers to the set of x,y and z coordinates that are associated with each component of the system, in this case, the anchor points. Each anchor point has a local coordinate system which describes its location and orientation.

Chapter 3

Positioning Estimation with Machine Learning

The processing of the input data to the Machine Learning system is key for proper estimation of the positions. The goal of our project is to predict future positions accurately, and to perform such predictions we have decided to use inputs such as AoAs in azimuth and elevation (in degrees), RSSIs, and true Cartesian coordinates of the tag.
As Figure 3.1 portrays, the AoAs, together with RSSI, are provided from four different anchor points. Each of these anchors are placed in different regions of an office environment, either simulated or real. Therefore, these four anchors experience different path loss between themselves and the position of the tag. Additionally, together with the ground truth coordinates provided by u-blox, we have sufficient inputs to start the computation of the estimates of the future positions. During this chapter, the outcome of our work will be shown. To give a better understanding of our project and its placement scenarios, floor plans of both simulated and real-life environments will be available for the reader.

3.1 Arranging Testbench Data

All datasets we were provided with are a compound of subsets of data, each of them corresponding to the values obtained during the simulations run at each of the three BLE advertising channels and with different polarization at the anchor side. Channels 37 (2402 MHz), 38 (2426 MHz) and 39 (2480 MHz), together with horizontal and vertical polarization of the antennas at every anchor, make up six subsets of available data for the training of ML models for positioning.
Every subset of data within a testbench (testing environment) has very similar values. Consequently, any subset could have been selected. Our choice was based on the highest average RSSI among those received when the anchor points were oriented in both horizontal and vertical polarization, and through all three channels. Channel 37 with horizontal polarization of the antennas is the environment we worked with for all experiments we did during the thesis.
The arrangement of the data is key for a good performance of every ML algorithm. Our case belongs to supervised learning, where the algorithms are trained using labeled examples, such as an input where the desired output is already known.

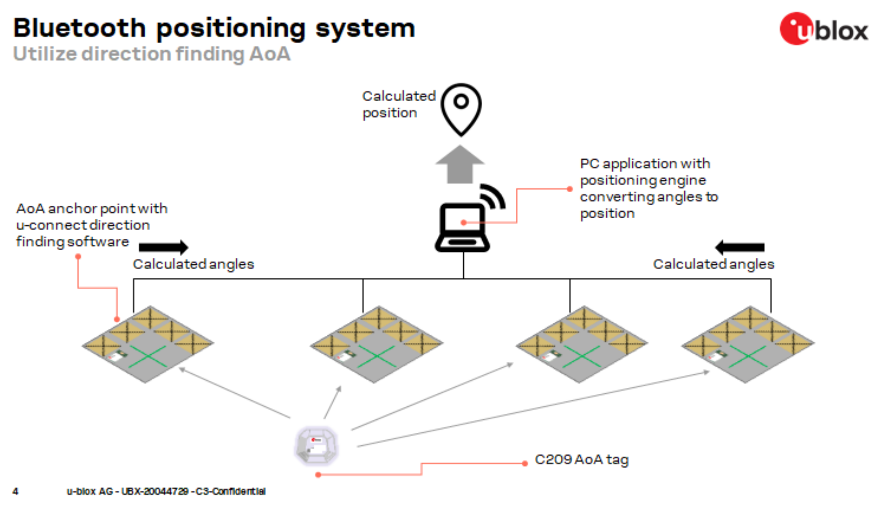

Bluetooth positioning system
Utilize direction finding AoA
u-blox
Calculated position
PC application with positioning engine converting angles to position
AoA anchor point with u-connect direction finding software
Calculated angles
Calculated angles
C209 AoA tag
4
u-blox AG - UBX-20044729 - C3-Confidential

Figure 3.1: Bluetooth positioning scheme performed at u-blox.

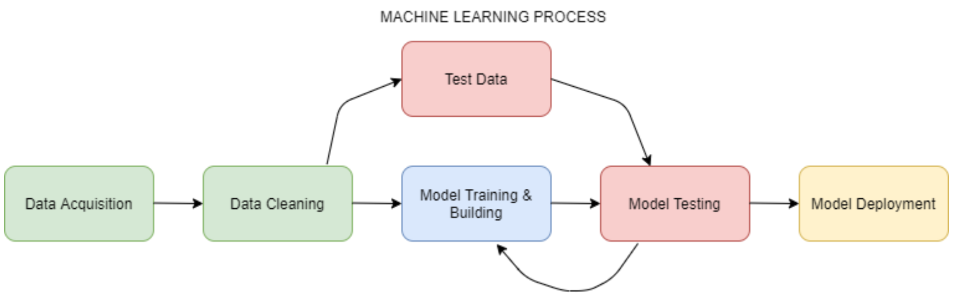


Figure 3.2: Scheme of a Machine Learning process.

We modify the model accordingly by adjusting parameters to achieve the best performance possible.
As depicted in Figure 3.2, the first step is to actually get the data. The process of data acquisition has been described above. Once we actually acquire the data, the next step is to decide how to arrange our data for our models to be able to process it. In our case, the first decision was to include information about each of the four anchors. Considering that each of the anchors suffers from different path losses, the information of each anchor needs to be treated separately in order to provide a proper accuracy of what we are going to predict subsequently. Therefore, the columns (features to the ML model) from every simulated dataset provide information about the azimuth and the elevation angles and the RSSI, all at every snapshot taken. The rest of the columns are the true coordinates of the position of the tag. Therefore, either the scenario is simulated or real, the inputs to our ML system will consist of 15 features, corresponding to 8 AoAs (in azimuth and in elevation), 4 RSSIs and 3 for Cartesian coordinates (x, y and z).

Furthermore, we split the data into training and testing data. Here, we take some portion of the data, usually between 50% and 80% of the total for the training data and the rest for the testing data. We will delve deeper into this subject in later sections. Following the diagram from Figure 3.2, we use that specific training set in order to fit a model to it. As a way to comprehend how well our model actually performed, we run that test data through the model and compare the prediction of the model to the true labels of the test data.

3.2 Parameter Optimization

Every ML algorithm has parameters that define how it predicts. The same algorithm might require different parameters, which differ based on the size of the data and the patterns in the data. Selecting the optimum parameters is known as hyperparameter optimization.
Since each of the test benches we worked with were different in various ways which are explained in the next section, we performed a grid search to find the best model for the learning to be done with the test bench in question. This was done using GridSearchCV in the model_selection library in scikit-learn. A grid is created by specifying which parameters we want to optimize, along with possible values that we are interested in exploring. For this thesis, we ran a grid search for each of the three algorithms. The parameters optimized and used are listed in Table 3.1 and selected values in table 3.2.

Algorithm	**Parameter**	**Range**
RF	n_estimators, max_depth	[1,100],[1,10]
SVM	kernel	[rbf,linear,sigmoid,poly]
kNN	n_neighbors	[1,5]

Table 3.1: Parameter values used for Optimization.

Algorithm	**Classifier**	**Regressor**
RF	n_estimators=40, max_depth=4	n_estimators=40, max_depth=5
SVM	kernel=rbf	kernel=rbf
kNN	n_neighbors=1	n_neighbors=2

Table 3.2: Optimized parameters.

We created a base model for the algorithms and performed the grid search based on this grid, and then fit the different datasets to this search. As we would expect, the optimum parameters were different for each dataset. The results of the grid searches varied slightly with different datasets. To keep the implementation as general as possible, we selected middle-ground parameters for similar algorithms. For example, say we are performing a grid search for the optimum number of trees (*n_ estimators*) for RF. The optimum number of trees by grid search for dataset A was 40, and that for dataset B was 50. We select 40 because the algorithm performs similarly with both 40 and 50 trees, and selecting the option with lesser trees would also mean that less time is taken for computation, as seen in Figures 3.3(a), 3.4(a) and 3.5(a).

A similar case was observed for the best kernel for SVM; the grid search returned *linear* kernel for some datasets but *rbf* (Radial Basis Functions) for most. The *rbf* kernel works well with data that cannot be divided simply by finding a straight line to separate it, meaning that it is not linearly separable. The *rbf* kernel finds a hyperplpane by computing the distance between all data points and one single point, which is referred to as the centre [25]. This is depicted in Figures 3.3(c), 3.4(c) and 3.5(c).

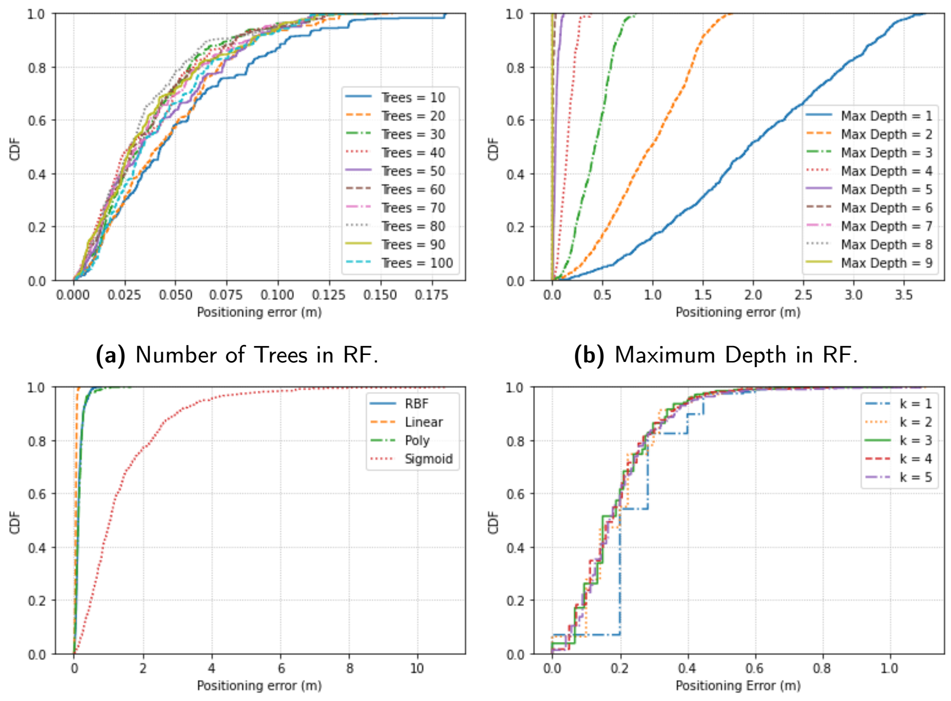
It is safe to assume that in a real-world situation, the movement pattern of a Bluetooth tag (the user) is just as likely to be random or haphazard as walking in a straight line. Hence, in these scenarios, using the *linear* kernel might not necessarily imply correct estimation. Therefore, we selected the *rbf* as the more realistic option. The grid search is a simple and effective way of finding the parameters that suit your dataset the best, which consequently gives you a better performing algorithm. The plots shown in Figures 3.3, 3.4 and 3.5 tell us the performance of ML algorithms with various parameters.

Tables 3.3 to 3.5 give us an idea about how much time is taken to train each ML algorithm as a regressor, as well as the time taken for each algorithm to compute estimates or to test based on new observations. We have used only the X-coordinate estimations as an example of the different training and testing sizes. These tables show the time taken for training to be performed by each algorithm and the time taken for testing to be complete. Along with this information, they also includes the inference time for each model, which is the average time taken for one training or testing iteration to be run. Only regression (performed by all three algorithms) has been used to display these times because this method seems to be the most likely to be used in real-world positioning situations.

In both cases, RF takes the most time to train. The reason for this is because there are two parameters *n_ estimators* and *max_ depth* that have been used, unlike for the other two algorithms, where only one parameter has been passed. The number of trees used in RF lead to more computational time.

The testing time taken by kNN is exceptionally low as compared to SVM and RF. This gives kNN an added advantage. If a system as this one is implemented in a real-world scenario, it will be pre-trained and the important aspect to consider would be the amount of time this pre-trained system takes to estimate positions based on new measurements. This means that kNN would take the least time to perform estimation in a real-time situation.

Judging from the three tables, we decided to split the data equally, i.e., 50% for training and 50% for testing. This was to maintain a better balance, because the difference between the testing times with varying test sizes is negligible; approximately 9ms for RF, 25ms for SVM, 4.5ms for kNN. However, with respect to the training time, the size of the training data makes a noticeable difference, especially for SVM in Table 3.4. Here, we can see a gap of about 35ms comparing 80% and 50% training data size.



(a) Number of Trees in RF.

(b) Maximum Depth in RF.

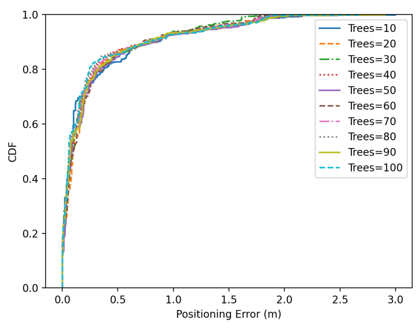
(c) Kernel in SVM.

(d) Number of Nearest Neighbors in kNN.

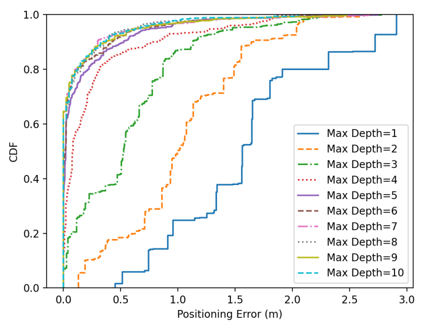
Figure 3.3: Different Parameters in the Simulated Scenario: Regression

Train Size Test Size	Train Time (s) Test Time (s)	Inference Times (s)
80%, 20%	0.0988, 0.0087	$1.57\text{x}10^{-4}$, $1.38\text{x}10^{-5}$
70%, 30%	0.1086, 0.0092	$1.72\text{x}10^{-4}$, $1.46\text{x}10^{-5}$
60%, 40%	0.0915, 0.0080	$1.45\text{x}10^{-4}$, $1.28\text{x}10^{-5}$
50%, 50%	0.0899, 0.0118	$1.42\text{x}10^{-4}$, $1.87\text{x}10^{-5}$

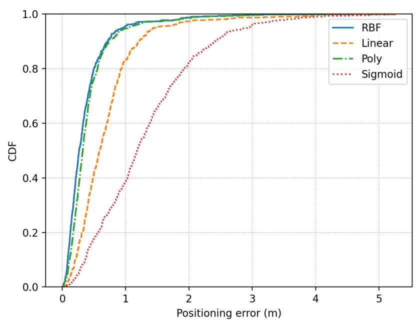
Table 3.3: Training and Testing Times: Random Forest.



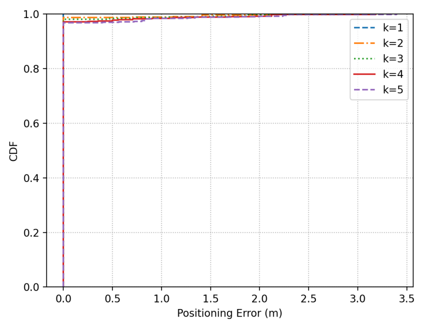
(a) Number of Trees in RF.



(b) Maximum Depth in RF.



(c) Kernels in SVM.

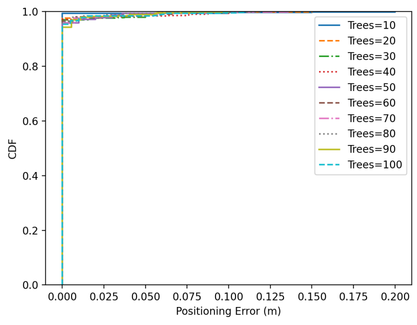


(d) Number of Nearest Neighbors in kNN.

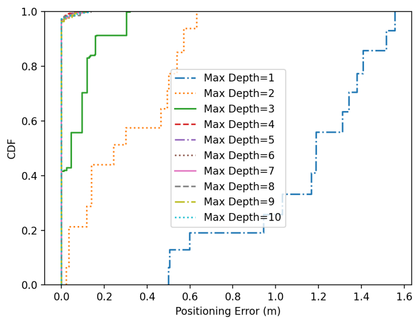
Figure 3.4: Different Parameters in the Real (Measured) Scenario: Classification.

Train Size Test Size	Train Time (s) Test Time (s)	Inference Times (s)
80%, 20%	0.0434, 0.0213	$6.89\text{x}10^{-5}$, $3.38\text{x}10^{-5}$
70%, 30%	0.0308, 0.0258	$4.89\text{x}10^{-5}$, $4.09\text{x}10^{-5}$
60%, 40%	0.0236, 0.0297	$3.75\text{x}10^{-5}$, $4.71\text{x}10^{-5}$
50%, 50%	0.0174, 0.0311	$2.76\text{x}10^{-5}$, $4.94\text{x}10^{-5}$

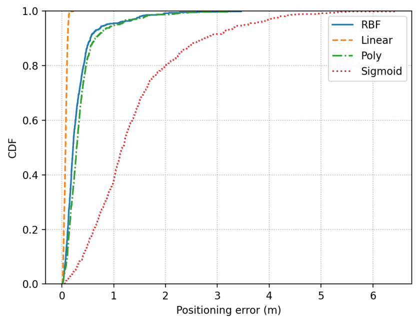
Table 3.4: Training and Testing Times: SV Regressor.



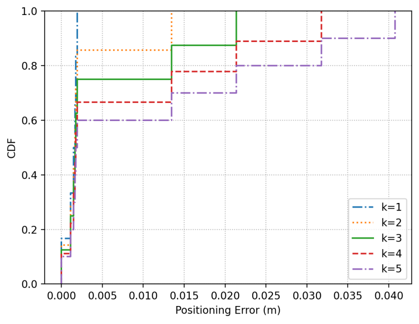
(a) Number of Trees in RF.



(b) Maximum Depth in RF.



(c) Kernels in SVM.



(d) Number of Nearest Neighbors in kNN.

Figure 3.5: Different Parameters in Real Scenario: Regression.

Train Size Test Size	**Train Time (s) Test Time (s)**	**Inference Times (s)**
80%, 20%	0.0028, 0.0042	$4.45\text{x}10^{-6}$, $6.67\text{x}10^{-6}$
70%, 30%	0.0024, 0.0045	$3.80\text{x}10^{-6}$, $7.11\text{x}10^{-6}$
60%, 40%	0.0023, 0.0050	$3.70\text{x}10^{-6}$, $8.05\text{x}10^{-6}$
50%, 50%	0.0023, 0.0053	$3.73\text{x}10^{-6}$, $8.47\text{x}10^{-6}$

Table 3.5: Training and Testing Times: kNN Regressor.

	Wood	**Concrete**
Rel. Permittivity	3	6
Rel. Permeability	1	1
Conductivity [S/m]	0.022	0.078

Table 3.6: Dielectric Properties of Wood and Concrete.

3.3 Estimations in Different Indoor Environments

Given the optimal parameters for the best performance of each of the ML algorithms, it is time to put those concepts into practice by applying them on different indoor scenarios. Here, the contexts vary depending on different factors which can be found in every real-life indoor environment. In total, we were given seven different simulated environments, which are depicted in the Figures 3.6 and 3.7:

- Testbench_01: Each of the four anchors are in LOS with the tag position. No obstacles present in the room obstruct the signal (see Figure 3.6).
- Testbench Furniture_High: As depicted in Figure 3.7(a), in this indoor scenario there are many MPCs due to objects made of wood which obstruct the LOS path.
- Testbench Furniture_Mid: A few objects, also made of wood, can interfere with the communication between the four anchors and the tag (see Figure 3.7(b)).
- Testbench Furniture_Low: Indoor scenario with only one high object present in the room (depicted in Figure 3.7(c)).
- Testbench Furniture_High_Concrete: Similar to the testbench furniture high, but those objects are made of concrete instead of wood.
- Testbench Furniture_Mid_Concrete: A few objects made of concrete are present in the room.
- Testbench Furniture_Low_Concrete: Only one high object made of concrete is in the indoor environment.

As explained above, the main difference between each of the scenarios is the amount of objects present in the room. Plus, within those objects, the material in which they have been made of can be either wood or concrete, which is explained after, can have a big impact on the path loss of the signal transmitted. That impact is due to the frequency dependent properties of the material, which are resumed in Table 3.6.
Emphasizing on those properties where there is a big gap between wood and concrete, both the relative permittivity and the conductivity of the materials clearly need to be taken into consideration. The relative permittivity refers to the factor by which the electric field between the charges of a material is decreased relative

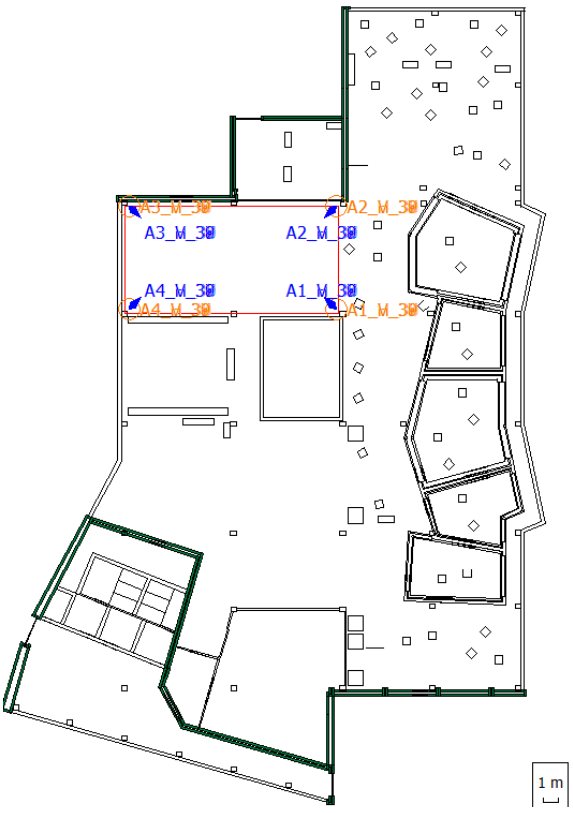
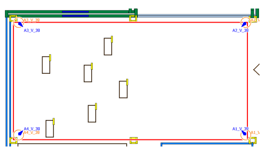
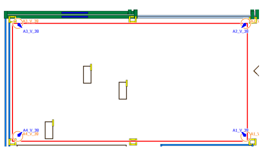


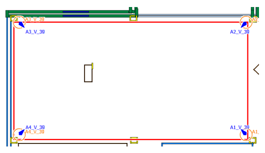
Figure 3.6: Floor Plan of an Office Environment.



(a) High Density.



(b) Mid Density.



(c) Low Density.

Figure 3.7: Simulated Environments.

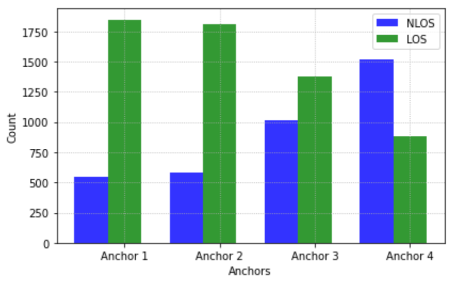


Figure 3.8: LOS Distribution for High Concrete/Wood.

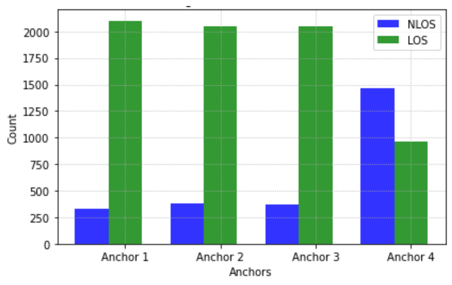


Figure 3.9: LOS Distribution for Mid Concrete/Wood.

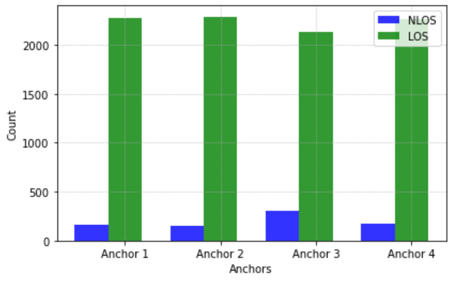


Figure 3.10: LOS Distribution for Low Concrete/Wood.

to the vacuum. The conductivity measures how much an electric charge can pass through a material. Therefore, judging by Table 3.6, one can expect that concrete scenarios are more prone to errors in communication.
Taking a look at the input data for our ML training models, an important aspect to take into account when the amount of obstacles in a room increases is the strength of the received signal. Additionally, we were provided with the LOS components of each simulated scenario in order to observe the amount of snapshots in which the tag is in LOS with each anchor. Figures 3.8-3.10 portray the distribution of LOS and NLOS components in each testbench (Testbench 01 is discarded from plotting due to its full LOS scenario). Figures 3.11-3.14 portray the average RSSI at every snapshot.

As expected, a large number of obstructions make the path between the tag and any of the anchors less direct, thus increasing the NLOS components. The distribution of LOS components for wooden scenarios has been discarded from the plot because of its similarities with the concrete scenarios.
When we discuss the RSSI feature, the strength of the received signal depends not only on the path between the tag and the anchor, but also on the dielectric properties of both materials. Figures 3.11 and 3.12 show how the signal power decays with the amount of obstacles present. Note how the levels of power also change for wooden and concrete materials, where the latter, due to its higher density, shows more drastic power drops. Figures 3.13 and 3.14 compare the average wooden and concrete scenarios with the LOS testbench. A decrease in

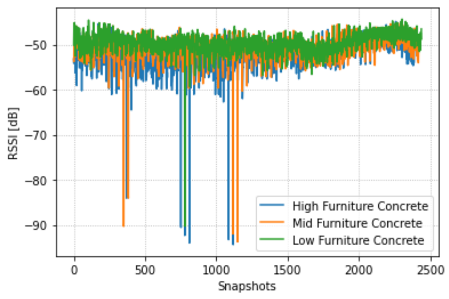


Figure 3.11: RSSI for Concrete Scenarios.

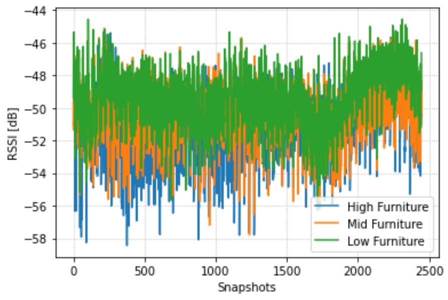


Figure 3.12: RSSI for Furniture Scenarios.

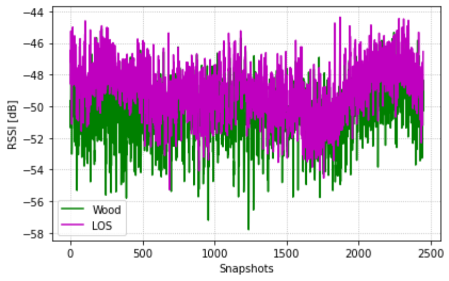


Figure 3.13: LOS vs. Furniture Scenario.

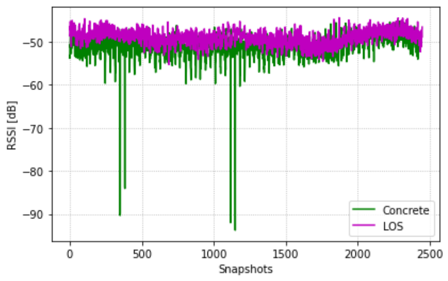


Figure 3.14: LOS vs. Concrete Scenario.

signal power can be perceived with the presence of furniture in the room.

3.3.1 Error Performance Criteria

Regression is a method used when a model attempts to predict continuous values. Here, we are trying to predict future Cartesian coordinates of the position of a tag, which is why regression seems to be the correct strategy. This ML technique is characterized by the way it comprehends the relationship between one dependent variable and one or more independent variables. Hence, in this thesis we treated the X- and Y-coordinates as two separate dependent variables. The Z-coordinate was discarded for estimation since all the inputs had the same value. When it comes to measuring the performance of regression models, there are various evaluation metrics which can be carried out [26]. In the following two subsections, two evaluation methods are going to be described for a proper understanding of our error performance criteria.

Mean Absolute Error

One of the most common evaluation metrics is the Mean Absolute Error (MAE)[27]. As seen in Equation 3.1, we compare the predictions with the true values by taking the difference between the two. Technically, our predictions could be positive or negative value, which is why it is necessary to take the absolute value. Finally, we compute the average of these errors.

$$\frac{1}{n}\sum_{i=1}^{n}|p_i - \hat{p_i}|, \tag{3.1}$$

where p_i refers to the true coordinate (either x or y) and $\hat{p_i}$ refers to the predicted coordinate.
As mentioned before, the MAE has been calculated for both X- and Y-coordinates separately. Tables 3.7-3.9 show the MAEs for every ML algorithm at each testbench:

	X-Coordinate (m)	**Y-Coordinate (m)**
High Concrete	0.043	0.012
Mid Concrete	0.053	0.009
Low Concrete	0.040	0.010
Testbench 01	0.048	0.010
High Furniture	0.043	0.009
Mid Furniture	0.047	0.009
Low Furniture	0.043	0.010

Table 3.7: MAE for x and y using RF.

	X-Coordinate (m)	**Y-Coordinate (m)**
High Concrete	0.120	0.073
Mid Concrete	0.116	0.076
Low Concrete	0.118	0.075
Testbench 01	0.112	0.075
High Furniture	0.115	0.074
Mid Furniture	0.119	0.074
Low Furniture	0.115	0.074

Table 3.8: MAE for x and y using SVM.

Euclidean Distance

A different approach to computing the positioning error is the Euclidean Distance, which calculates the distance between two points, as shown in Equation 3.2:

$$d(p_i, \hat{p_i}) = \sqrt{(x_i - \hat{x_i})^2 + (y_i - \hat{y_i})^2}, \tag{3.2}$$

where p_i is the true position and $\hat{p_i}$ the estimated position, both at snapshot i.
The result is a vector of positioning errors between the predicted x and y coordinates and the true values of x and y. In Table 3.10 we can observe the average

	X-Coordinate (m)	**Y-Coordinate (m).**
High Concrete	0.143	0.111
Mid Concrete	0.132	0.103
Low Concrete	0.126	0.095
Testbench 01	0.115	0.092
High Furniture	0.121	0.099
Mid Furniture	0.134	0.105
Low Furniture	0.121	0.099

Table 3.9: MAE for x and y using kNN.

	RF	**SVM**	**kNN**
High Concrete	0.048m	0.154m	0.204m
Mid Concrete	0.055m	0.151m	0.189m
Low Concrete	0.044m	0.153m	0.178m
Testbench 01	0.050m	0.148m	0.166m
High Furniture	0.045m	0.150m	0.175m
Mid Furniture	0.050m	0.154m	0.193m
Low Furniture	0.047m	0.150m	0.175m

Table 3.10: Mean Euclidean Distance.

Euclidean Distance for every testbench and for each of the ML algorithms in consideration.

3.3.2 Error Distribution

The Euclidean Distance, together with the CDF, is a useful tool in order to comprehend the distribution of the positioning error. The CDF of the error tells us how those are distributed across the error vector and it gives us a proper understanding and further analysis of how each algorithm performs at each simulated indoor scenario.
The procedure is to examine how each algorithm performs in the worst scenarios. Therefore, each algorithm has been applied to indoor environments with a large amount of obstacles present, regardless of the material. In order to give a proper comparison of each algorithm, the LOS scenario, that is, the ideal environment, is also present in the following CDF plots. The plot of the density of errors can be observed in Figures 3.15, 3.16, 3.17 and 3.18.

Considering the aforementioned, we deem a more dense environment as a more suitable indicator to real-life indoor scenarios. In order to provide a good overview of the positioning error performance, the ideal scenario has also been visualized along with the other two scenarios. From Figures 3.15, 3.16 and 3.17 we can

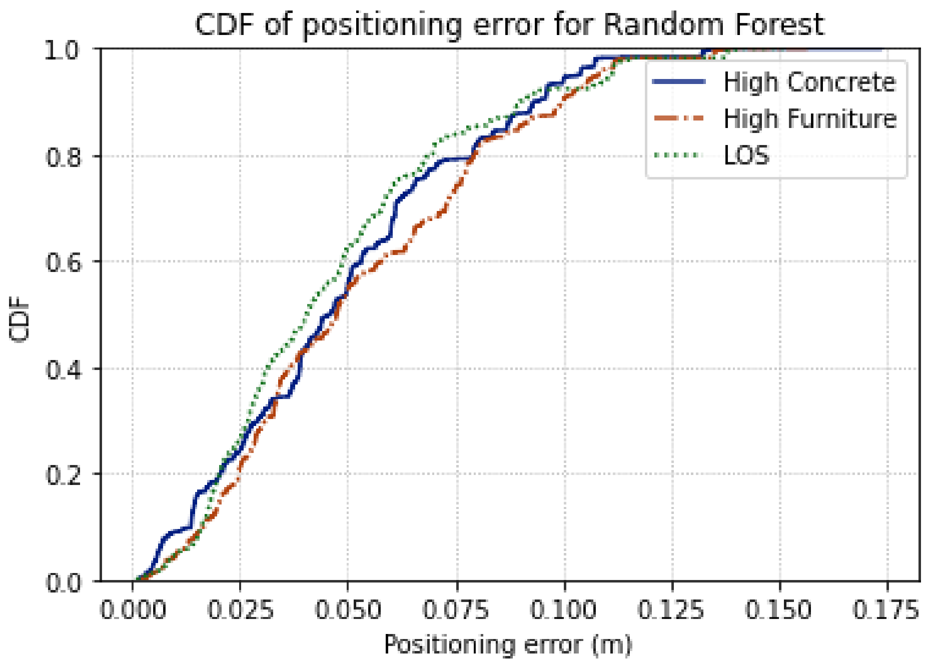


Figure 3.15: Error Distribution for RF.

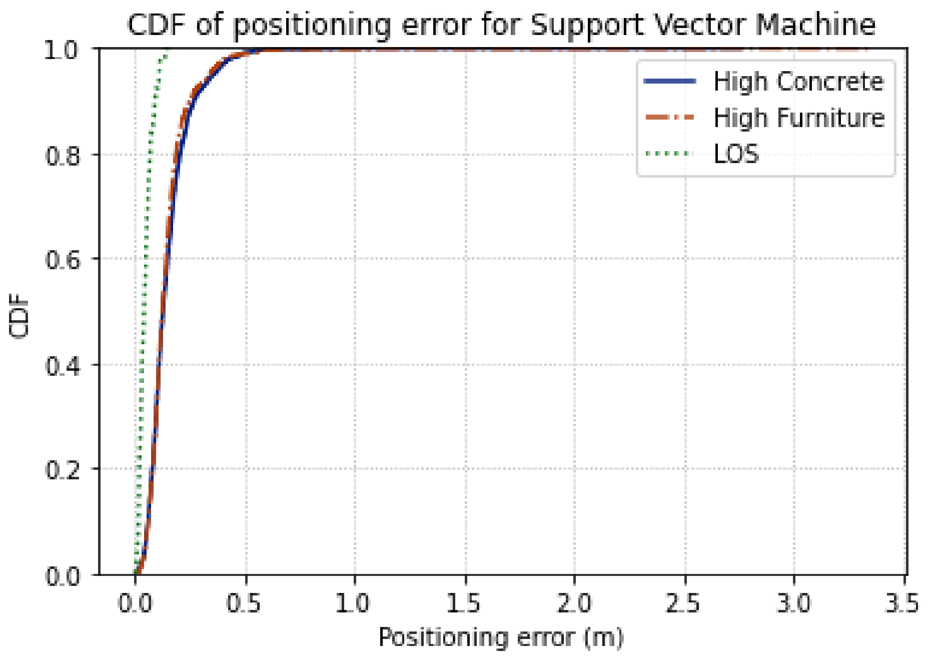


Figure 3.16: Error distribution for SVM.

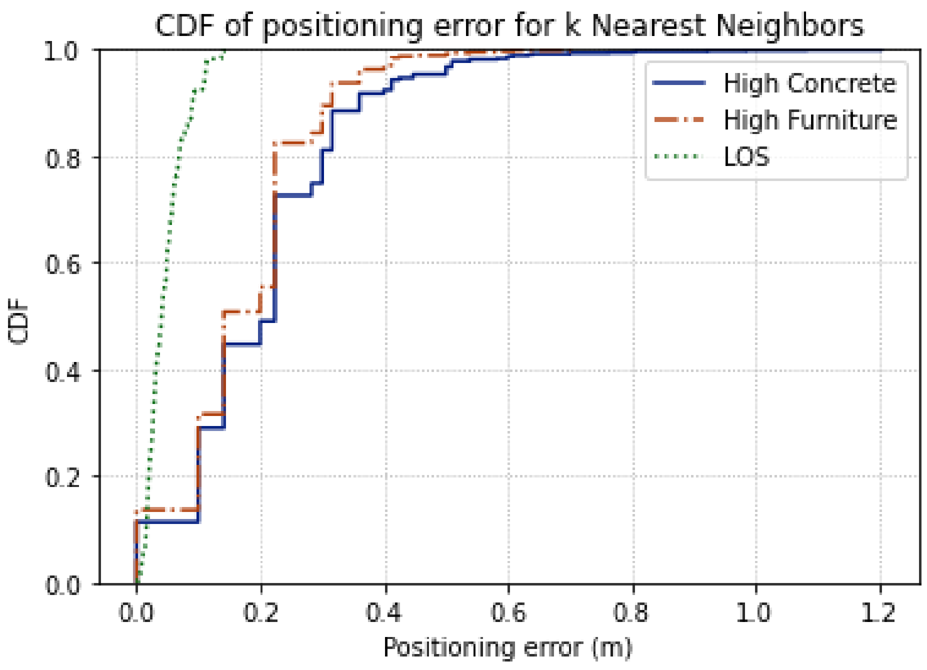


Figure 3.17: Error Distribution for kNN.

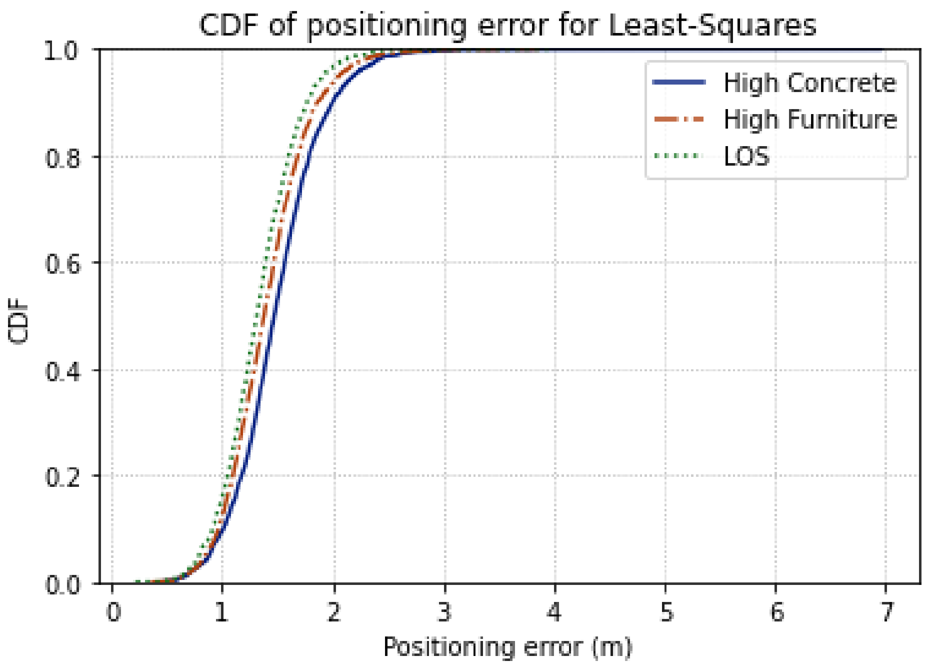


Figure 3.18: Error Distribution for LS.

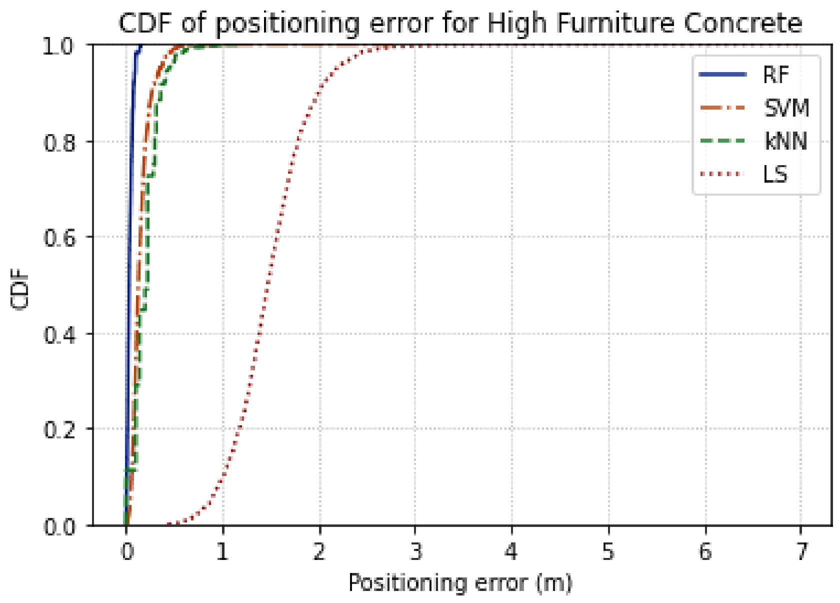


Figure 3.19: Comparison of Error Distribution in a Concrete Scenario.

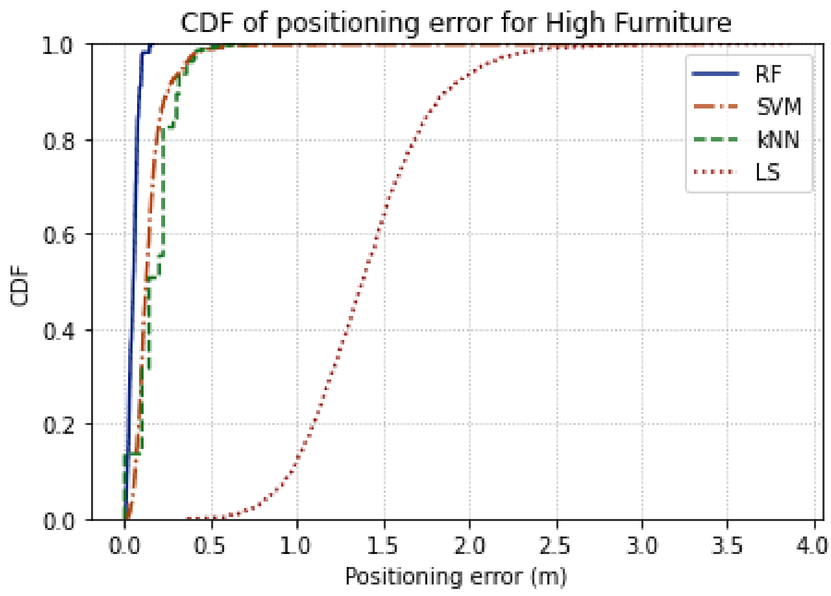


Figure 3.20: Comparison of Error Distribution in a Furniture Scenario.

observe that 95% of the positioning errors are below approximately 0.1m for LOS. We use this value as a benchmark to compare the efficiency of the three algorithms on the other two scenarios. On the other hand, from Figure 3.18, LS gives 95% of the errors below 1.8m.
From Figure 3.15, we can see that 95% errors using RF are below 0.11m for concrete and below 0.1m for wooden obstacles. Observing the next figure where SVM is used, the 95% of errors give very similar results, where the errors are below 0.3m. Lastly, from Figure 3.17, we can see that kNN gives slightly different results to the other two algorithms. The 95% of errors are below 0.4m.
Judging by these results, it is clear that RF outperforms SVM and kNN, with a much lower range of errors. If we refer to Section 3.2, we have talked about how the time taken for training different models is different for varying sizes of the training dataset. Given the training size, number of trees (40) and maximum depth (5), the time taken for training the RF regressor, although higher than the other two algorithms, can be excused because of its greater performance.
To summarize the above discussion, Figures 3.19 and 3.20 clearly show how RF can be the better choice in a real-world indoor scenario among the three ML algorithms studied. From these figures, we can observe how LS clearly underperforms as compared to the other three.

3.4 Cross-prediction with Simulated Indoor Environments

Cross-prediction is a method in which testing is done using data which is not a part of the data used for training, also called sample testing. The information given by the cross-prediction method is helpful to understand how well the model we have trained in a specific indoor environment performs when new observations are given to it.
As mentioned in the previous sections, we worked with 7 testbenches that correspond to different indoor environments with varying parameters and structures made of concrete and wood. Cross-prediction was performed for every combination of datasets. For example, we trained the RF, SVM, and kNN regressors with the all-LOS testbench and predicted position estimates by testing this model with itself and all the other six testbenches. The result is therefore a 7x7 matrix for each algorithm utilized consisting of average positioning errors.
The next step is to give a proper visualization of the outcomes of cross-prediction. To portray the mentioned matrices, we make use of a heatmap from the seaborn library of Python. A heatmap suitable by depicting the correlation between two variables in a more detailed fashion.
The values mapped into the heatmap also show us the patterns in the data if any. Figures 3.21-3.23 display the cross-prediction results as a function of the positioning errors for each of the ML algorithms. The colorbar on the right-hand side indicates the range of the positioning errors.
From the heatmaps, we can see that when the models are trained using the worst-case scenario, Furniture_High_Concrete, all three algorithms give good results, most noticeably RF.
We can also observe that when the models are trained by the LOS testbench,

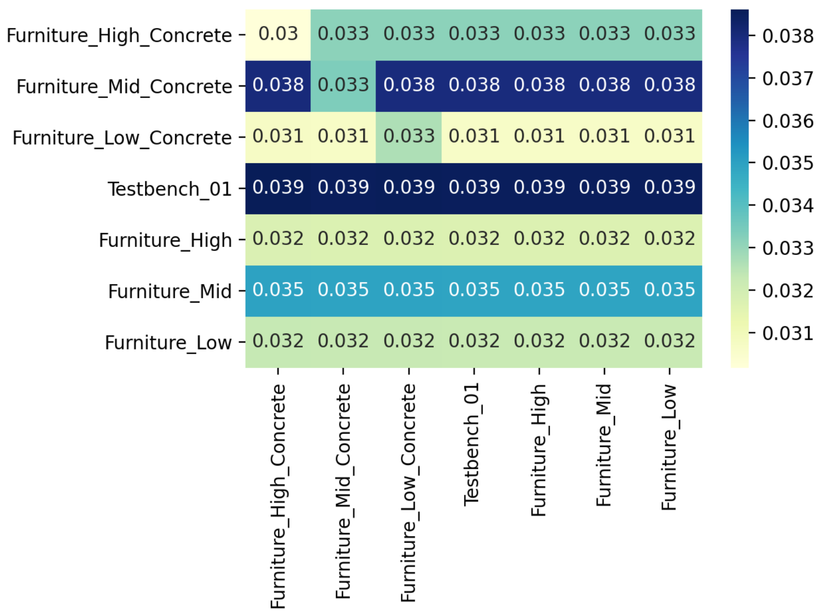


Figure 3.21: Cross-Prediction with RF Regressor.

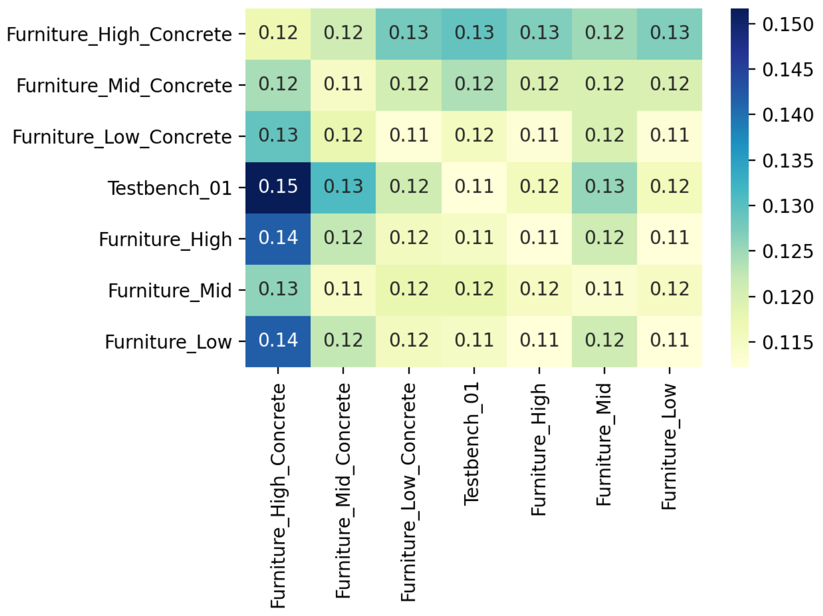


Figure 3.22: Cross-Prediction with SV Regressor.

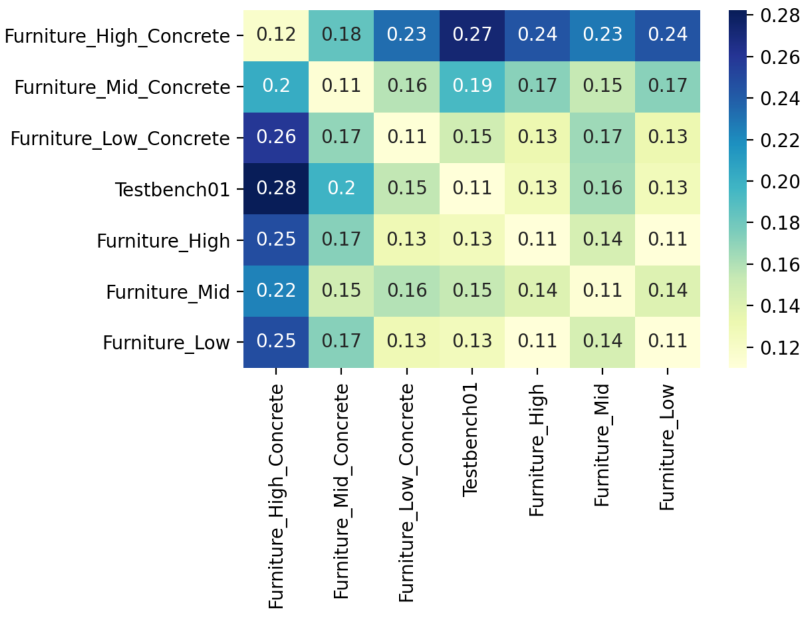


Figure 3.23: Cross-Prediction with kNN Regressor.

the cross-prediction errors are higher. This could be because in this environment where all paths are LOS, this testbench is something of an “ideal” indoor scenario. All other testbenches have at least some NLOS components, making them less ideal. The model trained with LOS testbench is not aware of worse cases of indoor environments, such as the High Concrete and High Furniture scenarios. This can be directly related to the fact that wood is less dense than concrete, complying with the dielectric parameters used, especially relative permittivity and conductivity, when simulations were run to obtain these testbench datasets, given in Table 3.6 in the previous section.

3.5 Estimations using Measurement Data

The data obtained through simulations is as close to real-life scenarios as possible, but does not take into account the constantly changing environment due to various movements of people and objects in an indoor space, which is the true nature of real indoor scenarios.

The dataset created by taking measurements in the company’s office (Figure 3.24) in Malmö was also provided to us. Similar to the testbenches obtained from simulations, this measurement data also contains information about the AoAs, RSSI, estimates done using the LS method, and the ground truth values of the location of the tag. The measurement was done by placing the tag at 14 positions. At each position, 90 snapshots were captured holding information from all four anchor points, including tag location estimates in the form of Cartesian coordinates.

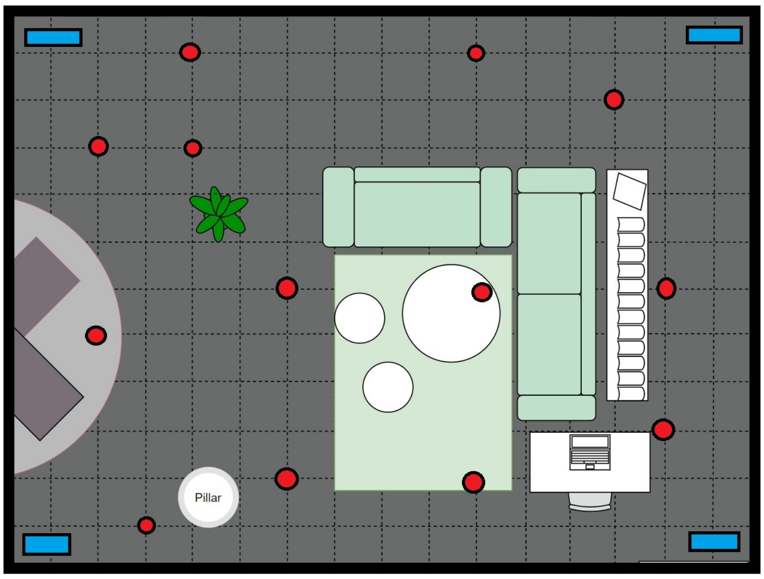


Figure 3.24: Floor Plan of a Real-Life Office Environment.

(a) The red circles refer to the 14 positions of the tag and the blue rectangles to the four anchor points.

Since there were 14 distinct locations of the tag, we decided that we can treat this as more of a classification problem than a regression problem, which was also used. The idea was to arrange the data in such a way that information from half of the positions can be used for training and the other half for testing. The original dimensions of the dataset were 14x90x4 for each of the features (i.e., AoAs in azimuth and elevation, RSSI, and X-, Y- and Z-coordinates), which correspond to 90 snapshots from all 4 anchors, taken at 14 positions. We rearranged the data to make the dimensions 1260x15. The 15 features were made of 4 columns each (corresponding to 4 anchor points) consisting of ϕ, θ and RSSI. Since 90 snapshots were taken at each of the 14 positions of the tag, meaning that there were only 14 sets of tag coordinates, the corresponding set of coordinates was repeated 90 times. This was done for each position.
Similar to regression used to treat the testbenches, we keep X- and Y-coordinates separate and estimate them independently in case of measurement data. Note again that Z-coordinate estimation has not been done because it is constant, i.e, the tag is at a constant height throughout the time the measurements were being taken.
All three algorithms were used to perform the classification. To do so, we first used the LabelEncoder() available in scikit-learn in order to assign a unique value to each datapoint in the target variables (x or y), since classifiers do not accept continuous values of the target variable while training. Note that the positon estimates by classification presented in section 3.6.2 have values corresponding to the

actual position coordinates given by the LabelEncoder(). We referred to the grid searches for the best parameters from Table 3.2.
Table 3.11 shows the average positioning error from each of the algorithms as a classifier and a regressor trained using the optimal parameters, along with the average positioning error given by the currently-used LS method. Note that the error is actually the mean Euclidean distance.

Algorithm	**Classifier**	**Regressor**
RF	0.01m	0.17m
SVM	0.68m	0.31m
kNN	0.02m	0.01m
LS (not ML)	1.78m	1.78m

Table 3.11: Average Positioning Error for Measurement Data.

From Table 3.11, we can draw two conclusions directly: every ML algorithm performs better than LS; and that for this specific dataset with very distinct ground truth coordinates used for training, classifiers perform better than regressors.
The CDF plots in Figures 3.26 and 3.27 show the positioning errors obtained from all three algorithms used as classifiers and regressors, compared with those obtained using LS method.
From both CDFs, we can broadly comment that error density is in centimetre-range for all ML algorithms, and a higher error density lies above 1m for LS. These plots also clearly indicate that ML algorithms outperform LS.
Another interesting way of visualizing the estimates from classification algorithms with the true values is the confusion matrix. It tells us how many estimates were done correctly and how many incorrectly, given the knowledge of the true values. The confusion matrices computed from all three classifiers are shown in Figures 3.28-3.33. The elements of the matrices have the total number of estimates obtained. Hence, the diagonal elements are the ones with the total number of correct estimates, and the rest are incorrect estimates.

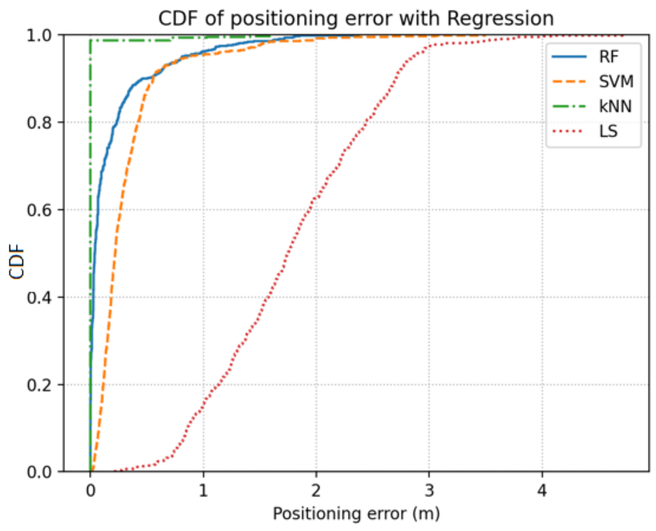


Figure 3.26: CDF of Positioning Error from Regression.

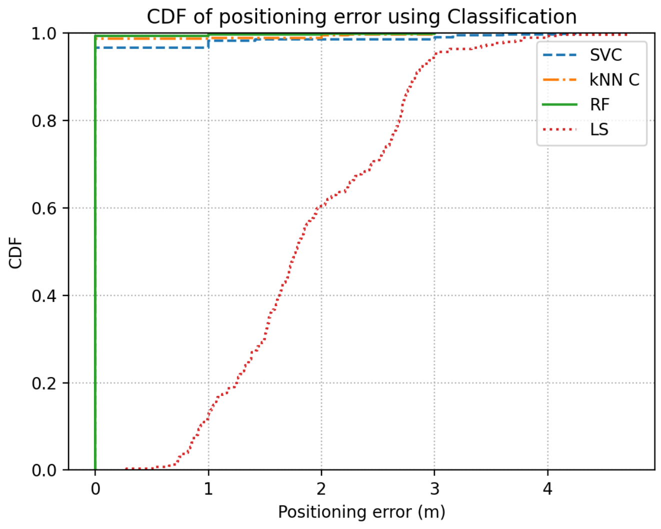


Figure 3.27: CDF of Positioning Error from Classification.

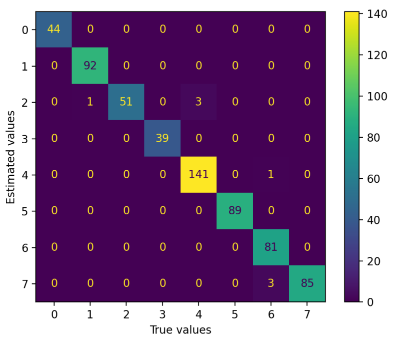


Figure 3.28: X-coordinate: RF

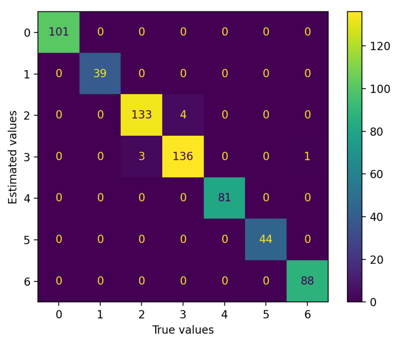


Figure 3.29: Y-coordinate: RF

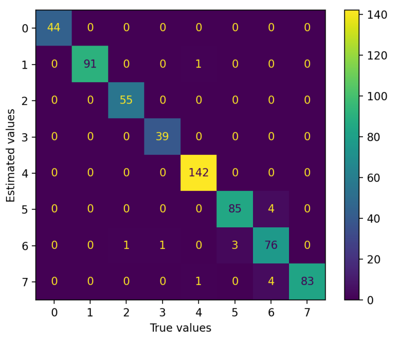


Figure 3.30: X-coordinate: SVM

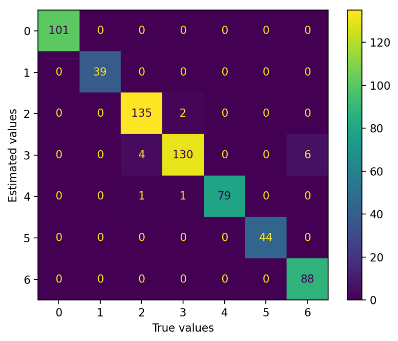


Figure 3.31: Y-coordinate: SVM

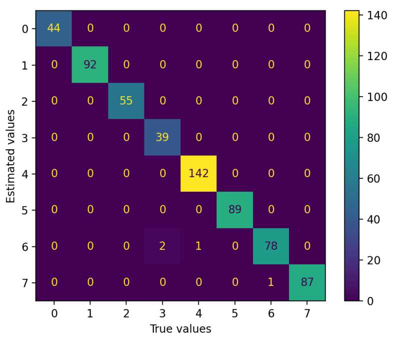


Figure 3.32: X-coordinate: kNN

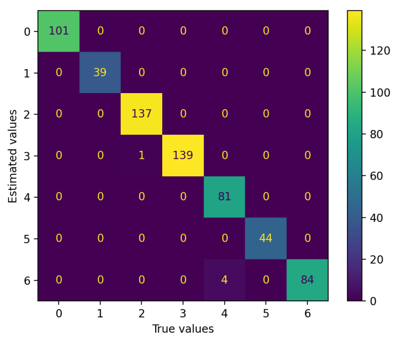


Figure 3.33: Y-coordinate: kNN

3.6 Position Estimation Results

As a final presentation of our work, in this section we show the estimated positions of the tag obtained from each algorithm in highly dense simulation scenarios and from physical measurements taken in an office environment. These estimates, along with the true coordinates of the tag, give a clear view of the performance of ML algorithms. Figure 3.34 includes a concise representation of the ML system designed during this thesis. In a real-world scenarios, it is difficult to estimate

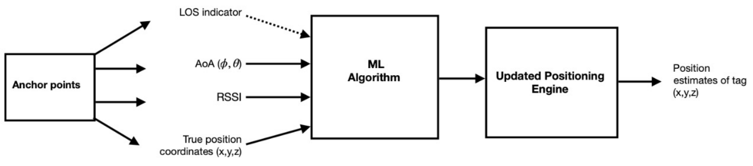

LOS indicator
AoA (ϕ, θ)
RSSI
True position coordinates (x,y,z)
Anchor points
ML Algorithm
Updated Positioning Engine
Position estimates of tag (x,y,z)

Figure 3.34: Proposed ML System.

whether a received signal has travelled via an LOS path or NLOS path. If it is possible to extract this information in real-time, it can add to position estimation. Hence, we have mentioned the LOS indicator as a potential feature that might be of use in the future if hardware systems permit.
We also worked with a simulation programme of the current positioning engine that uses the LS method. As an added comparison and representation of how the LS technique works, Figure 3.35 shows how the positioning engine performs the task. The figure has plots of the true path followed by the tag and the estimated coordinates obtained from the engine. This is demonstrated using a simulator of the positioning engine.

3.6.1 Estimates in Simulated Environments

The estimation of the position of the tag is done using regression. In Figures 3.36-3.41, we can see these position estimates plotted along with the corresponding true coordinates, i.e., where the tag was actually located. We use the most dense indoor scenarios, testbenches high concrete and high furniture. As mentioned previously, we think that these are the closest at describing how the position estimation would take place in similar real-world situations.

The size of the dataset procured from the simulated environments was higher than that of the data obtained from measurements. Hence, we have a high number of coordinate estimates. In both scenarios, we can observe RF to perform better with more precise estimations. But, from the results that we have presented in the earlier chapters, we also can conclude that all three algorithms perform better when compared to LS.

3.6.2 Estimates in Office Environment

In Figures 3.42-3.47, the true position of the tag is plotted along with its estimated position computed by each ML algorithm trained as both classifiers and regressors. This gives us a good picture about how classification and regression

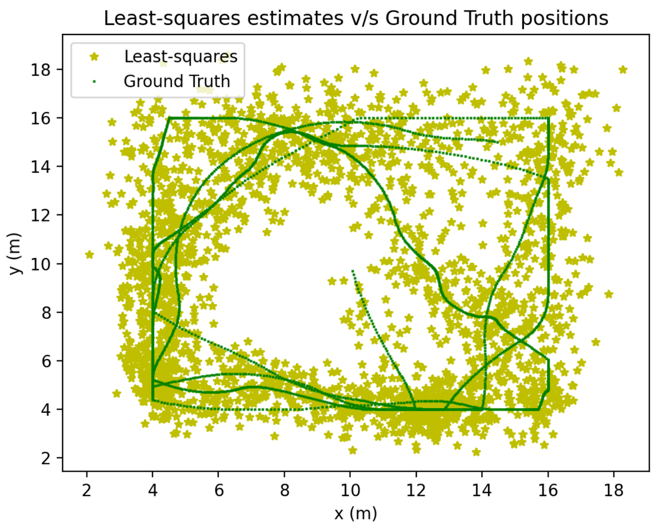


Figure 3.35: True and Estimated Path of the tag using LS Technique.

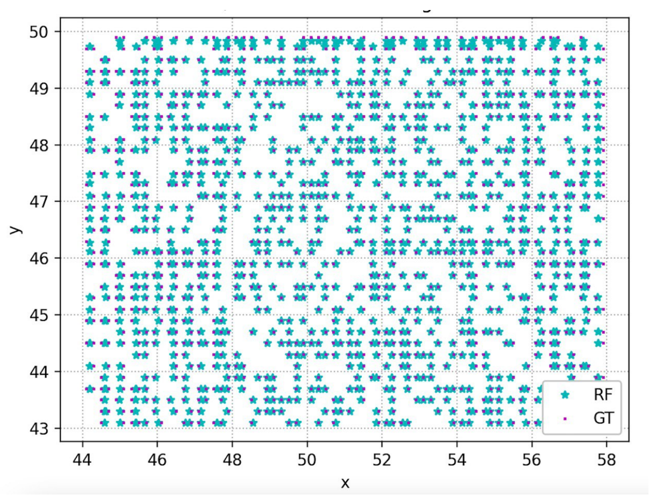


Figure 3.36: Position Estimations using RF in High Concrete Scenario.

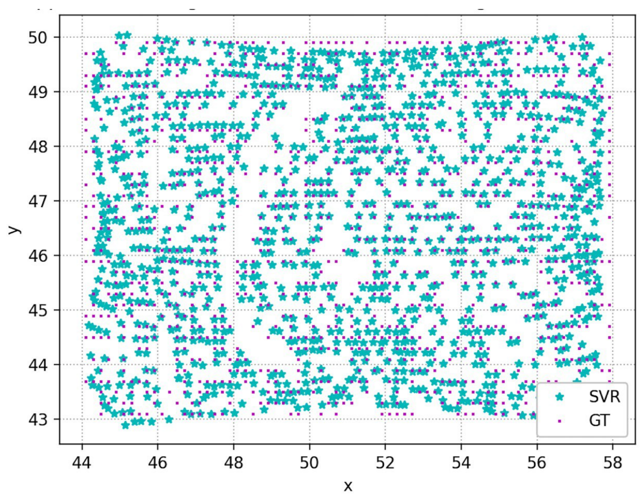


Figure 3.37: Position Estimations using SVM in High Concrete Scenario.

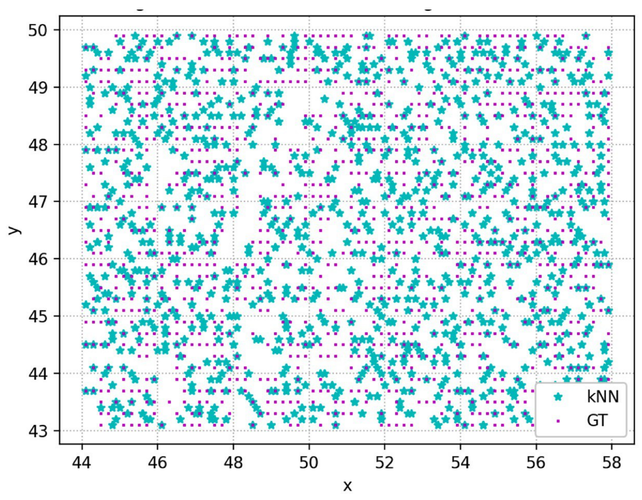


Figure 3.38: Position Estimations using kNN in High Concrete Scenario.

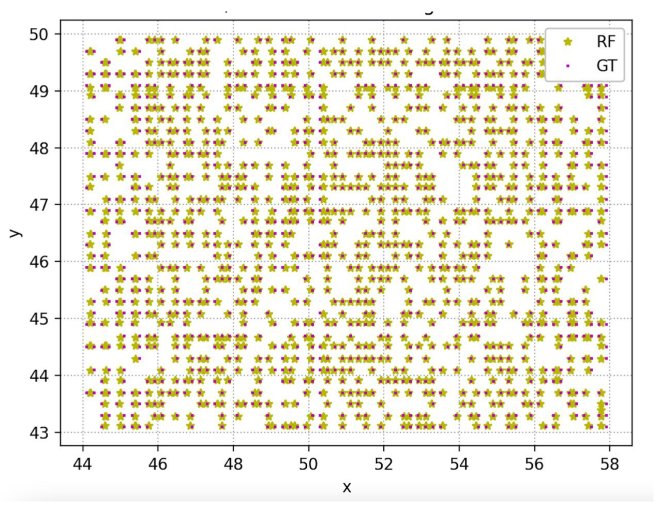


Figure 3.39: Position Estimations using RF in High Furniture Scenario.

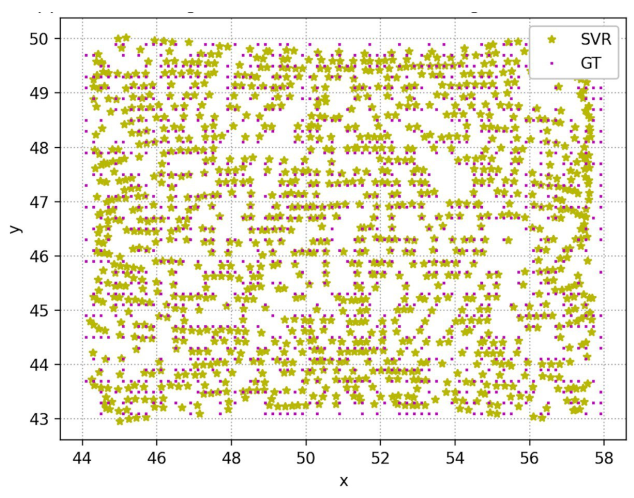


Figure 3.40: Position Estimations using SVM in High Furniture Scenario.

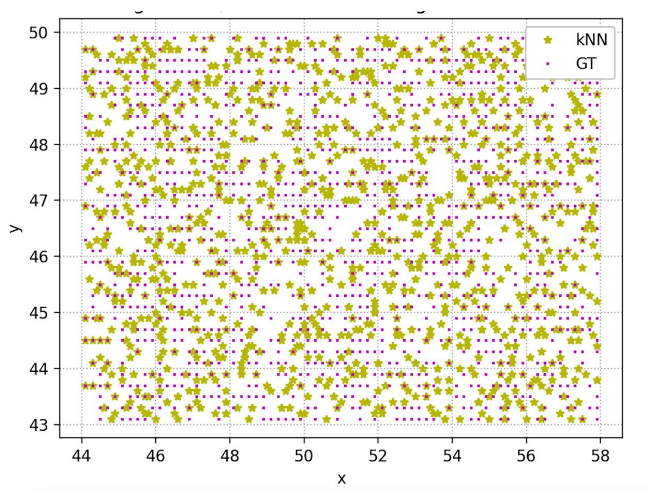


Figure 3.41: Position Estimations using kNN in High Furniture Scenario.

both perform.
The two figures also hint towards the fundamental difference between classification and regression. Classification algorithms simply label new observations based on the labels that they already know, whereas regression algorithms perform more of a prediction. Hence, we can see a cluster of estimates around the true coordinates. The important thing to note here is that even in the figure showing estimates obtained from regression, which is less suitable than classification in this scenario, there is less clustering of estimates around a ground truth value and visually more precise estimates than the LS method.

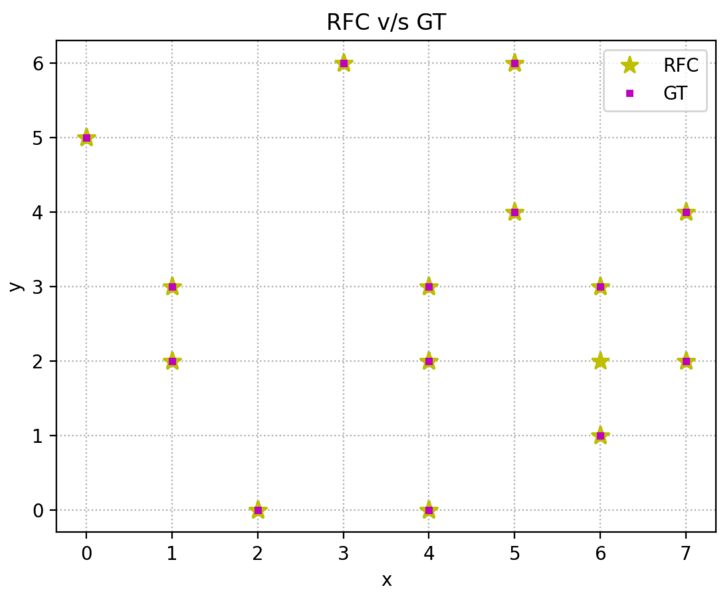


Figure 3.42: Position Estimation using RF Classifier in Office Environment.

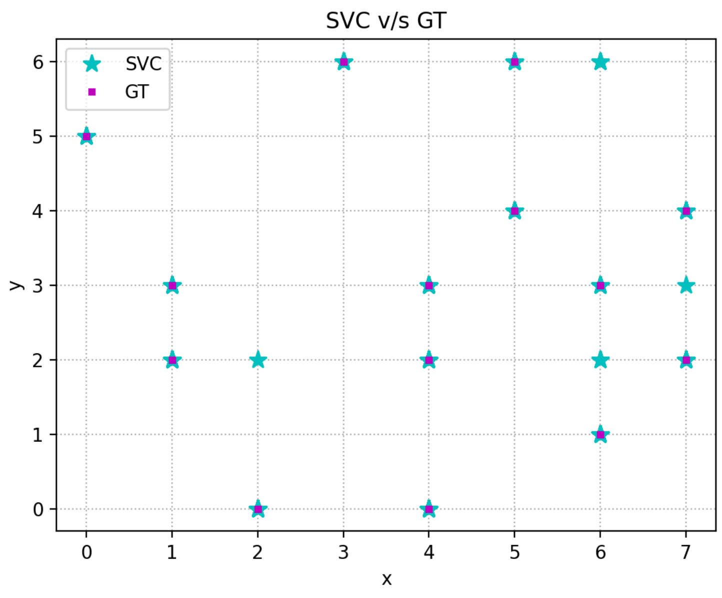


Figure 3.43: Position Estimation using SV Classifier in Office Environment.

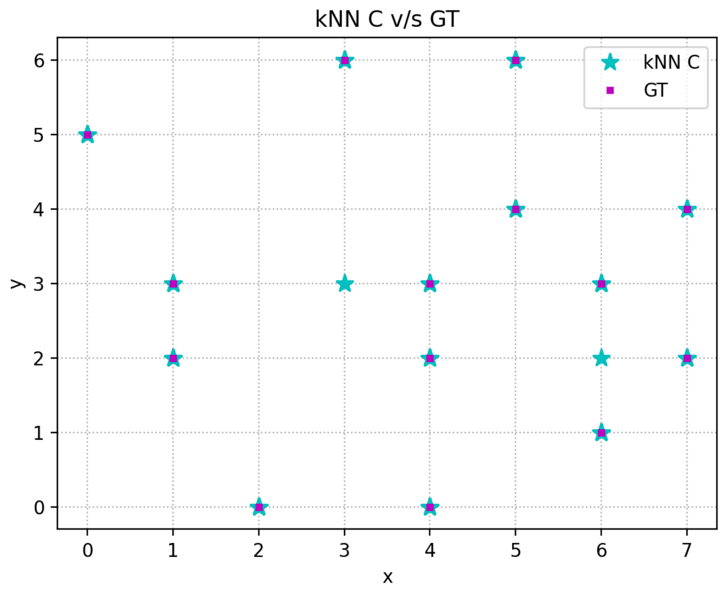


Figure 3.44: Position Estimation using kNN Classifier in Office Environment.

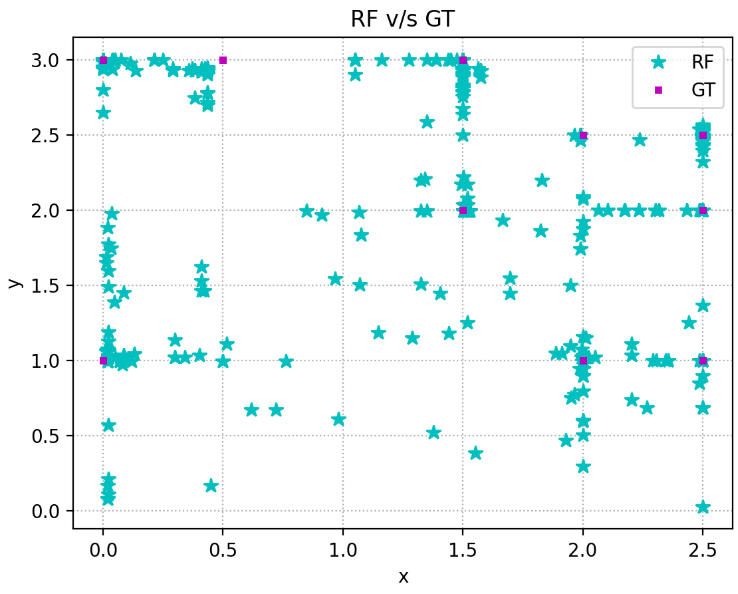


Figure 3.45: Position Estimation using RF Regressor in Office Environment.

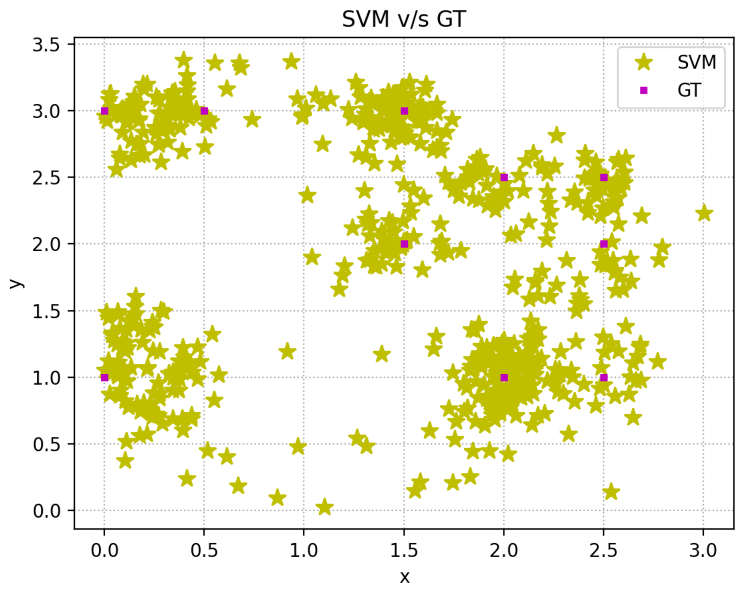


Figure 3.46: Position Estimation using SV Regressor in Office Environment.

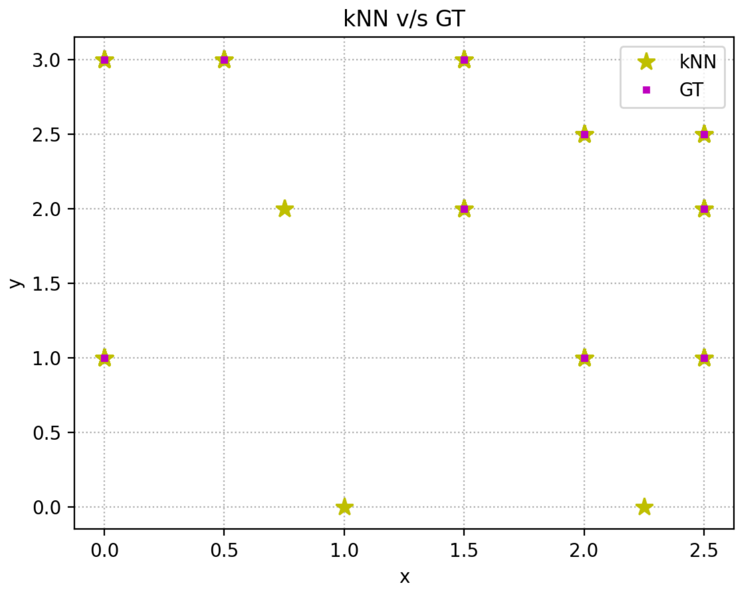


Figure 3.47: Position Estimation using kNN Regressor in Office Environment.

3.7 Discussions and Future Work

As we have learnt while working on our thesis, there is a lot to explore in this area of indoor positioning and ML. In addition to the ML models we have presented in the thesis so far, we also have some ideas on how some more learning or processing can be done in addition to ML, such as tracking using a Kalman Filter. We got inspiration to discuss ML and KF and suggest exploration in this direction from [5].
In this thesis, data from only one out of three available channels and two polarizations was used. Merging all available channels to increase the effectiveness of the ML estimation is definitely something that can be implemented.
The Kalman filter is a recursive algorithm used widely in tracking applications, like navigation and control in aircrafts and naval scenarios, robotics, signal processing, smoothening of data, etc., where there is significant uncertainty arising from statistical noise due to continuously changing states. It works on prediction and updation, and uses a series of measurements made over time (which makes it a time series analysis method) and outputs estimates of unknown variables.
The estimates are made by taking into account the previous and current states of the variable, which requires very little memory. In tracking applications, the "state" generally includes the position, velocity and acceleration information of the variable. These state variables are assumed to be random Gaussian variables. The interesting aspect about this algorithm is its straightforwardness; that it needs the state transition matrix and covariance matrix [28]. The state transition matrix

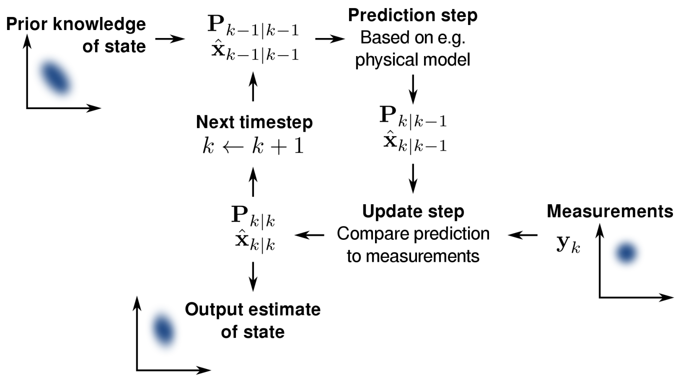

Prior knowledge of state
$\mathbf{P}_{k-1|k-1}$ $\hat{\mathbf{x}}_{k-1|k-1}$
Prediction step Based on e.g. physical model
Next timestep $k \leftarrow k+1$
$\mathbf{P}_{k|k-1}$ $\hat{\mathbf{x}}_{k|k-1}$
$\mathbf{P}_{k|k}$ $\hat{\mathbf{x}}_{k|k}$
Update step Compare prediction to measurements
Measurements $\mathbf{y}_k$
Output estimate of state

Figure 3.48: Mechanism of the Kalman Filter [29].

describes how the state changes from one to the next and the covariance matrix captures the correlation between the state variables which are being used for estimation. For example, the covariance matrix of a tracking system for which the filter is being used describes how the velocity and position are related. There is a wide range of modifications to the Kalman filter to make it suitable for even more varied scenarios, including non-linear filters such as the Extended, Unscented, and Cumulative Kalman filters.
The tracking of the tag can be done with its movement information, velocity and acceleration, so that the future "state", i.e., position, can be estimated using the

previous position of the tag. We think that using a Kalman Filter to track the tag's movement and further passing the filter's outputs to a ML algorithm to further improve these estimates is an interesting approach.
Another approach to using the Kalman Filter is to use it as a layer after the ML estimates have been made, in order to smoothen the outputs using the velocity and acceleration information at each time instance.
We suggest one more approach where we use estimates done by both methods and within a specific time window, select the estimate which gives the smaller error. In this scenario, the selection between the two methods, LS and ML, could give an even more improved accuracy. The difference between the models we have trained in this thesis and this approach is that estimation is done by giving ground truth values to an ML model that does not use the LS method at all. This approach uses both methods for estimation, which potentially takes longer, since it is an added step in the estimation process and might not be suitable in situations where fast computing is required.
The above approaches can be explored if the trade-off between computation time and accuracy is weighed, and time is the resource we are willing to sacrifice.

Chapter 4

Conclusions

We presented possibilities of the potential that Machine Learning brings to the field of indoor positioning by comparing the algorithms Random Forest, Support Vector Machine and k-Nearest Neighbors. Using optimal parameters and tailoring them to realistic scenarios, we have tried to show the superior outcomes of ML over the pre-existing Least-Squares method.
In this work, we investigated the features of each algorithm. Firstly, we examined the computational complexities of each algorithm by varying the size of the training and testing datasets, and studied the impact of varying parameters of the ML models on the outcomes. The important observations made are that RF gives the lowest positioning error whereas kNN gives the fastest predictions. But, as discussed previously, there are two parameters being used for the RF models and only one for kNN and SVM. Hence, relatively slower training and testing is expected in the case of RF. Considering both these observations, we can say that RF is a viable option. However, if the goal is to execute faster training, we suggest exploring different training sizes to come to a conclusion.
Secondly, we studied the behaviour of the three algorithms modelled as regressors in high-density simulated environments, imitating day-to-day indoor spaces. Although the LS method has shown to be accurate, all three ML algorithms in this project have proven remarkably noteworthy.
Thirdly, the estimations done in a physical environment add to the fact that ML, done with both classification and regression, performs better than LS. However, with higher number of measurements, the results could be more promising. In this specific measurement scenario, classification seemed to be the more suitable option but regression performs similarly, especially RF and kNN. The cross-prediction results and confusion matrices support the efficacy of ML.
Despite the approach of using the Angle of Arrival along with the RSSI is not very popular for position estimation, combined with Machine Learning, it shows promising results and has broad scope for research.

References

[1] Peng Dai, Yuan Yang, Manyi Wang, Ruqiang Yan, "Combination of DNN and Improved KNN for Indoor Location Fingerprinting", Wireless Communications and Mobile Computing, vol. 2019, Article ID 4283857, 9 pages, 2019. https://doi.org/10.1155/2019/4283857 https://www.hindawi.com/journals/wcmc/2019/4283857/

[2] Internet of Things, https://en.wikipedia.org/wiki/Internet_of_things

[3] Martin Woolley, *Bluetooth Direction Finding, A Technical Overview* https://www.bluetooth.com/bluetooth-resources/bluetooth-direction-finding/

[4] J. Zhang, M. Shafi, A. F. Molisch, F. Tufvesson, S. Wu and K. Kitao, "Channel Models and Measurements for 5G," in IEEE Communications Magazine, vol. 56, no. 12, pp. 12-13, December 2018, doi: 10.1109/MCOM.2018.8570033.

[5] Jia-You Hsieh, Chun-Hung Fan, Jian-Zhi Liao, Jyh-Yih Hsu and Huan Chen, "Study on the application of indoor positioning based on low power Bluetooth device combined with Kalman filter and machine learning" https://easychair.org/publications/preprint/VhvV

[6] Arogyaswami Paulraj, Rohit Nabar and Dhananjay Gore *Introduction to Space-Time Wireless Communications* 2003

[7] A. A. M. Saleh and R. Valenzuela, "A Statistical Model for Indoor Multipath Propagation," in IEEE Journal on Selected Areas in Communications, vol. 5, no. 2, pp. 128-137, February 1987, doi: 10.1109/JSAC.1987.1146527. https://ieeexplore.ieee.org/document/1146527

[8] Bluetooth Core Specification Version 5.1 Feature Overview, https://www.bluetooth.com/bluetooth-resources/bluetooth-core-specification-v5-1-feature-overview/

[9] Angle of Arrival https://en.wikipedia.org/wiki/Angle_of_arrival#Applications

[10] Zhang, H.; Zhang, Z. AOA-Based Three-Dimensional Positioning and Tracking Using the Factor Graph Technique. Symmetry 2020, 12, 1400.

https://doi.org/10.3390/sym12091400 https://www.mdpi.com/2073-8994/12/9/1400#cite

[11] Burk, M. (1962). Ramifications of the Relationship between Income and Food. Journal of Farm Economics, 44(1), 115-125. doi:10.2307/1235490 https://www.jstor.org/stable/1235490?seq=1

[12] Supervised vs. Unsupervised Learning https://towardsdatascience.com/supervised-vs-unsupervised-learning-14f68e32ea8d

[13] Nonparametric Statistics, https://en.wikipedia.org/wiki/Nonparametric_statistics

[14] k-Nearest Neighbors algorithm, https://en.wikipedia.org/wiki/Nonparametric_statistics

[15] Building a k-Nearest-Neighbors (k-NN) Model with Scikit-learn urlhttps://towardsdatascience.com/building-a-k-nearest-neighbors-k-nn-model-with-scikit-learn-51209555453a

[16] Decision Tree vs. Random Forest – Which Algorithm Should you Use? https://www.analyticsvidhya.com/blog/2020/05/decision-tree-vs-random-forest-algorithm/

[17] S. Bozkurt, G. Elibol, S. Gunal and U. Yayan, "A comparative study on machine learning algorithms for indoor positioning," 2015 International Symposium on Innovations in Intelligent SysTems and Applications (INISTA), 2015, pp. 1-8, doi: 10.1109/INISTA.2015.7276725. https://ieeexplore.ieee.org/document/7276725

[18] Support Vector Machine, https://en.wikipedia.org/wiki/Support-vector_machine

[19] Support Vector Regression Tutorial for Machine Learning, https://www.analyticsvidhya.com/blog/2020/03/support-vector-regression-tutorial-for-machine-learning/

[20] Support Vector Machine (SVM) https://www.mathworks.com/discovery/support-vector-machine.html

[21] Ensemble Learning https://en.wikipedia.org/wiki/Ensemble_learning

[22] Computational Complexity of Machine Learning Algorithms, https://www.thekerneltrip.com/machine/learning/computational-complexity-learning-algorithms/

[23] Big O Notation Explained with Examples https://www.freecodecamp.org/news/big-o-notation-explained-with-examples/

[24] Fast Computation of Moore-Penrose Inverse Matrices, https://arxiv.org/abs/0804.4809v1

[25] Support Vector Machine-Simply Explained, https://towardsdatascience.com/support-vector-machine-simply-explained-fee28eba5496

[26] Chris Albon *Machine Learning with Python Cookbook: Practical Solutions from Preprocessing to Deep Learning* p.235-238, 2018

[27] C3, *What is Mean Absolute Error?*, `https://c3.ai/`.

[28] FilterPy Documentation, *KalmanFilter* `https://filterpy.readthedocs.io/en/latest/kalman/KalmanFilter.html`

[29] Kalman Filter `https://en.wikipedia.org/wiki/Kalman_filter`