Deep Learning Aided NLOS Detection and Localization using RIS

Erik K. Roursgaard, Jonas Schjællerup, Oliver Damborg, Paw Johnsen, Thien Nguyen, Yutao Xie

Electronics systems - ES7-720 - 2024

Electronic Systems
Aalborg University
http://www.aau.dk

AALBORG UNIVERSITY
STUDENT REPORT

Title:
Deep Learning Aided NLOS Detection and Localization using RIS

Theme:
Machine learning for LOS/NLOS detection

Project Period:
Autumn Semester 2024

Project Group:
ES7-720

Participants:
Erik K. Roursgaard
Jonas Schjællerup
Oliver Damborg
Paw Johnsen
Thien Thach Nguyen
Yutao Xie

Supervisor:
Ming Shen
Gilberto Berardinelli

Copies: 1

Page Numbers:
Main paper: 4 to 89
Appendix: 95 to 144

Date of Completion:
September 22, 2025

Abstract:

With advances in development in 5G and research in 6G, higher frequencies are being utilized to comply with the higher demand in data speed. At higher frequencies, the signals become more vulnerable to reflections, scattering and interference when meeting obstacles. A solution to this could be the use of Reconfigurable Intelligent Surfaces (RIS) that have the potential to improve the propagation of these signals. However, to obtain full advantage of this, the detection of Line of Sight (LOS) and Non-Line of Sight (NLOS) is needed. Therefore, this project seeks to train several machine learning models to recognize, if a received signal is LOS or NLOS, and hereby compare the different models to each other. The machine learning models used are DNN, SVM and LSTM, with accuracies of 94%, 97% and below 80%, respectively. Furthermore, the signals are processed and augmented in different ways to investigate how this will affect the different models accuracies.
As a further research, there is presented two methods to estimate the devices direct distance to each other for both LOS and NLOS cases. One using trigonometry with the ToF of the signal and the other being the minimum of the signal's group delay, with a mean error of 21 cm and 4 cm, respectively.

Contents

Preface

This project is developed during the seventh's semester topic of "Machine Intelligence" of Aalborg University's M.Sc. in Electronic Systems in the period from 3/9-24 to 20/12-24. It is completed with the purpose to fulfill the curriculum, through a comprehensive analysis, design and implementation of a digital and analog system.

The group collectively wishes to extend appreciation to Sigurd Sándor Petersen for his valuable guidance throughout the tests and data knowledge for this semester project.

Reading guide
Citations in this report are denoted by [X]. A complete list of all used sources can be found in the Citations chapter. References are made throughout the text to figures/tables etc. Figures without a citation in the caption are self-constructed. References are also made to the appendices A to G, which contain supplementary material and testing journals for performed tests with results. In this paper, three datasets are used and denoted as "*OG*" for Original dataset, "*HF*" for High frequency lab dataset, and "*BC*" for Biggest chamber dataset. For mathematical expressions and equations involving complex numbers, the complex variable j is utilized to represent the imaginary unit. Additionally, the period symbol (".") is used as a decimal separator throughout the report.

Aalborg University, 20 December, 2024.

Erik Roursgaard
<erours21@student.aau.dk>

Jonas Schjællerup
<jschja21@student.aau.dk>

Oliver Damborg
<odambo20@student.aau.dk>

Paw Johnsen
<pmjo21@student.aau.dk>

Thien Nguyen
<tnguy21@student.aau.dk>

Yutao Xie
<yxie24@student.aau.dk>

Abbreviations

AD	Automatic Differentiation
ADAM	Adaptive Moment Estimation
AF	Antenna Factor
AI	Artificial Intelligence
ALU	Arithmetic Logic Unit
AWGN	additive white Gaussian noise
BC	Biggest Chamber
BP	Backpropagation
BPTT	Backpropagation Through Time
CI	Close-in
CIR	Channel Impulse Reponse
CM	Confusion Matrix
CNN	Convolutional Neural Network
CPU	Central Processing Unit
CSV	Comma Separated Values
DeepLift	Deep Learning Important FeaTures
DL	Deep Learning
DNN	Deep Neural Network
FFT	Fast Fourier Transform
FI	Floating-interception
FN	False Negative
FNN	Feed-Forward Neural Network
FP	False Positive
FSPL	Free Space Path Loss
FT	Fourier Transform
GD	Gradient Descent
GPU	Graphics Processing Unit
HF	High Frequency
KF	Kalman Filter
LLM	Large Language Model
LMMSE	Linear Minimum Mean Square Error
LOS	Line of Sight
LSTM	Long Short-term memory
MA	Moving Average
ML	Machine Learning
MLP	Multi Layer Perceptron

mmWave	Millimeter Wave
MSE	Mean Squared Error
NLOS	Non-Line of Sight
NN	Neural Network
OG	Original
PCA	Principle Component Analysis
PDP	Power Delay Profile
PSD	Power Spectral Density
RBF	Radial Basis Function
ReLU	Rectified Linear Unit
RIS	Reconfigurable Intelligent Surface
RNN	Recursive Neural Network
Rx	Receiver
SVM	Support Vector Machine
TL	Transfer Learning
TOA	Time of Arrival
TOF	Time of Flight
TP	True Positive
Tx	Transmitter

1. Introduction

In the age of digitalization, there is a growing need for data collection as well as sharing and distribution of these. This have resulted in a large scientific research effort into faster and more reliable data transfer between users and/or machines, e.g. Internet of Things (IoT). With the development of 5G and researching 6G, data transfer has achieved higher data rates by utilizing Millimeter Wave (mmWave) frequencies around 30-300 GHz. As consequences of the utilization of higher frequencies, the signals have become more vulnerable towards reflection and scattering when meeting obstacles in the environment [1, Chapter 10]. One solution to these problems is active repeaters, which can amplify and retransmit signals. However, they are energy- and cost-expensive, which have been leading into researching in newer technologies, which are cheaper and more energy efficient alternatives, such as reconfigurable intelligent surfaces (RIS) [2].

RIS is a planar array antenna (surface), which can be intelligently reconfigured in a way, such that it can receive a signal from a device and reflect it by manipulating the phase of the incoming signal, hereby beamforming and aiming the signal directly to the end user/device. The reconfiguration of the phase of the signal is achieved by passive components, which means that no additional energy are fed to the surface, hence making it energy efficient [3].

Utilizing the RIS and the channel impulse response (CIR), the environment can be mapped, making it possible to assess if the channel is line of sight (LOS) or non line of sight (NLOS), and the localization of the receiver [4]. In [5], different machine learning (ML) methods were utilized to assess if the channel between two devices communicating wirelessly was in LOS or NLOS without the use of RIS. The use of ML showed promising results in determining LOS from NLOS, with an increment in accuracy from 84.4% to 98%.

An initial problem statement presented below serves as guidance for determining which topics to investigate and scrutinize for further analysis.

Initial Problem Statement

> *How can machine learning be applied to classify Line Of Sight (LOS) and Non Line Of Sight (NLOS) conditions using Reconfigurable Intelligent Surfaces (RIS) in wireless communication networks to improve communication performance?*

2. Analysis

Throughout this analysis, some of the aspects that contribute to the reliability of the transmission, such as the path loss propagation of a signal when occurring line of sight and non line of sight, and radiation patterns of antennas will be analyzed. Furthermore, a brief description of antenna arrays and reconfigurable intelligent surfaces (RIS) will be presented. Afterwards, different machine learning methods, models and their functionalities will be analyzed.

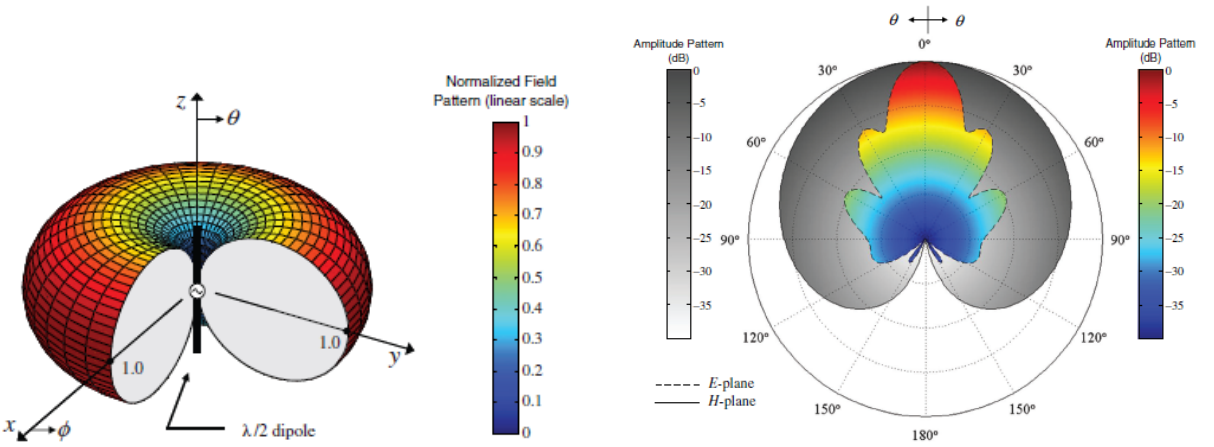
2.1 Wireless propagation

The following section will analyze and describe the radiation pattern of different antenna types such as directional and omnidirectional antennas and how arrays can enhance the antenna functionalities. Furthermore, three path loss propagation models will be described and analyzed with the intend to describe the challenges when the signal needs to travel in line of sight or non line of sight. Lastly, there will be a brief description of noise and its effect on the signal.

2.1.1 Antenna propagation

Antennas come in various sizes and shapes, depending on the frequencies, the phase, the amplification and so forth. The most common antenna types are the directional antenna and the omnidirectional antenna. The directional antenna is designed to have a narrow field of view, hereby amplifying the signal in certain directions, while the omnidirectional antenna has a circular radiation pattern which is orthogonal to the antenna, hereby making the antenna able to transmit or receive signals from all directions in the half plane [6]. The radiation pattern of an antenna is 3-dimensional and can be difficult to describe with numerical values, therefore it is common to represent the pattern using a field pattern or power pattern (linear or in dB). This can be shown in a 3D graph, shown in figure 2.1a, but also for the E- and H-plane, shown in figure 2.1b [7, Chapter 2].

In figure 2.1b, the mainlobe is at 0°, hereby defining the direction of the antenna. Following the field pattern along the angles, the signal field pattern decreases in strength, with some side lobes at around 30° and 60°, and as good as 0 amplification towards 180°, whereas the field pattern in figure 2.1a has a torus shape with equal signal amplification in the half plan pattern but zero amplification in the z direction.



(a) 3D field pattern of an omnidirectional antenna [7, Figure 4.2].

(b) E- and H-plane field pattern of a horn antenna [7, Figure 13.4].

Figure 2.1: Two illustrations of a 3D field pattern of an omnidirectional antenna and a 2D field plane pattern of a directional horn antenna.

The usage of the different antenna types in figures 2.1 depend on the use-case, i.e. omnidirectional or directional, whereas the former can be useful in situations, where multiple receivers are placed in all directions from the transmitter antenna (in this case the omnidirectional), while the latter can prove useful in use-cases, where the location of the receiver is known, The directivity of an antenna is also stated as the Q-factor or the quality factor of the antenna. It is a measure of the bandwidth of the antennas frequencies relative to the center frequency, which is calculated as

$$Q = \frac{f_c}{f_2 - f_1} \tag{2.1}$$

where

- f_c Center frequency
- f_1 Lower 3 dB cutoff frequency
- f_2 Upper 3 dB cutoff frequency

To increase the gain of an antenna, various methods can be used. One method can be to arrange the antennas in an array. An antenna array is a collection of different antenna elements, usually with equal dimensions, which are placed in either a one dimensional line or a 2 dimensional surface. Usually the elements in an array have equal spacing and equal physical dimensions, hereby simplifying the control parameters. An array of identical elements and magnitude with progressive phase is referred to as an uniform array. The total electromagnetic field of an array is determined by addition of all the field vectors from each n element called the Antenna Factor (AF), which is put in equation in 2.2 [7, Chapter 2].

$$\text{AF} = 1 + e^{+j(kd\ cos\ \theta+\beta)} + e^{+j2(kd\ cos\ \theta+\beta)} + ... + e^{j(N-1)(kd\ cos\ \theta+\beta)} \tag{2.2}$$

which can be simplified to

$$\mathrm{AF} = \sum_{n=1}^{N} e^{j(n-1)(kd\ cos\ \theta+\beta)} \tag{2.3}$$

where

AF	Antenna Factor
N	Total number of elements
k	Wave number
d	Distance between the elements from the reference element
θ	Frequency of the signal
β	Phase shift of the element.

From Equation 2.3 it is shown that by changing the number of elements in an array, or by changing the phase and/or distance between these, it is possible to manipulate the radiation from the array, hereby beam forming a signal towards a desired target [7, Chapter 2].

2.1.2 Propagation

As a signal is propagating through space, it will be stretched out, as the signal propagates further from its source, hereby making the energy smaller at the receiving antenna relative to the transmitting antenna. In the far field, the Friis' Free Space Path Loss (FSPL) [7, S. 2.17.1] can be used to calculate the power relative to the distance, which is

$$\mathrm{PL}_{FSPL}(f, d) = 20 \log_{10} \left(\frac{4\pi df}{c} \right) \tag{2.4}$$

where

d	Distance between Tx and Rx
f	Frequency
c	The speed of light

Using Equation 2.4, the path loss for three frequencies (100 GHz, 500 GHz and 1 THz) is shown in figure 2.2 with the distance shown in meter on the x-axis, and the signals attenuation in dB on the y-axis.

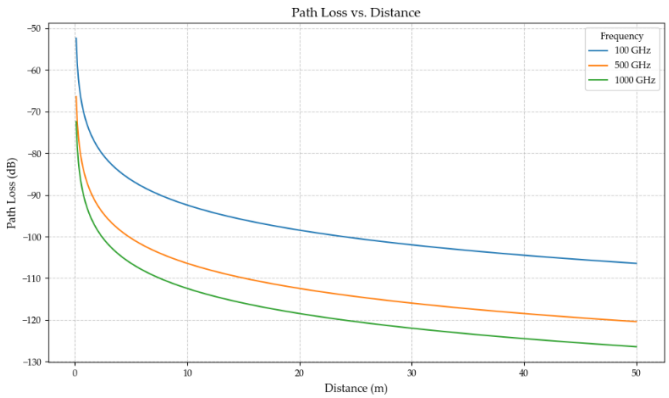


Figure 2.2: The free space path loss of a three signals at frequencies 100 GHz, 500 GHz and 1000 GHz, respectively. Distance in meter is shown on the x-axis, while the loss in dB is shown on the y-axis.

Equation (2.4) only works in systems, which are in each other's line of sight (LOS), which means that no obstacle of any kind is in between the Transmitter (Tx) and the Receiver (Rx). When the signal propagates through obstacles, and some of the obstacles are in the direct line, it is called non line of sight (NLOS). When in NLOS the signal will be scattered, diffracted or simply be highly attenuated going through an object, hereby making equation (2.4) inefficient for calculating the received power at the receiving antenna.

Scattering is when a signal hits an obstacle and is then reflected back in a new angle relative to the incoming, hereby altering the trajectory for the signal, as shown in the lower part of Figure 2.3. This can be described by a two-ray model [1, S. 10.4] depending on the (estimated) number of reflected rays. Diffraction is when the signal is "bent around" and object on its path to the receiver, as shown in the upper part figure 2.3. Diffraction occurs when for example an obstructions are blocking the LOS path between the Tx and Rx.

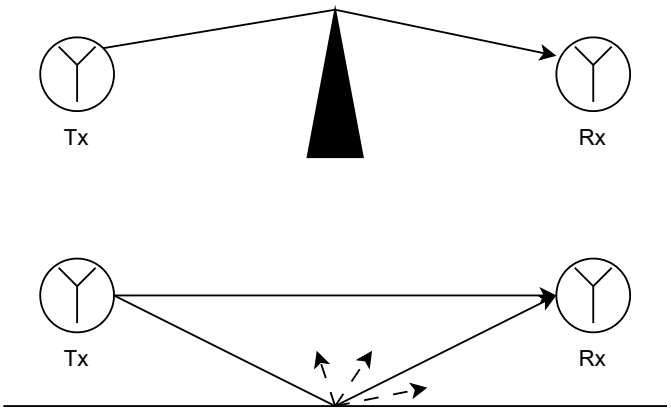


Figure 2.3: Upper: Example of a signal being diffracted around a corner. Below: Two signals, one direct and the other being reflected, hereby arriving later at the receiver. The dotted signals on the lower figure are the scattering results of the reflection.

Therefore, when two antennas are NLOS, it is more realistic to use a Close-in (CI) for modeling the behavior of a signal's propagation[8]. This model can to some extend resemble a random environment such as Urban Microcell or indoor hotspot. The CI model expands on the FSPL model by adding a Gaussian random variable with a variance σ which accounts for the shadow fading of the signal to the FSPL model:

$$\mathrm{PL}_{CI}(f,d) = \mathrm{PL}_{FSPL}(f,d_0) + 10n\log_{10}\left(\frac{d}{d_0}\right) + \chi_\sigma^{CI} \tag{2.5}$$

Where

PL_{CI}	Path loss for CI model
f	Transmission frequency
d	Distance between Tx and Rx
d_0	Reference distance, typically 1
χ_σ^{CI}	Gaussian random variable $\mathcal{N}(0,\sigma)$

In a scenario, where f and d_0 will be fixed, n is a variable that describes the path loss in dB in terms of decades of reference distance beginning at 1. d is the distance between the Tx and Rx. It can be derived by using the minimum mean-square error (MMSE). It is used to describe the environment.

The Floating-interception (FI) model does not consider a physical distance between Tx and Rx, as the CI model does [8]. Instead, the model is built by testing the environment, and then fitting the model to simulate the environment. The FI model is defined as

$$\mathrm{PL}_{FI} = \alpha + 10\beta \log_{10}(d) + \chi_{\sigma}^{FI} \tag{2.6}$$

where

PL_{FI}	Path Loss for floating-interception model
α	Floating-intercept
β	Slope of the model
χ_{σ}^{FI}	Gaussian random variable $\mathcal{N}(0, \sigma)$.

α serves as a fitting parameter to fit the model to the measured or expected path loss behavior of the environment, while β controls the slope of the path loss. Therefore, the FI model is better at modeling a signal in a specific environment, while the CI model is less complex and therefore also more general. The variables are used to simulate shadow-fading, which is caused by objects in the signal's path [8].

Noise

Wireless transmissions are exposed to noise when propagating through the channel. The noise is added onto the signal, hereby making it commonly described as Additive White Gaussian Noise (AWGN) and denoted as $w(t)$. The noise is Gaussian distributed, $N(\mu, \sigma^2)$, hereby making it a stochastic process. The Power Spectral Density (PSD) is described as [9, S. 10.2].

$$S_w(f) = \frac{N_0}{2} \tag{2.7}$$

Where N_0 is thermal noise described as

$$N_0 = kT \tag{2.8}$$

where

N_0	Thermal noise applied for 1 Hz
k	Boltzmann's constant
T	Temperature at the receiver in Kelvin

To determine the power of the noise P_N on the measured signal, the following equation is used:

$$P_N = N_0 \cdot B \tag{2.9}$$

Where B is the bandwidth of the measured window.

In Figure 2.4 the propagation of a signal from a transmitter to a receiver are shown, where the noise $W(t)$ is being added to the transmitted signal $X(t)$, hereby making the received signal as combination $X(t) + W(t)$.

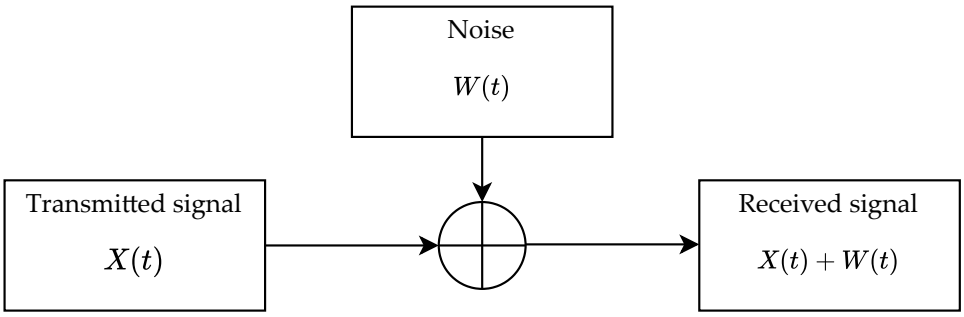


Figure 2.4: The propagation of a signal from the transmitter to the receiver with the noise added.

As the distance between the antennas increases, the received power decreases following equation 2.4, hereby making it difficult to distinguish between the signal and the thermal noise. This relation is referred to as the signal-to-noise ratio (SNR), which is described as

$$\gamma = \frac{P_r}{P_N} \tag{2.10}$$

where

γ	Signal-to-noise ratio
P_r	Received power
P_N	Power of the noise

2.1.3 RIS

A Reconfigurable Intelligent Surface (RIS) is a 2-dimensional planar antenna array of N passive elements, which can be manipulated to reflect and direct incoming electromagnetic waves from one or more sources to others through modified control signals, illustrated in Figure 2.5. At high frequencies, the wavelength of the signal are very small, therefore making the size of the elements practically small. For the RIS to reflect the most possible, the antenna should be made a bit smaller than the wavelength of the signal, hereby making it easier to group in larger surfaces. Furthermore, the short waves become more vulnerable towards obstacles, scattering and diffraction, thus making RIS useful for 5G and 6G technology [3, Chapter 33].

The surface contains elements of 10 and up to several 100s. By configuring the amplitude and phase of each element, it is possible to direct and beamform incoming signals from a source in all directions of the halfplane. Hence, it is possible to control the scattering, absorption, reflection and diffraction properties of the entire RIS in realtime [10].

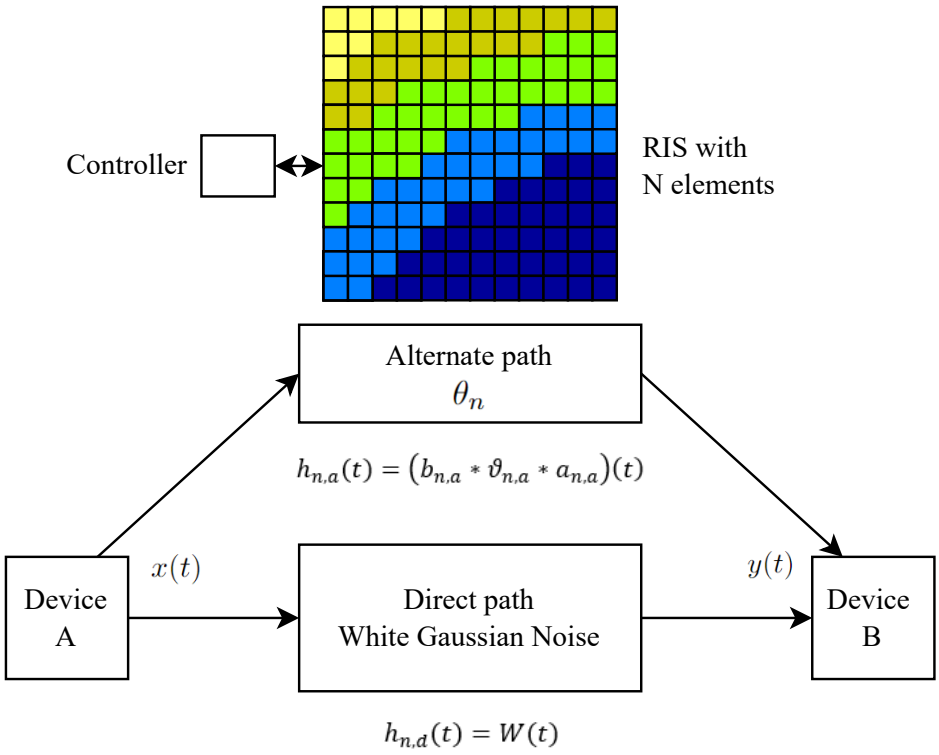


Figure 2.5: Communication system of simplified propagation paths between a transmitter and a receiver. The directivity of RIS is shown with the difference in colors of the elements, with dark blue being a high gain and yellow being low gain.

In figure 2.5, the principle behind RIS is shown, whereas device A transmits a packet to device B by two different paths; directly and reflected by RIS. When the message $x(t)$ is transmitted directly to device B it is subject to various different disturbances such as white gaussian noise, diffraction and interference from other sources. That is simplified as the impulse response $h_n(t) = W(t)$ in the figure. This results in a received signal at device B as

$$y(t) = (h_n * x_n)(t) \tag{2.11}$$

When the message is received by the alternate route, the RIS is capable of manipulating the signal which results in another impulse response

$$h_{n,a}(t) = (b_{n,a} * \vartheta_{n,a} * a_{n,a})(t) \tag{2.12}$$

where

$b_{n,a}$	The arbitrary impulse response from the RIS to the device B
$\vartheta_{n,a}$	The reconfigurable impulse responses generated by RIS, affecting the phase
$a_{n,a}$	The arbitrary impulse response from device A to the elements of RIS

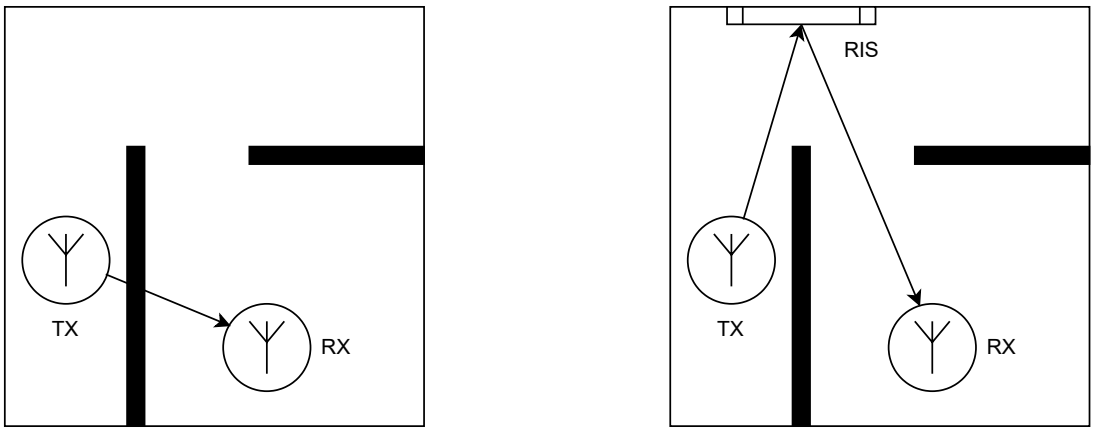
The received signal at device B will then have the form

$$y(t) = \sum_{n=1}^{N} (h_{n,d} * x)(t) + (h_{n,a} * x)(t) \tag{2.13}$$

which will improve the reception at device B, as the RIS is capable of filtering and alternating the phase and amplitude of the signal to overcome the different obstacles and noise factors between device A and B [10].

As the signals phase is shifted for each element, the configurable impulse response obtained by the RIS $\vartheta_{n,a}(t)$ can be achieved by controlling the delay of the phase for each element, thus amplifying the signal passively. This could be done by using a frequency selective surface, where each element is embedded with a positive-intrinsic-negative (PIN) diode. By varying the biasing voltage, the PIN diode can be switched between two states, ON or OFF, thus realizing a phase-shift of π in radians. Another method is varactor-tuned resonators, which is to implement tuneable capacitors in each element. This makes it possible to phase-shift each element according to the biasing voltage across the capacitor in a more "smooth" way, compared to the ON/OFF state. When utilizing passive components, RIS requires no additional energy to reflect incoming signals. This makes it highly energy efficient and therefore a coveted method of amplifying and improving signal coverage [3].

An advantage, which can be gained from the usage of RIS in an indoor environment, is shown in Figure 2.6. As the signal will be attenuated through the wall in figure 2.6a and going around the wall. The RIS allows for a higher signal reception by focusing the signal to the RX and thereby minimizing the signal loss as seen in figure 2.6b.



(a) Propagation of a signal between two devices through a wall.

(b) Propagation of a signal being reflected on a RIS, hereby propagation around the wall.

Figure 2.6: A scenario where the implementation of a RIS could be a great improvement of a signal's power or to obtain LOS when obstacles occur.

2.2 Machine learning and Neural Networks

Machine Learning (ML) is considered a subset of the term Artificial Intelligence (AI). ML algorithms have the ability to learn patterns based on previous data and give predictions or make decisions on new data. ML is commonly applied in use-cases such as linear regression, dimensionality reduction techniques like Principle Component Analysis (PCA), and clustering algorithms to group data. In particular the subfield of ML called *Deep Learning* utilizing Neural Networks (NNs) is of particular interest for its ability to model complex tasks and automatically find optimal features for a given task [11].

ML include parameters that adjust over time and are estimated as more data becomes available. This process of parameter estimation, known as *training* [12, CH.6 S.1.2], enables the model to *learn* or *adapt* to the given data. There are several different learning methods that determines how an ML model should be trained. The choice of method and model is highly dependent on the specific use-case and the characteristics of the training data provided. Assumptions of the data can often be made, such that some particular ML methods might perform well. However there is no guarantee for this as stated by the No Free Lunch Theorem[13, p. 33].

Supervised learning

In supervised learning the model is tasked to solve two different separate problems. Firstly is Classification problems, where the model has to classify some input data and assign it to a class $\mathcal{C}$. Secondly is the regression problems, where the model has to yield a new value. The ML model is given input data and produces output data called *predictions*[1]. The class $\mathcal{C}$ of the input data is known and is called the *label*. Labels are assigned to an entire dataset in the process of *data annotation*, which is often done manually. Although data annotation is a tedious and time-consuming process [14, S. Labeling], especially with larger datasets, better annotation often results in more effective learning and improved model performance.

In many cases, there are multiple outputs representing a probability distribution. Choosing the final class prediction is simply taking the highest probability:

$$\mathcal{C}_{pred} = \max_{k} p(\mathcal{C}_k|x) \tag{2.14}$$

Where:

$\mathcal{C}_{pred}$	Predicted class
$\mathcal{C}_k$	Class k
x	Input data

Unsupervised Learning

The focus of unsupervised learning is to determine patterns and relationships in the data without having any labels. Unsupervised learning methods include dimensionality reduction, such as PCA, and data clustering such as k-means.

Unsupervised learning is mostly used for pattern recognition.

[1] Regardless of it being a classification or regression problem.

An example of pattern recognition is the automatic recommendation of products for customers on e-commerce websites. This can be done using clustering. Over time as the user shops on the website, the ML model will be able to group the customers with similar customers and recommend products typically bought by the customers in the grouping.

2.2.1 Deep Learning

ML refers to the concept of an algorithm learning and adapting to data by analyzing examples of other data [15, S. 1.1]. Deep Learning (DL) differs from ML, by using a NN inspired by a biological brain. Among many different NN architectures the Deep Neural Network (DNN) is notable for its use of multiple *hidden layers*. A fundamental architecture of a DNN is the Multi Layer Perceptron (MLP), an example of a DNN can be seen in Figure 2.7. MLPs are also known as Feed-Forward Neural Network (FNN), where the input data exclusively propagates towards the output layer without backward connections in the NN. The *depth* of a network is an important aspect that describes the amount layers in the network. The *width* of a model describes the amount of neurons in a layer. Deeper networks tend to learn complex patterns more efficiently than wide, shallow networks due to their *parameter efficiency*. It requires exponentially more neurons for wide and shallow networks to model a complex function as opposed to a narrow and deep network [13, p. 323].

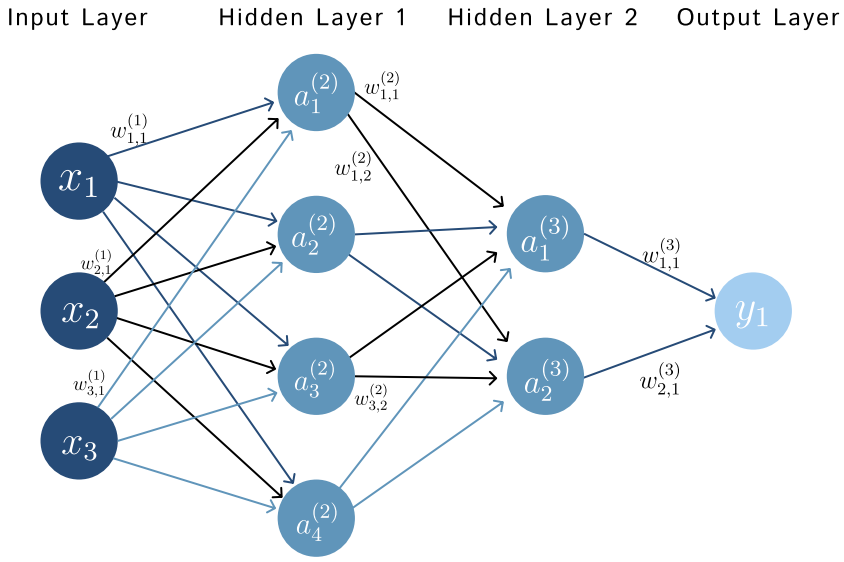


Figure 2.7: Illustration of a Multi Layer Perceptron that consists of 3 different layer types: Input, hidden and output. The circles are called *nodes* or *neurons*, while the lines between each neuron is called a *weight*. Adapted from [16].

Visually, the circles in a MLP are called *nodes* or *neurons* and the lines between each neuron are a unique *weight* that is randomly initialized. Each neuron outputs a linear combination of the associated weights and inputs called the *pre-activation*. The pre-activation[2] from one layer j to the other i, where $i = j + 1$ can be denoted as [17, Eq. 5.2]:

[2]In literature, this is often just referred to as activation, but will be referred to as pre-activation to avoid ambiguity as the output of an activation function is also called an activation or post-activation.

$$a_i^{(l)} = \sum_{i=1}^{N} \left(w_{j,i}^{(l-1)} \cdot x_{j,i}^{(l-1)} + b_{j,i}^{(l-1)} \right) \tag{2.15}$$

Where:

$a_i^{(l)}$	Pre-activations of node i from layer l
$x_{j,i}^{(l-1)}$	Input to the neuron i at layer $l-1$
$b_{j,i}^{(l-1)}$	The unique bias between two neurons at layer j and i at layer $l-1$
$w_{j,i}^{(l-1)}$	The unique weight between two neurons at layer j and i at layer $l-1$.

Equation 2.15 is fundamental to what makes a DNN. The equation is inherently a linear combination and can therefore only do linear transformations. This is insufficient for solving most real world problems, as they often are non linear. To solve this issue about non-linearity *activation functions* are used to enable non-linearity.

2.2.2 Activation functions

Neurons calculating the linear combination in Equation 2.15 are a generalization of the perceptron algorithm, as reflected in the name Multi Layer Perceptron. The pre-activation of a perceptron is used for a step function as seen in equation 2.16 [17, Eq. 4.53]:

$$f(a) = \begin{cases} +1, & a \geq 0 \\ -1, & a < 0 \end{cases} \tag{2.16}$$

As a consequence of the step function, perceptrons are only able to model linear processes. However, neurons can utilize any activation function which are ideally non-linear and differentiable [12, S. 6.1.5]. This allows the modeling of non-linear relations and the calculations of gradients to optimize model weights. The non-linearity added to the linear model makes the optimization problem a non-convex problem which will be discussed in Section 2.2.3. The output of an activation functions is called the *activation*[3], which follows the form:

$$z_k^{(j)} = h(a_k^{(j)}) \tag{2.17}$$

Where:

$a_i^{(j)}$	Pre-activation of node i in layer j.
$z_i^{(j)}$	Activation of of node of i in layer j.
$h(\cdot)$	Non-linear and differentiable activation function for layer activation of a layer j.

Choosing an appropriate activation function is important for addressing different problems, when training a NN including the *vanishing* and *exploding* gradient problem (See Section 2.2.3). Three popular activation functions are:

[3] In literature this is sometimes called post-activation, but will be referred to as activation.

1. ReLU
2. Sigmoid
3. Tanh

ReLU

Rectified Linear Unit (ReLU) is a common activation function that is computationally cheap to calculate, and returns the maximum value between 0 and the input a:

$$R(a) = \max(0, a) \tag{2.18}$$

ReLU is generally considered the best activation function [12, S. 6.1.4] and is additionally very computationally cheap as the gradient is either $0 for a \leq 0$ or $1 for a > 0$.

Equation (2.18) is plotted in figure 2.8.

Sigmoid

The sigmoid function is another popular activation function that maps all input data into the range of $[0; 1]$ and therefore also has applications as a probability map. Sigmoid is defined as [17, Eq. 5.6]:

$$\sigma(a) = \frac{1}{1 + e^{-a}} \tag{2.19}$$

Equation (2.19) is plotted in figure 2.8.

Tanh

The hyperbolic tangent function, also called tanh, is defined as in equation (2.20) [17, Eq. 5.59]. The tanh functions output ranges from -1 to 1, which makes it able to handle negative values in contrast to the sigmoid and ReLU function. However, a consequence of this is that the tanh function suffers from the vanishing gradient problem during backpropagation, as the gradients of the tanh function can become very small (close to 0) [17, Eq. 3.6].

$$\tanh(\alpha) = 2 \cdot \sigma(\alpha) - 1 \tag{2.20}$$

Where:

$\sigma(\alpha)$ Sigmoid activation function

A plot of the tanh function compared with the ReLU and Sigmoid function can be seen in Figure 2.8.

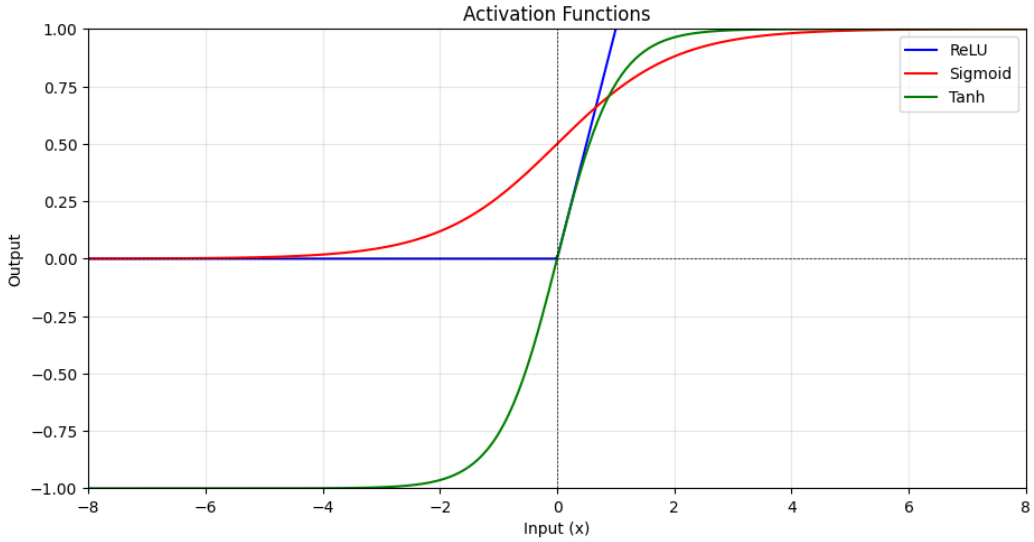


Figure 2.8: Comparison of the activation ReLU, Sigmoid and Tanh. Note the ReLU function becomes larger than one.

2.2.3 Backpropagation

To adapt a DNN to learn and adapt to new data using supervised learning, the standard algorithm used is Backpropagation (BP). BP uses both the loss and the gradients to update the weights in the model. Although BP relies on gradients of an error curve, it is actually algorithm agnostic. This means that it is a category of *optimizer* algorithms known as Gradient Descent (GD). It can be divided into three high-level steps of BP:

1. Calculate loss
2. Perform gradient descent
3. Perform back propagation to update weights

Loss refers to the error between the ground truth, i.e. labels and the predicted result from the DNN model. This loss can be represented as a function and is minimized in MLPs. A visual representation of a loss surface as a function of two weights can be seen on figure 2.9.

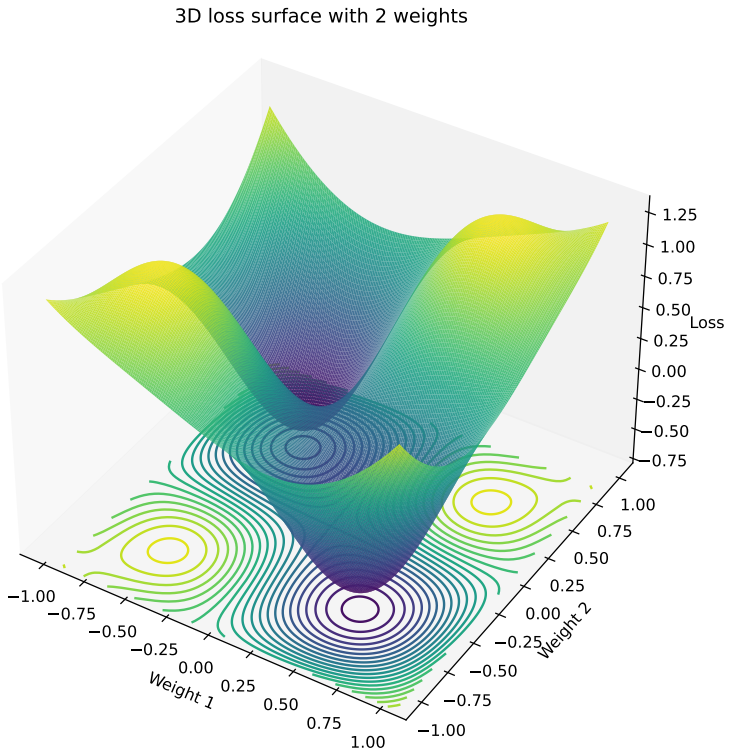


Figure 2.9: Simplified 3D Loss surface with 2 weights. Typically this would be from thousands to millions of dimensions.

On the loss surface on the figure 2.9 it can be observed that this is a non-convex problem, thus finding the global minimum is not a trivial task. The loss surfaces are functions of every weight and bias in the network, thus the dimensions will typically be in many orders of magnitude higher, but for the sake of visualization has been kept at 3 dimensions. The update rule for BP is given as [17, Eq. 5.41]:

$$w_{t+1} = w_t - \eta \cdot \frac{\partial E(w_t)}{\partial w_t} \tag{2.21}$$

$$= w_t - \eta \cdot \nabla E(w_t) \tag{2.22}$$

Where:

$w_{\tau+1}$	New updated weights
w_τ	Current weights
η	Learning rate
$\nabla E(w_\tau)$	Gradient of the loss function as a function of the current weights

In deep learning, the purpose is to minimize the loss value as much as possible. The learning rate is a constant factor, which is often set to a small positive value, such as 0.001, 0.0001, or smaller, that determines the rate at which the loss value decreases. The choice of the specific value is based on several considerations. 1) When the network is very complex, the learning rate should not be too small. 2) When the network is not complex, the learning rate should be less

than $1 \cdot e^{(-5)}$. If it decreases too quickly, it may lead to oscillations. 3) The initial learning rate can be chosen randomly. If the loss function decreases too quickly, the learning rate should be reduced, while if too slowly, the learning rate should be increased.

The chosen loss function determines the type of gradient result that is returned to the back propagation. Thus it is important to determine the correct loss function for the model.

2.2.4 Loss functions and regularization

Loss functions serve the purpose of telling the bias between a supervised model's prediction and the ground truth. Choosing a loss function usually is use-case specific, as seen in Table 2.1. Each loss function is described in further detail in Appendix A.1.

Name	Use-case
Mean Square Error	Regression
Binary Cross Entropy	Classification
Cross Entropy	
Focal loss	

Table 2.1: Loss functions grouped by use-case.

Mean square error

Mean Squared Error (MSE) is a common loss function used in regression tasks to measure the average squared difference between predicted and actual values. The equation for MSE can be found in Equation (A.1).

Binary Cross Entropy

Binary cross entropy is a loss function used in binary classification problems, where the output has two possible values (0 or 1). The equation can be found in Equation (A.2).

Cross Entropy

The cross entropy loss function is used to calculate the difference between the two probability distributions, the true labels and the predicted probabilities. A clear benefit of the cross-entropy function is that it punishes confident incorrect predictions more than small errors. Equation (A.3) shows the cross entropy loss function.

Focal loss

Focal Loss, an extension of the Cross-Entropy Loss, reduces the impact of well-classified examples, focusing more on harder-to-classify samples. Equation (A.4) explains the focal loss function and its extension of cross-entropy.

Dropout regularization

For loss functions there are many approaches to try to prevent weights from overfitting, which is called *regularization*. One approach is the dropout regularization, where during each training step, the neurons in the hidden layers are randomly 'dropped' with probability p. Being 'dropped' means that their activation function is set to zero. Dropped neurons will not participate in

computations during the current training step, but has the possibility for reactivation for future training steps. An example of dropout regularization can be seen in Figure 2.10.

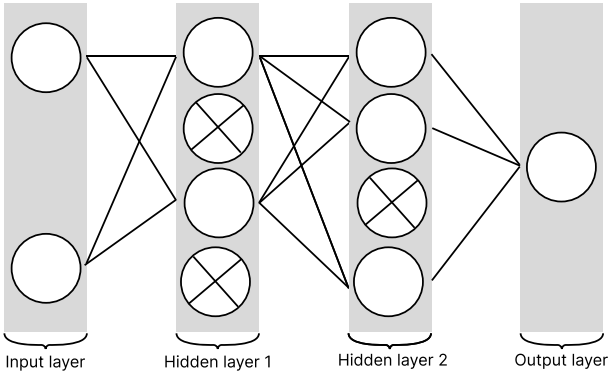


Figure 2.10: Illustration of an MLP network where all hidden layers are also dropout layers. Every neuron in a dropout layer has a probability p of dropping out.

2.2.5 Gradient descent

Gradient descent is a category of multidimensional gradient methods, whose goal is to minimize a cost function $f(x)$ so that the loss can be minimized. If the cost function $f(x)$ is differentiable, it is possible to compute the negative gradient of the function, and apply a small "step" in that direction to gradually approach the minimum of the function. By repeating these steps, the local minimum of the function can be found, and potentially even the global minimum. Depending on the step-size, the algorithm can either perform slowly with a somewhat high accuracy (vanishing gradient) or quickly and with a somewhat bad accuracy (exploding gradient), with a small and large step-size, respectively [18, Ch. 5].

Datasets for machine learning and deep learning models can contain up to millions of data samples. With a dataset that size, the standard gradient descent algorithm performs slowly as all data is used in every iteration. Therefore, it is most common to use the extended version Stochastic Gradient Descent (SGD). It works by splitting the data up into several subsets also called batches, and by an iterative process it works through all the training examples until all the data is used. This is done multiple times, which is referred to as an epoch. By adding some noise to the gradient at each step, it has the possibility of "jumping" from one valley to another of the loss function, hereby raising the possibility of finding the global minimum. The SGD algorithm is usually applied with a learning rate schedule, which starts at high values in the beginning, but as the number of epochs increases, the learning rate decreases, hereby fine-tuning the parameters [19, Ch. 6].

Adam

The Adaptive Moment Estimation (ADAM) optimizer is a widely used optimization algorithm that combines momentum and adaptive learning rate. Momentum can be thought of as an acceleration down the error curve given by the loss function and the current weights of the model. Momentum is given as [13, Eq. 11-8]:

$$m_{\tau+1} = \beta_1 \cdot m_\tau + (1 - \beta_1) \cdot \nabla E(w_\tau) \tag{2.23}$$

An implication of the momentum term is that it enables ADAM to continue in a particular direction and "skip" small local minimas, as seen in Figure 2.11. This might help ADAM

m_τ	Momentum at time τ
β_1	Exponential decay
$\nabla E(w_\tau)$	Gradient of the loss function

converge towards a global minimum, although there is no guarantee that this new local minima is necessarily better than the one previously skipped due to the momentum term.

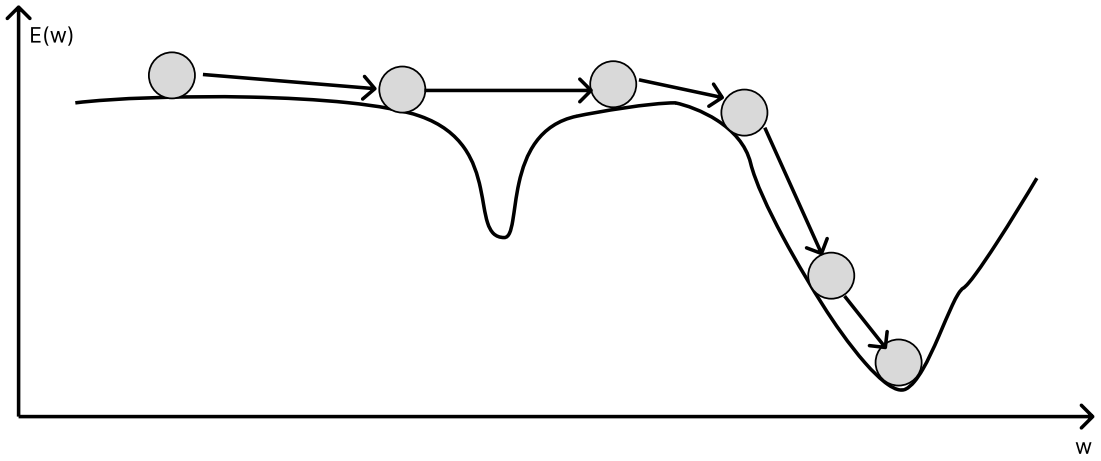


Figure 2.11: The momentum used in ADAM can help skip small valleys in the error curve, which might result in a better local minima.

Additionally, ADAM keeps track of another parameter called the velocity, which handles the adaptive learning rate and is given as [13, Eq. 11-8]:

$$v_{\tau+1} = \beta_2 \cdot v_\tau + (1 + \beta_2) \cdot \nabla E(w_\tau) \otimes \nabla E(w_\tau) \tag{2.24}$$

Where:

v_τ	Velocity at step τ
β_2	Exponential decay
$\nabla E(w_\tau) \otimes \nabla E(w_\tau)$	Element wise product between gradients of the loss function

At initialization of the ADAM algorithm, m_0 and v_0 are both 0. In early timesteps, especially at initialization of the algorithm m and v is biased towards 0 [20]. The bias corrected versions of m_τ and v_τ are given as:

$$\hat{m}_\tau = \frac{m_\tau}{1 - \beta_1^\tau} \tag{2.25}$$

$$\hat{v}_\tau = \frac{v_\tau}{1 - \beta_2^\tau} \tag{2.26}$$

Where:

Finally, the update rule for ADAM is given as:

$\hat{m}_\tau$ Bias corrected m_τ
$\hat{v}_\tau$ Bias corrected v_τ

$$w_{\tau+1} = w_\tau - \eta \cdot \left(\frac{\hat{m}_\tau}{\sqrt{\hat{v}_\tau} + \epsilon} \right) \tag{2.27}$$

Where:

ϵ Smoothing term for numerical stability

Pseudocode for the adam optimizer can be seen in algorithm 1. Note that the default parameters might differ between implementations,

Algorithm 1 Adam Optimizer

1: **Default parameters** [21] $\eta = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$
2: Initialize $m_0 = 0$ (Initial momentum)
3: Initialize $v_0 = 0$ (Initial velocity)
4: **while** not converged **do**
5: Update momentum $m_{\tau+1}$ using Equation 2.23
6: Update velocity $v_{\tau+1}$ using Equation 2.24
7: Bias correct $\hat{m}_\tau$ using Equation 2.25
8: Bias correct $\hat{v}_\tau$ using Equation 2.26
9: Update weights $w_{\tau+1}$ using Equation 2.27
10: **end while**

2.2.6 Types of Machine learning models

When developing a neural network it is necessary to look at the intended usage of the model. In DL there are many different architectures, each with its own positives and negatives. These architectures are based on the neurons connected in ways that determine the specific function of the network. Depending on the desired output and the input data, different NN are chosen. One such architecture is the DNN, described in sections 2.2.1, with this model being utilized for identifying patterns in the data. For high dimensionality classification problems, Support Vector Machines (SVMs) are used due to their high efficiency [22].

For text-based dataset, models such as the Recursive Neural Network (RNN) and Large Language Model (LLM) are used due to their structure, where the RNN is useful for sequences [23]. For image-based purposes, networks such as the Convolutional Neural Network (CNN) are used, however for the purpose of this project, CNNs are not ideal. For this project, the DNN, SVM and the Long Short-term memory (LSTM) appear to be the more suitable models. The LSTM and SVM will be described in the following section.

SVM

Support Vector Machine (SVM) is a classical machine learning algorithm usually used for classification, regression and anomaly detection. SVM is very effective in high dimensional spaces, as well as being memory efficient and versatile. A property of the SVM is that the determination of the model parameters are fundamentally an optimization problem. As the SVM is a decision

machine, it does not output probabilities, but a either $+1$ or -1 like the perception in equation (2.16) [17, S. 7.1]. An SVM can be seen as a binary problem, where it consists of a function separating two or more classes. This function can be found and adapted through training, so that it can differentiate between similar data. The SVM decision boundary is usually in the form of a hyperplane in cases of models using multiple dimensions of data, and the goal is to maximize the distance between each class as seen on Figure 2.12.

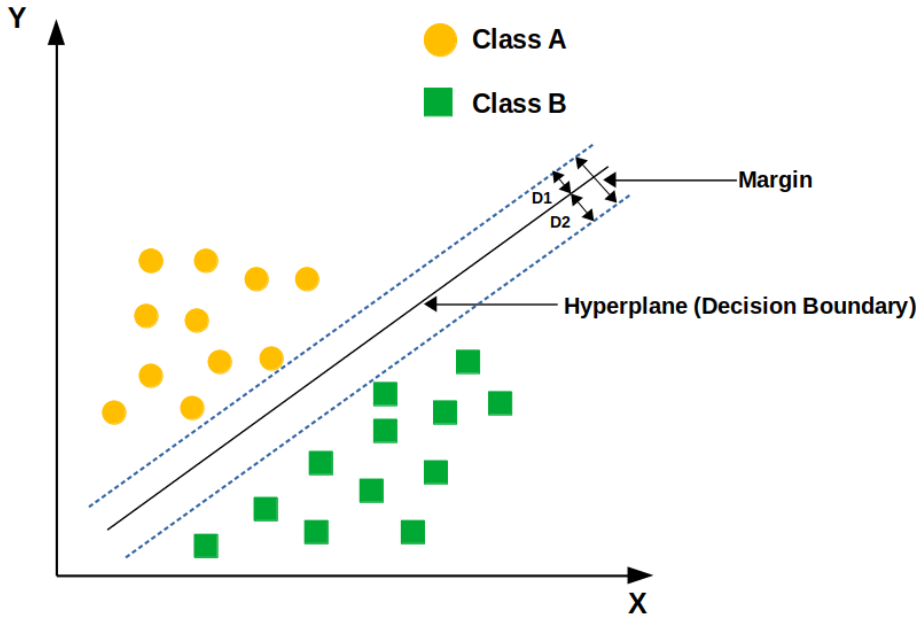


Figure 2.12: Illustration of the SVM classifying two different classes in 2 dimensions by maximizing the margins between each class for the hyperplane. Illustration from [24].

For multiclass solutions, there are mainly 2 methods used, one being *one-versus-the-rest* and the other being *one-versus-one*. The *one-versus-the-rest* method constructs K separate SVMs, where each SVM is trained on its related data as the positive data, and the rest as the negative data. This can lead to many problems, as multiple SVMs may claim that a data point belongs to the class it covers, making it difficult to differentiate between, and knowing which class it actually belongs to.

The other method, *one-versus-one*, trains an SVM for each possible combination of classes, resulting in $K(K-1)/2$ different SVMs. For classifying datapoints, it uses the class which has the highest amount of "votes", meaning which class claims it the most. The issue with this method is its high complexity due to the high amount of SVMs [17, chapter 7].

LSTM

The LSTM is a specific implementation of the RNN described in appendix A.3. The LSTM was designed to handle the issues of vanishing or exploding gradients which are issues plaguing the RNN. The LSTM is well-suited for handling tasks focused on sequential data. This is due to the introduction of a long term memory value C_t together with the short term memory value h_t. An example of an LSTM cell can be seen on figure 2.13.

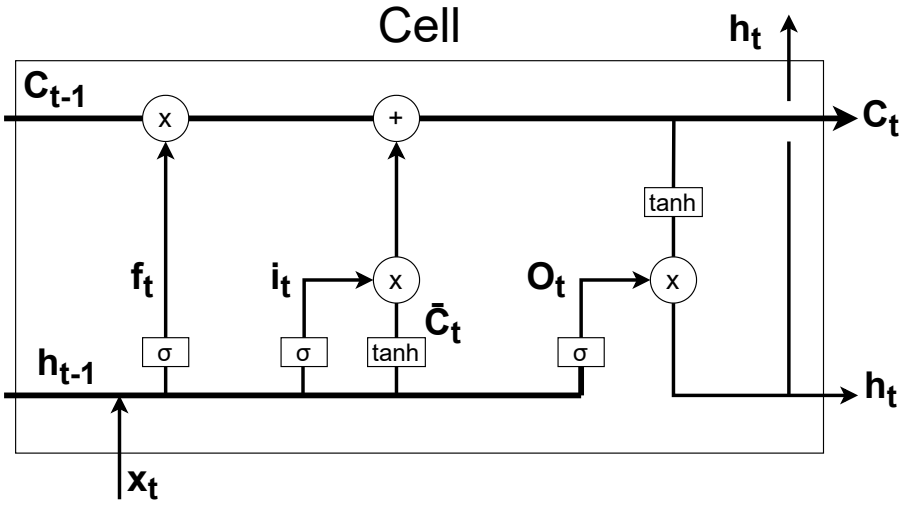


Figure 2.13: Single cell of an LSTM model.

Figure 2.13 shows the flow of the inputted data through three gates, and how they relate in the formulas described in appendix (A.7) - (A.12). The Figure 2.13 uses the forget gate f_t to determine how much of the new input (x_t) that should be used in the long term memory, which gives a value between 0 and 1. Next up, the input gate i_t calculates the long term memory value C_t. Finally, the output gate O_t determines the next short term memory value h_t, which is also the output/prediction of the current input [25].

2.2.7 Model parameters

The parameters in a model decide the design, complexity and behavior. There are two groups of parameters that do this; model parameters and hyperparameters. Model parameters are internal parameters such as weights and biases that are optimized through algorithms such as ADAM [13] described in section 2.2.5.

The hyperparameters depend on the model used, and are configured externally to be a certain value during training. These are fixed and are not changed during each run, but they are configurable outside of training to optimize the model. The hyperparameters can consist of the following possible parameters:

- Amount of layers
- Neurons in each layer
- Dropout rate
- Activation functions
- Batch normalization
- Batch size
- Softmax
- Kernel

- Degree of polynomial kernel.
- Gamma γ
- Regularization parameter (C)

Through a combination of these, the model is constructed with a specific number of layers, neurons and dropout rate with additional hyperparameters in between. As the data passes through the layers, the distribution of each layer's input changes throughout the training process. This will slow down the training process due to the model requiring a lower learning rate when trying to reach a optimal minimum [26]. To speed up the training time and improve the performance, Batch normalization is utilized to standardize the mean and variance but with trainable parameters. The definition of Batch normalization is described in Equation (2.28).

$$y^{(k)} = \gamma^{(k)} \cdot \frac{x^{(k)} - E[x^{(k)}]}{\sqrt{Var[x^{(k)}]}} + \beta^{(k)} \tag{2.28}$$

where

$y^{(k)}$	The k-th feature/channel normalized result from $x^{(k)}$
$x^{(k)}$	The k-th feature/channel input data
$\gamma^{(k)}$	The learnable scaling factor for the feature/channel
$\beta^{(k)}$	The learnable shifting factor for the feature/channel

The $\frac{x^{(k)}-E[x^{(k)}]}{\sqrt{Var[x^{(k)}]}}$ in the function is the standardization equation, where $\gamma^{(k)}$ and $\beta^{(k)}$ are the learnable parameters that are learned through each batch. A batch is normalized each time the function is run and the learnable factors are changed depending on the results.

Batch normalization is proven to drastically improve the performance of DNNs, since it lowers the change in values between each layer, and therefore helps multiple parts of the neural network [26].

The hyperparameter *batch size* controls how many data instances are passed through a ML model before BP is performed. Choosing a small batch size of 32 or less leads to better generalization at the cost of increasing computation time [13, p. 326].

The Softmax function is a multiclass function used to predict the probability of a sample belonging to a specific class. In a classification problem, the softmax function is a requirement. The softmax function takes in the data from the model and calculates the probability of the data belonging to a certain class. The highest probability found is then used to determine the class. The equation looks like [13, p. 148]:

$$\hat{P}_k = \frac{e^{(s_k(x))}}{\sum_{j=1}^{K} e^{(s_j(x))}} \tag{2.29}$$

where

$\hat{P}_k$	The probability of the instance x belonging to class k.
$s(x)$	A vector containing the scores of each class for instance x
K	The number of classes

A parameter that is specific to the SVM and its submodels, is the kernel parameter. The kernel is a placeholder term for a subgroup of functions that are available for the SVM. These functions are; Linear, Polynomial, Radial Basis Function (RBF) and Sigmoid. The kernel sigmoid, despite its name, is actually a tanh function [27]. Depending on which kernel is utilized, other parameters are required to be set. In the case of a polynomial kernel, the order of the polynomial is important. Another important parameter is the gamma (γ) as it is determines how much influence a training sample has[27]. The gamma (γ) and the regularization parameter (C) are both used for several of the kernels, where the regularization parameter controls the trade-off between achieving a low error on the training set, and minimizing model complexity. The RBF kernel is commonly used for SVM classification problems. It measures the similarity between two data points based on their distance in a higher-dimensional space. When the distance between the points increase, the output approaches 0. When the distances decrease, the output approaches 1. The equation for the RBF kernel is [13, p. 159]

$$K(x, x') = exp(-\gamma||x - x'||^2) \tag{2.30}$$

Here the $\gamma = \frac{1}{2 \cdot \sigma^2}$, where σ is a variable, sometimes set to the variance of the dataset.

2.2.8 Model evaluation

To determine the performance of a trained model, it is required to evaluate how well the model performs on an independent dataset. This is commonly known as cross-validation, where the model is tested on an independent testset[13].

From the evaluation, the hyperparameters can be optimized to obtain a better performance. Improving a model repeatedly can lead to higher accuracies, however it also increases the risk of overfitting to the specific dataset.

Overfitting and underfitting

There are two states that a model can easily risk entering when training. These are called overfitting and underfitting. Overfitting occurs when a model is trained too much on a specific dataset, causing it to fail at generalizing on new data. This can occur when a model is too complex or the dataset is too small or noisy compared to the size of the model. Overfitting can be resolved through a simpler model, more training data or reducing the level of noise in the dataset[13, p. 27-29].

Underfitting happens when a model is too simple to learn the underlying pattern in the data. One way of resolving this is to increase the parameters, increase the complexity of the model or reducing the regularization constraints such as batch normalization or dropout[13, p. 29]. Deciding when the model is too complex or simple for a task is called the *bias-variance trade-off* [13, p. 134]. Making the model too complex will lead to overfitting (increasing variance), but making the model too simple will lead to underfitting as the model will be unable to learn the underlying patterns (bias).

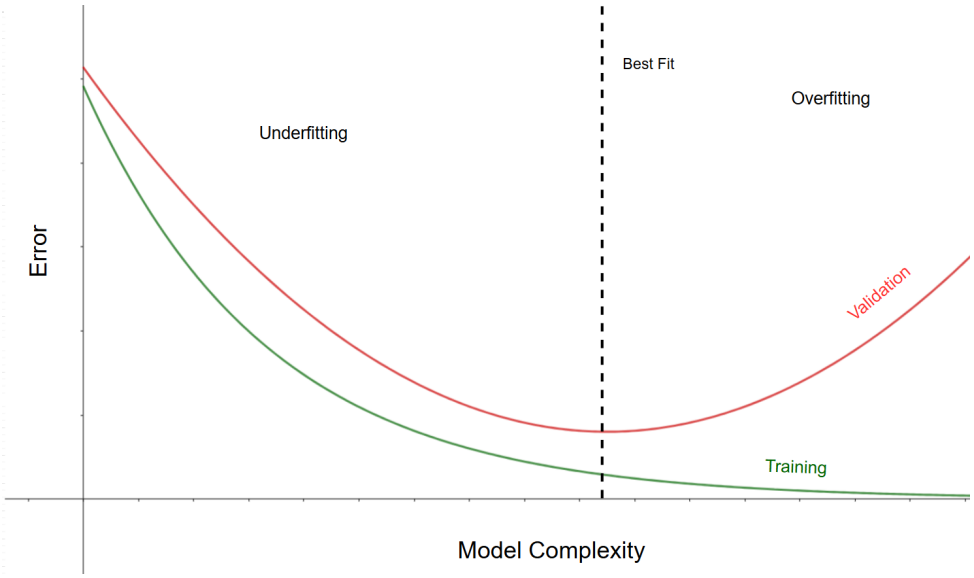


Figure 2.14: Example graph of the overfitting vs underfitting of a model when comparing its validation and training data. The dotted line shows the optimal model complexity to achieve the lowest error.

An example showcasing the relationship between error, model complexity and over/under-fitting can be seen on Figure 2.14. The dotted line shows the optimal complexity to achieve the lowest loss without overfitting.

Cross-validation

In order to determine if the model is overfit, cross-validation can be utilized which introduces an additional dataset. This set is called the validation set. The validation set is an independent dataset often extracted from the overall data before a model is trained to ensure similar conditions. At the same time a third dataset is extracted overall data called the test dataset. To verify the performance of the model, it is trained on the reduced dataset and every epoch is evaluated on the validation set.

The model that achieved the best accuracy on the validation set is then evaluated on the test dataset. This determines whether the performance achieved with the validation set is specific to that dataset or if it can generalize to another dataset.

This helps assess if the model has overfitted to the training and validation set and is a effective method for validating its performance.

The size of the distribution of data between the three datasets (training, validation, test) is important to an extent. Too little validation data and the validation accuracy will be imprecise. A too large validation set will cause imprecise accuracies if the remaining training set is too small. A small validation and testing datasets are likewise not large enough to ascertain a proper performance of the model with inconsistency in the accuracy being common [13, p. 31].

K-fold cross-validation

One option to determine and validate the performance of the model on the data is to perform a repeated cross-validation on many small datasets known as folds. This is known as K-fold cross validation and it helps determine potential issues if certain datasets have issues with accuracy.

K-fold cross validation works by taking the full dataset, splitting it into k folds and using one fold for validation or testing while the rest *k-1* is used for training the model. The training is operated for each fold, ensuring that every single fold has been used as a validation/testing set. The average accuracy of the model is then determined by finding the average of the k-fold validation accuracies. If it appears that the variance in accuracy is widely different between the folds, it shows that the model may require some modification of its hyperparameters[13].

The k-fold validation method is useful for determining the model with many different combinations of data. However, with the k-fold amount of models, the training model is extended by a factor k, making it harder to utilize for large models[13].

F1-score

In classification tasks, the F1-score metric is common as it combines *recall* and *precision*, which shows the relationship of how precise actual predictions are using True Positive (TP), False Negative (FN) and False Positive (FP). Precision is defined as:

$$\text{Precision} = \frac{TP}{TP + FP} \tag{2.31}$$

One shortcoming of the precision metric is the fact that it can not account for the model just being lucky and estimating correctly. Recall on the other hand accounts for this shortcoming and is defined as:

$$\text{Recall} = \frac{TP}{TP + FN} \tag{2.32}$$

The F1 is a combination of the two, which is defined as:

$$\text{F1-score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}, \tag{2.33}$$

The F1 score is the harmonic mean of precision and recall in the interval [0;1]. As it is a harmonic mean, the low values have a higher weight. As a result of this, a high F1-score is only possible if both precision and recall are high [13].

F1 favors if both the precision and recall are similar, however that is not always desired. In some cases such as a spam filter, it would be preferable to have a high recall so almost every spam email is properly filtered but it may also flag some important mails as spam. The other situation is with a high precision so it only flags spam it is very confident in, meaning that important mails are not filtered, but spam may be avoided by the filter given then low recall. This is known as a precision/recall Trade-off[13, p.92].

Confusion matrix

When validating a model, it can sometimes be desired to determine what type of errors occurs. For a classification problem, this can be especially useful to determine whether there is a pattern in the misclassified data. A method to determine the misclassification distribution is through the use of a Confusion Matrix (CM). The CM is a visual way of presenting the determined classes for the datasets compared to their actual labels. By comparing these two, it can be determined if there is a general issue with separating two specific classes from each others[13]. An example of a CM is seen in Figure 2.15.

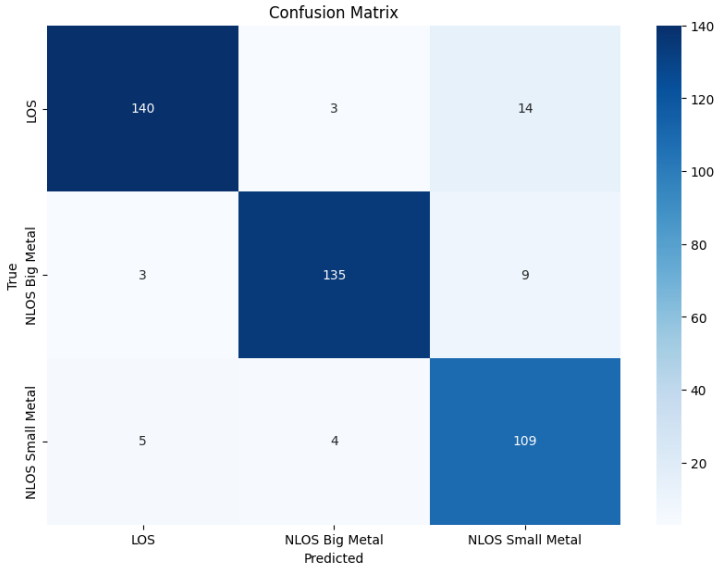


Figure 2.15: Example of a confusion matrix. In the CM, the predicted labels for the test data is compared to the actual labels. It is noticeable, that the NLOS small metal plate wrongly predicted as a LOS or a big metal plate.

In Figure 2.15, it appears that the NLOS small metal plate is difficult for the model to differentiate between LOS and NLOS big plate. When a certain pattern appears in the data like this, it shows difficulty for the model to determine the differences between these classes. Such a pattern can be focused on improving in order to more clearly differentiate between each class through e.g. an enhancement of features through preprocessing of data.

2.2.9 Transfer learning

Transfer Learning (TL) is a paradigm in machine learning that can adapt a trained model to another use-case. This is often a way to train a model faster in case it is adapted to a similar use-case, or if training data is sparse. Many different variants of TL exists based on what the original use-case (source) had as a goal and what the new use-case (target) is. On Figure 2.16 an illustration is made on how transfer learning utilizes general knowledge from a pre-trained model to be adapted to a new use-case.

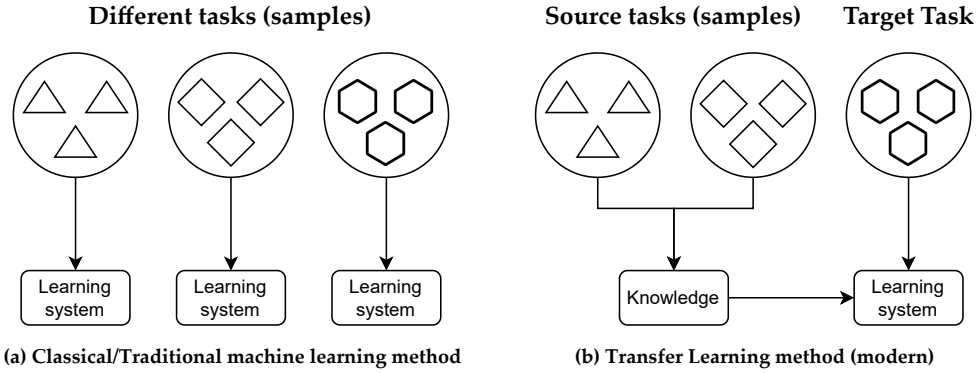


Figure 2.16: Traditionally, models are trained on data for one specific goal, while in transfer learning, a pre-trained model is taken and adapted to a new use-case. Adapted from [28, Fig. 1]

A variant of TL is domain adaptation, where the task between the source and target are the same, but they have different domains and distributions of data [28]. An example of this could be if a model is trained to classify images of dogs vs cats in a properly lit environment, with professional cameras(this is the source domain). Now the model is to be used for classifying dogs vs cats in poorly lit environments, with blurry images from user movements(this is the target domain).

TL can be an optimal way of adapting a model to a new use-case by utilizing the weights of the pre-trained model. However it can cause problems such as *catastrophic forgetting*, where the previously learnt information is forgotten because the new weights changed too much from the old weights [13, pp. 637-638].

To prevent catastrophic forgetting, techniques such as lowering the learning rate significantly and freezing layers are useful. For an description of different types of TL and a more detailed explanation of how layer freezing is done, see section A.5.

2.2.10 Tensors

A fundamental way of representing data in machine learning is through tensor representation, which generalizes vectors matrices up to any dimensionality [12, S. 3.1] as seen on Figure 2.17. Tensors are most commonly understood as multi-dimensional matrices [29].

This representation makes it easy to represent data anywhere from time series data to images with multiple channels for color.

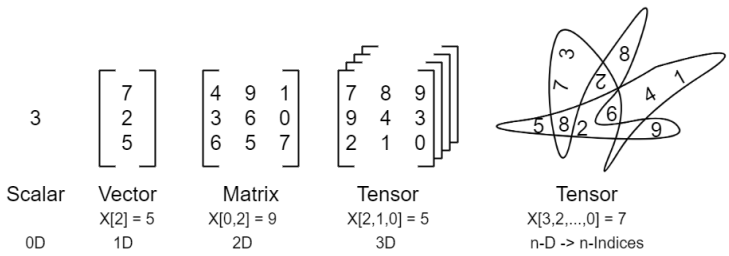


Figure 2.17: Tensors in machine learning can be represented in many dimensions from a scalar to multidimensional matrices. Adapted from [12].

Many DL frameworks utilizes Graphics Processing Unit (GPU) acceleration for tensors as GPUs have more Arithmetic Logic Units (ALUs) than Central Processing Units (CPUs). Many tensor operations are matrix and vector operations and can therefore be done highly parallelized, including tracking of gradients used for BP using Automatic Differentiation (AD) [30].

To utilize GPU acceleration, the tensors will have to be moved from the host device to a GPU by copying the data. The GPU can now finish all its necessary tensor operations before the result can be transferred back to the host device as seen in Figure 2.18. An important aspect to note is that copying between devices can result in a memory transfer bottleneck [31] and is therefore an overhead that needs to be accounted for.

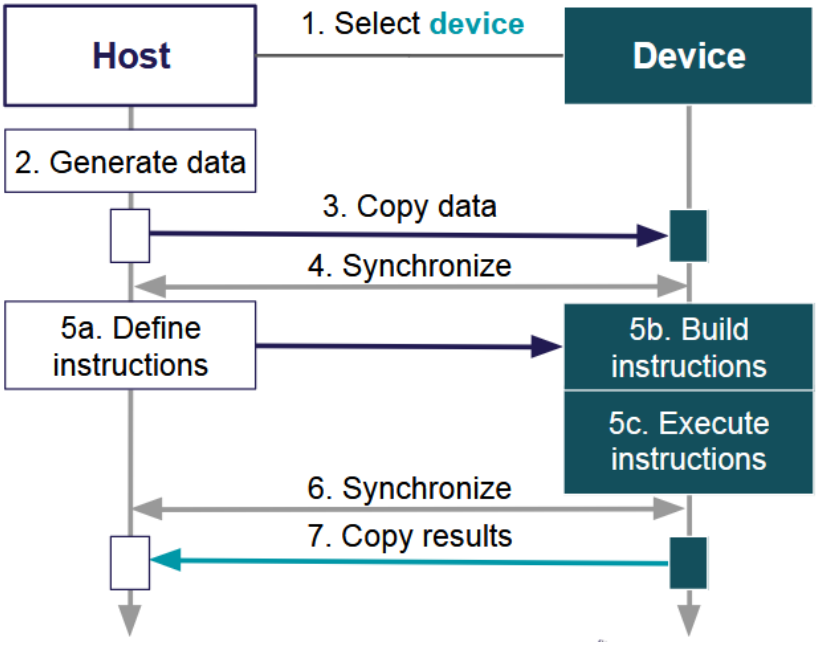


Figure 2.18: Performing GPU operations requires the host to copy all the data from the CPU memory space to the GPU memory space. There will be a generally a significant speedup of computation time on a GPU, although the process of copying data might be costly. Taken from [32]

3. Dataset

To train a machine learning model, data is needed. The data provided for this project is presented in the following section, where the size of the dataset and the physical properties of the test setups are presented.

Collecting the data

The first test setup is shown in figure 3.1, and is what makes up the dataset which will be referred to as the original (OG) dataset. There are two antennas in the room, one of which is used to transmit the signal and the other one is used to receive the signal. The signal has a frequency of 5 GHz, and the sampling rate is 25 GHz. The transmitting antenna is pointing away from the receiving antenna. There is a RIS in front of the transmitting antenna to reflect the signal to the receiving signal. An obstruction is placed between the two antennas to create non line of sight (NLOS). There are two kinds of obstructions, a small and a big metal plate. These plates are used to simulate different levels of obstructions, with the larger plate representing a more significant blockage of the signal.

The distance between the two antennas are 366 cm, and the metal plate is placed in between. The length and width of the entire room are 729 cm and 538 cm, respectively. Additionally, the distance between the transmitting antenna and the RIS is 60 cm. All the physical measurements can be seen in figure 3.1.

Additionally, two other datasets were also provided being from the High Frequency lab (HF) and Biggest Chamber (BC), which is an anechoic chamber. The HF dataset provided consists of LOS measurements, and NLOS measurements using the small metal plate. For the BC measurements, it consists of only NLOS data for both smaller and bigger metal plate. The two measurement locations and room dimensions can be seen in appendix B. To fully achieve all reflections of these locations, the sampling frequency differs from the original dataset, with HF having a sampling frequency of 7.5 GHz, and BC being 6.9 GHz. Each measurement still consists of 1801 samples of both time (s), magnitude (dB) and phase (deg).

All measurements were taken by changing the horizontal angle of the transmitting antenna and the rotation angle of the receiving antenna. The experiments recorded the signal's dB and phase as the channel impulse response (CIR) in relation to the reflections from the environment of the room, which were then stored in a **.csv** file.

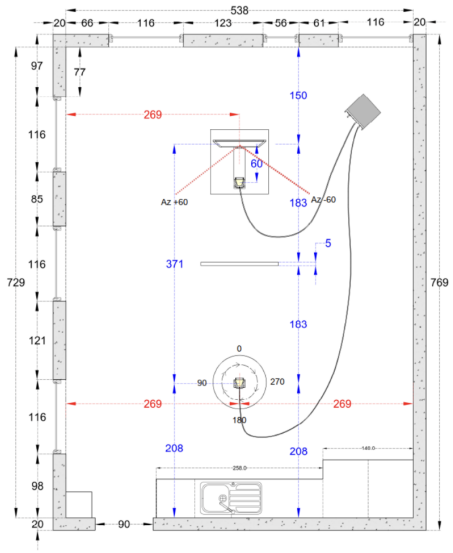


Figure 3.1: The physical properties of the test setup with the distances shown in centimeters. The thickness of the metal plate is shown in millimeter

Data Extraction

As previously mentioned, two types of data are provided, namely LOS and NLOS data. The filenames are a combination of numbers and letters, whereas the first number indicates the different rotation angles of the turntable, the second number indicates the elevation angle of the RIS, and the last number indicates the azimuth angle of the RIS. For example, 0.0_E_0.0_A_-10.0.csv would be where the turntable is at 0 degree and the RIS is set to 0 degree in elevation and -10 degree in azimuth.

In all the datasets provided, there are three different variables, which are time (time of sample reception), dB (amplitude) and degree (phase). A representation of the data is shown in Table 3.1 for the OG LOS case and Table 3.2 for the HF NLOS case.

Time(s)	S21(DB)	S21(DEG)
0.00	-90.64	108.55
$38.9 \cdot 10^{-12}$	-90.75	178.07
$77.8 \cdot 10^{-12}$	-90.86	-112.45

Table 3.1: First 3 data samples of the 0.0_E_0.0_A_-10.0.csv file in the original LOS dataset.

Time(s)	S21(DB)	S21(DEG)
0.00	-96.11	179.48
$133.33 \cdot 10^{-12}$	-96.72	60.94
$266.66 \cdot 10^{-12}$	-97.44	-57.25

Table 3.2: First 3 data samples of the 0.0_E_0.0_A_-40.0.csv file in the HF NLOS dataset.

As seen in tabels 3.1 and 3.2, the time values vary greatly making it inconsistent across different environments. Therefore, it does not represent a general distinction between LOS and NLOS but rather the environments.

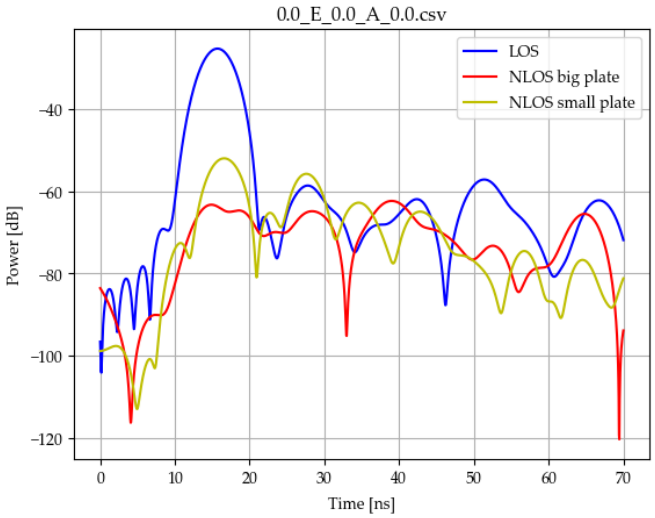


Figure 3.2: Plot of three dataset from LOS (blue), NLOS big plate (red), and NLOS small plate (yellow). This is from the measurement '0.0_E_0.0_A_0.0.csv' from the OG dataset.

The datasets received for the project was measured in an interference-free location with reflections on objects and walls. The interference-free data allows a higher focus on the CIR and its reflections. On the graphical representation of the data on Figure 3.2 it can be seen that the amount of noise on the data is minimal.

3.1 Feature Extraction

In this section, the filtering method used in the project will be described. Hereafter, two ways of normalization will be presented, followed by a description of Kurtosis. Lastly, the delay spread, phase and group delay of wireless signals will be discussed and analyzed.

3.1.1 Filtering

A signal transmitted over a wireless channel is a stochastic process that contains an uncertainty due to the noise over the wireless channel. For estimating the original signal $\hat{Y}(n)$ propagated given the noisy signal, a Kalman Filter (KF) can be used.

The KF is a recursive algorithm that on a high level has two steps. 1) The prediction step, where the KF estimates the next value given the current system model. 2) The updating step, where the system model is updated based on the error between the newest measured value and the predicted value calculated during the prediction step ($R(n|n)$). This process is illustrated in the block diagram seen on figure 3.3.

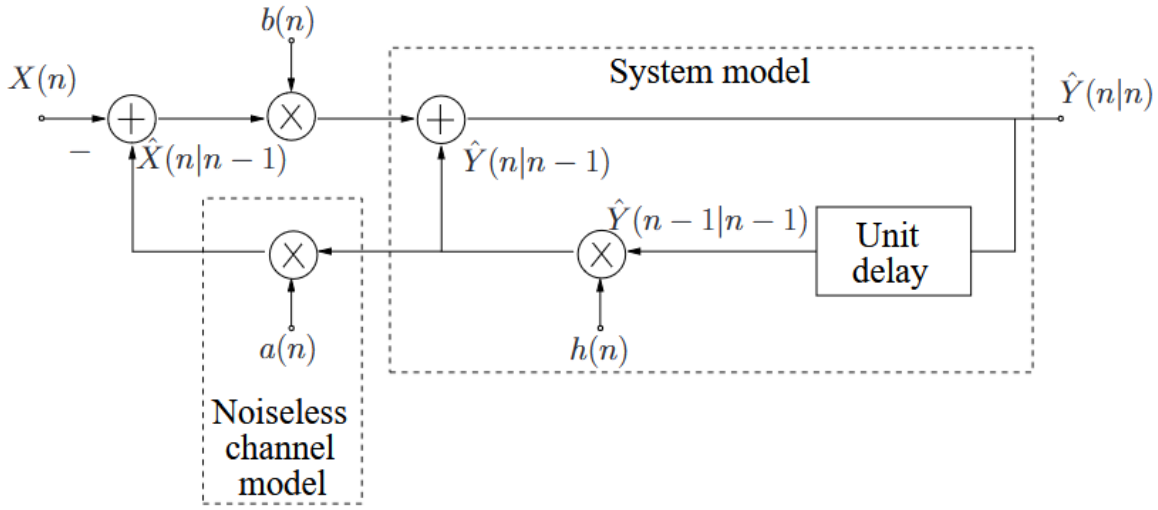


Figure 3.3: Block diagram of how the prediction and update steps work together to estimate the next state $\hat{Y}(n|n)$ of the system. Figure source: [33]

Firstly, the initial state of the system $\hat{Y}(0|0)$ has to be set, and the error $R(0|0)$ is set to the measurement variance at initialization. Utilizing the kalman gain, the KF can weigh how much influence the error between prediction and measurement, together with information about how noisy the measurements and prediction, might have.

Prediction step

To estimate the current system state, the KF uses the previous state with a state transition coefficient $h(n)$ to predict the new state, and takes the point process noise into account. The KF uses the following equations for the prediction step [34, p.436]:

$$\hat{Y}(n|n-1) = h(n) \cdot \hat{Y}(n-1|n-1) \tag{3.1}$$

$$\hat{X}(n|n-1) = a(n) \cdot \hat{Y}(n|n-1) \tag{3.2}$$

$$R(n|n-1) = h(n)^2 \cdot R(n-1|n-1) + \sigma^2_{ZZ}(n) \tag{3.3}$$

Where:

$\hat{Y}(n\|n-1)$	LMMSE estimate of $Y(n)$ based on $X(1), \cdots, X(n)$
$h(n)$	Known state transition coefficient
$\hat{Y}(n-1\|n-1)$	Previous prediction
$\hat{X}(n\|n-1)$	LMMSE estimate of $X(n)$ based on $X(1), \cdots, X(n)$
$a(n)$	Known process noise coefficient

Updating step

Given that the KF has made a new prediction on what the estimated new system state should be, the updating step will now compare the error between the predicted system state with the

measured value. The KF uses the following equations for the update step:

$$\hat{Y}(n|n) = \hat{Y}(n|n-1) + b(n) \cdot (X(n) - \hat{X}(n|n-1)) \tag{3.4}$$

$$R(n|n) = [1 - b(n) \cdot a(n)] \cdot R(n|n-1) \tag{3.5}$$

$$b(n) = \frac{a(n) \cdot R(n|n-1)}{a(n)^2 \cdot R(n|n-1) + \sigma_w^2} \tag{3.6}$$

Where:

$\hat{Y}(n\|n)$	LMMSE estimate of $Y(n)$ based on $X(1), \cdots, X(n)$
$b(n)$	Kalman gain
$R(n\|n)$	Mean Squared Error (MSE) of $\hat{Y}(n\|n)$
$a(n)$	Process noise coefficient
σ_w^2	Known measurement noise variance

For a deeper explanation of the KF, see section A.6.

3.1.2 Normalization

In situations where data reaches values in very high numbers or situations where the shape of the data is more important than the values, it is useful to normalize the data. Many different types of normalization exists such as normalizing between 0 and 1, normalizing the length of the data (L2-normalization) or standardizing the data. In this section the normalization methods L2-normalization and z-score normalization will be described.

The L2-normalization operates by finding the L2-norm value and dividing each sample by this. The L2-norm is normally called the euclidean norm due to the scaling of the data getting a length of 1[35]. The L2-norm is found as the root of the sum of the squared datapoints, and the formula is seen on

$$x_{normalized} = \frac{x}{||x||_2} = \frac{x}{\sqrt{\sum_{i=1}^{n} x_i^2}} \tag{3.7}$$

where:

$x_{normalized}$	L2 Normalized datapoint
x	Original datapoint
$\|\|x\|\|_2$	L2 Norm

The L2-norm maps the data in the range [-1;1] depending on each sample's relation to it.

Another option is the Z-score normalization. Z-score normalization operates by making the mean value of the dataset 0 and the standard deviation to 1. To apply the Z-score normalization, the mean and deviation are found for the dataset and converted by using the following equation [36]:

$$z_i = \frac{x_i - \mu}{\sigma}$$

Where:

- z_i Normalized datapoint
- x_i Original datapoint
- μ The mean
- σ The standard deviation

Since z-score normalization represents the data as a standard distribution, it becomes easier to see outliers.

3.1.3 Kurtosis

Kurtosis is used to describe the degree of "tailedness" in a probability distribution or real-valued random variable. Tailedness describes, how often outliers occur in a dataset, and the use of sample kurtosis is an estimate of the kind kurtosis of the data. The kurtosis is defined as the standardized fourth moment of a distribution, which can be calculated by dividing the central fourth moment with the standard deviation to the power of four.

Excess kurtosis tells how the tailedness of the distribution compares to a normal distribution. The excess kurtosis is calculated by subtracting 3 from the kurtosis. To calculate the sample's excess kurtosis, the following equation is used for getting an unbiased estimate of excess kurtosis [37]:

$$\text{kurtosis} = \frac{n(n+1)}{(n-1)(n-2)(n-3)} \frac{\sum(x_i - \overline{x})^4}{(\sum(x_i - \overline{x})^2)^2} - 3\frac{(n-1)^2}{(n-2)(n-3)} \tag{3.8}$$

where

- n Sample size
- x_i Observations of the random variable x
- $\overline{x}$ Mean of the variable x

After calculating the excess kurtosis, it can be used to tell whether the distribution is platykurtic, mesokurtic or leptokurtic. When the excess kurtosis is negative, it is a platykurtic distribution, which is thin-tailed, meaning the frequency of outliers are low. Normal distributions have an excess kurtosis of 0, meaning they are mesokurtic with a medium frequency of outliers, and hence are medium-tailed. The final distribution is leptokurtic, which is when the excess kurtosis is positive, meaning a high outlier frequency. This distribution is also called fat-tailed [37]. A comparison of the distributions can be seen in Figure A.1 in section A.2.

3.1.4 RMS delay spread

The average and root mean square (RMS) delay spread is used to determine the delay spread relative to the channel power delay profile, as it is a good measure of the variance about the average [38, Chapter 3].

Delay spread is an important metric in telecommunication that describe the time dispersion caused by multipath propagation. It represent the range of the time between the arrivals of signal through different paths from the transmitter to the receiver.

A transmitted signal is described mathematically as:

$$s(t) = \mathrm{Re}\left(u(t)e^{j2\pi f_c t}\right) \tag{3.9}$$

where

$u(t)$	The complex signal
f_c	The carrier frequency
t	Time

When neglecting the noise, the delay spread will affect the received signal as the sum of the LOS path and all the multipath components:

$$r(t) = \mathrm{Re}\left(u(t-\tau_n(t))e^{j(2\pi f_c(t-\tau_n(t))+\varphi_n}\right) \tag{3.10}$$

where

τ	The delay time of reception
n	The relevant multi path component
φ	The phase delay

The RMS delay spread quantifies the spread of the arrival times of multipath components relative to the strongest path, typically the LOS path. The RMS delay spread is calculated using the Power Delay Profile (PDP) denoted $A_c(\tau)$, defined as the autocorrelation of the time-invariant channel impulse response, which provides a statistical representation of the power of the received signal as a function of delay. The average RMS delay spread is then defined as

$$\mu_{T_m} = \frac{\int_0^\infty \tau A_c(\tau)d\tau}{\int_0^\infty A_c(\tau)d\tau} \tag{3.11}$$

and

$$\sigma_{T_m} = \sqrt{\frac{\int_0^\infty (\tau-\mu_m)^2 A_c(\tau)d\tau}{\int_0^\infty A_c(\tau)d\tau}} \tag{3.12}$$

where

T_m	The time delay

By analyzing the RMS delay spread, system designers can assess the severity of multipath fading and design countermeasures, such as equalization or spread spectrum techniques to mitigate its effects. This makes the RMS delay spread an essential parameter for the design and optimization of modern wireless communication systems [38, Chapter 3].

3.1.5 Phase and group delay

When a signal is transmitted through a dispersive environment, some delay will be introduced to the signal at the receiver relative to the transmitter. In a distortionless system, the phase will have a linear slope, but in an environment with different disturbing elements, the phase will be distorted. This leads to a phase shift at the receiver, which can disturb the reception of the signal. This phase distortion is measured using the group delay [39]. If a phase delay is linear, this will be shown by a constant group delay. To calculate the group delay, the phase is differentiated with respect to the angular frequency of the signal. This is represented as [39]

$$\tau(\omega) = -\frac{\mathrm{d}\phi}{\mathrm{d}\omega} \tag{3.13}$$

where omega (ω) is the angular frequency of the signal in radians. The group delay τ is expressed in units of time (nanoseconds) [39]. The phase and group delay can be seen in figure 3.4 for the same scenario.

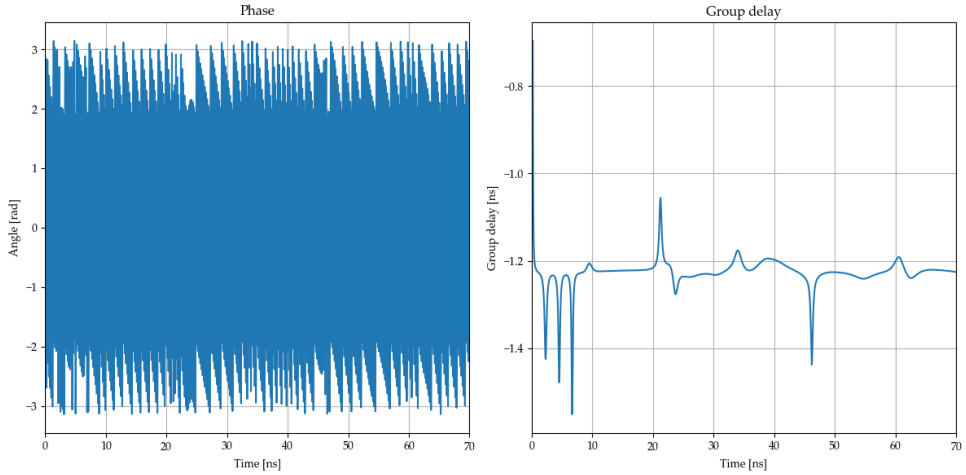


Figure 3.4: Plots of the phase and group delay of a received signal with a frequency of 5 GHz. Both x-axis' are in nanoseconds. The y-axis of the phase plot is in radians, and the y-axis for the group delay plot is the ratio of change in the phase.

Utilizing the formula 3.13 for group delay, it is possible that objects in the environment can be derived, as different objects will have different effects on the group delay. Furthermore, as the group delay comes out in nanoseconds, it is possible to calculate the time of flight (ToF) of the signal, since the group delay with the lowest value would be the one being the most direct. However, this depends on whether the antennas are LOS or NLOS, as the lowest group delay in an NLOS case could prove to be the first reflected signal, hereby not the direct line between the antennas [39].

In [39] the group delay obtained from an antenna is dependent on the antenna's Q factor. Here it is shown that, the group delay of a low Q antenna is around 1.3 ns, while for a high Q antenna the group delay is 22 ns. This can lead to incorrect group delays obtained from antennas, hereby always miscalculating the distance between the antennas using this method, if the Q factor of the antennas are not known.

4. Research Questions

The development of Reconfigurable Intelligent Surface (RIS) has given a great opportunity to enhance the communication paths in environments with lots of movements and obstacles. The RIS does this by providing an alternative route between the transmitting unit to the user, i.e. the transmitter (Tx) to receiver (Rx). This can improve the signal strength in otherwise noisy environments and as a result improve the SNR.

To know when the direct path is blocked, and the path provided by the RIS is optimal, a machine learning model can be trained to recognize the received signal, when it is line of sight (LOS), and when it is non line of sight (NLOS). From this, the problem definition is specified:

How can a machine learning model be trained to recognize a line of sight and non line of sight signal from a transmitter, and is deep learning models better than classical machine learning models?

4.1 Questions

1. How does preprocessing of data and data augmentation affect the models accuracy?
2. How can a general model be designed to be effective in different environments and is transfer learning a possible solution?
3. How accurately can the distance between a transmitter and receiver be estimated in an NLOS situation using current data?

Explanation of questions

1. By preprocessing the data using different methods the contrasts between the classes may be more obvious. Using domain knowledge to extract useful features of an RF signal, it is researched what the most effective combination of input data which achieves the highest accuracies are.
2. By adding noise to the signal, it will be researched what possible accuracies can be achieved for a generalized model, and if transfer learning can improve the accuracies to make a more generalized model.
3. An extension of the LOS and NLOS detection will be designed, where the direct distance between Tx and Rx is estimated utilizing the current data given in section 3, regardless of LOS or NLOS case. Therefore, possible methods for calculating the distance between Rx and Tx will be researched.

5. Model Design and preprocessing

Following the research questions stated in chapter 4.1, an initial system has been designed. This overall system is designed with the intent of achieving the maximum accuracy of determining the classification, and determine the physical distance from the receiver to the RIS for several environments using localization. This chapter will focus on the preprocessing methods of the data, and the design of a classification model. The localization part in figure 5.1 is designed in order to seek answers for research question 3 and is described in chapter 8.

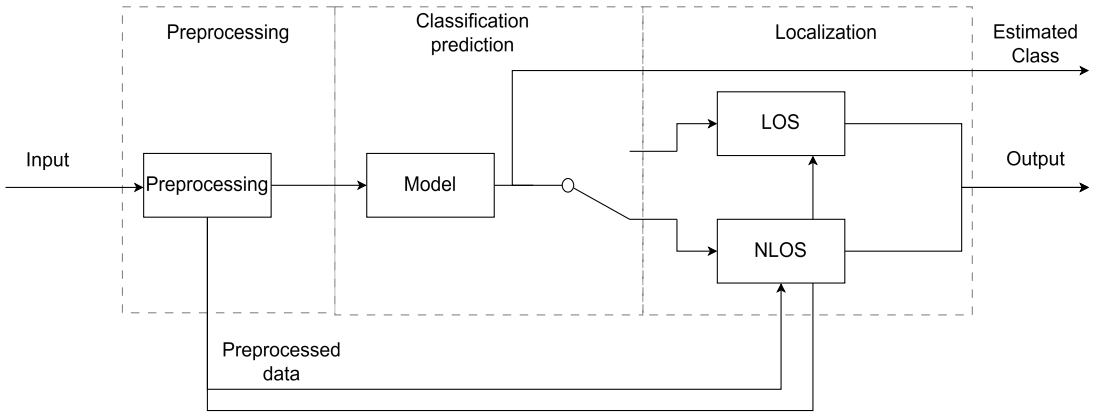


Figure 5.1: Overview of the different modules and how they interact. Input is preprocessed before being sent into the model and the localization algorithm, after which the class output from the model determines which algorithm to utilize. The output is a length in meters and the determined class.

The overview in figure 5.1 shows the relation of the different functions between each other. The input is the received data, which is preprocessed before being transmitted to the model. It then determines the classification and utilizes different localization methods depending on the class, with the physical distance and class being the output of the functions.

For the model to perform in the context on Figure 5.1, it is required to perform the same preprocessing tasks during training as during deployment. The preprocessing and model design will take several matters such as noise and size of the model into account based on the pre-analysis in section 2.

5.1 Preprocessing of the data

This section aims to design preprocessing methods for increasing the data available for the model, or making the data more generalized to see how it affects the model's accuracy.

5.1.1 Noise

As the measured data is with marginal amounts of noise, it shows the actual reflection behavior within the room. However in a realistic situation, noise is expected to appear, such as additive white Gaussian noise (AWGN) applied across all the data. Thus to emulate this behavior, the application of AWGN is a possible preproccessing method.

The AWGN can be found through finding the signal power in the provided data, and scaling the generated noise to match a predetermined SNR before adding it across all measurements in the given dataset.

Another option could be the use of randomized noise. The noise level added to all samples in each measurement matches a random SNR each, which lies in a range between a predefined minimum and maximum.

5.1.2 Filtering

As a noise method has been designed with the purpose of emulating present noise, a method to reduce the effect of the noise should be designed, hereby emulating a real life scenario.

The Kalman filter for reduction of the MSE in a discrete time situation was described in section 3.1.1. The kalman filter is useful for dynamic systems, and the filter will be designed for the purpose of reducing the noise level to achieve a better SNR and increasing the visibility of the data's features.

5.1.3 Normalization

When the signal propagates, a certain loss of energy will appear in the form of path loss. The power of the signal will therefore display the distance it has traveled between transmitter and receiver. This however may be problematic if the model focuses on power levels. This is due to situations where the distance between Tx and Rx are different compared to the training data. To handle this, the data can be normalized in two different methods, the L2-normalization and the Z-score normalization, which are described in section 3.1.2.

5.1.4 Group Delay

Another method that can be utilized is the group delay. The group delay describes the rate of change of the phase of the signal with respect to the angular frequency, which is further described in section 3.1.5. Instead of representing the phase as S21(DEG), the group delay can be calculated and used. The group delay can also be used as an addition to the dataset, as it helps describe the path of the signal, hereby increasing the information provided to the model.

5.1.5 Fast Fourier Transform

The Fast Fourier Transform (FFT) is another method of representing the data. Instead of representing the S21(dB) data in the time domain, using FFT on the data will convert the data to the frequency domain. The FFT can be used as additional data to provide the model. FFT can be seen as a potential method, as it is possible that the NLOS situation will contain slight frequency differences.

5.1.6 Kurtosis

As the signal will follow a random distribution given it is a stochastic process, kurtosis can be used to find the tailedness of each measurement. As described in section 3.1.3, the excess kurtosis can be used to describe the distribution of a given measurement. The difference between LOS and NLOS data could be associated to a different kurtosis distribution, giving the data more information about possible differences between the two situations.

5.1.7 RMS Delay spread

The RMS delay spread quantifies the spread of arrival times of different multipathing components, relative to the strongest path in the signal. The delay of the stronger power lobes in the Power Delay Profile (PDP) will account for a higher RMS. The difference between NLOS and LOS may be able to bring forth a difference in RMS delay spread, which may be useful for training the model. Hence, an RMS delay spread function is designed to obtain the RMS delay spread of the received signal, thus possibly increasing the difference of the received signal when LOS or NLOS.

5.2 Determination of model type

As described in the initial problem statement, the machine learning model needs to distinguish LOS from NLOS data between a transmitter and receiver, which makes the problem a classification problem. Given the labels are known, supervised learning is used to train the models. This section will aim to design three different classification models, namely DNN, LSTM and SVM, which were described in 2.2.6.

5.2.1 DNN design

The model is designed by adding, changing or tuning the model's hyperparameters, as described in 2.2.7. For a DNN, these hyperparameters are:

- Number of layers
- Amount of neurons in each layer
- The dropout rate
- Activation functions
- Batch normalization

There are no definitions that help determine the choice of hyperparameters, as the model itself depends on the received data, which makes tuning the model an iterative process. Though, there are some general design rules, depending on the type and size of the data.

The size of the model is an important aspect to determine. If the model is too small it will fail to properly discern the features which is known as underfitting. If it is too large it may overfit or the size fails to improve the model.

The model will be built upon a classification problem with two classes, LOS and NLOS. The model consists of an input layer, four hidden layers and an output layer.

The input layer's size is determined by the amount of samples in a measurement and the amount of columns of data used, such as S21(DB) and S21(DEG). The batch size was determined to be a size of 32. The amount of neurons are chosen to be values to the power of 2, due to it potentially increasing the efficiency of the model and increasing the speed by allowing the model to map values 1 to 1 between neurons [40]. Each input will have at least 1801 values, being the amount of samples in a single measurement, which can change depending on the amount of additional data included. The first hidden layer's size is determined to be of 1024 which is 2^{10}. It is also chosen for the model to consist of four hidden layers, reducing by half for each layer. The second hidden layer's size is therefore 512, the third hidden layer's size is 256, the last hidden layer's size is 128, and lastly the output size is 2 corresponding to the binary classification between NLOS and LOS.

In between each layer, data modification occurs in the form of ReLU, Batch normalization and dropout. Batch normalization can speed up the training process by reducing the data-size and it can in some cases remove the necessity of dropout rate[26]. The activation function is decided to be a ReLU to potentially improve the performance between the layers. The dropout is the last function between each layer that is applied in order to randomize the connections between neurons and improve performance.

5.2.2 SVM design

Among the hyperparameters for the SVM that are planned to be utilized are the parameters:

- Kernel
- Order of the poly kernel
- Gamma parameter of the RBF kernel
- Regularization parameter C

There are many different types of kernels for the SVM, some achieving better than others on a specific dataset. For this project, the Poly and RBF kernel is utilized for the SVM. As the kernel can be a Poly or rbf kernel, the order of the poly is important. The standard order for the Poly kernel for SVM is 3 [27], meaning that the polynomial separating the classes is of the third order. Both kernels use a gamma parameter, which by default is defined as:

$$\gamma = \frac{1}{N \cdot \sigma^2} \tag{5.1}$$

The regularization parameter C controls the tradeoff between a low error and a minimization of the model complexity to avoid overfitting. A low C will encourage a simpler model where errors are allowed while a high C will attempt to classify all samples correctly. A high C sounds optimal, but it may cause the model to overfit, thereby it being a balancing act.

5.2.3 LSTM

Similar to the DNN in section 5.2.1, the LSTM requires definitions for the following:

- Number of layers
- Hidden size
- Number of classes

- The dropout rate

Given that it is an LSTM, the input data is a sequence of data with a certain length and therefore the input is of size 1. The amount of layers is increased to two with the purpose of increasing the probability of achieving a better model. Having more than one layer also allows for the use of dropout. The number of classes is the same as for the DNN and SVM and the dropout is in between each layer. The hidden size for the LSTM is similar to the amount of neurons in each layer for the DNN, except each hidden size contains the short term memory, long-term memory and the gates that regulate the flow.

5.3 Validation method

When training a model, the goal is to achieve the lowest loss or highest accuracy for the data it has been fed with. This can result in overfitting due to having no data to compare with. One way of avoiding this is to split the dataset into three subsets of the data. These subsets are the training, validation and test datasets. The specified distribution of the datasets is 75% test, 10% validation and 15% test data to ensure enough data for validation and test for the results to be comparable to the models actual performance.

The validation dataset is used during training where the model is evaluated on it. This ensures that even if the model is overfitting on the training data, it would be observable if the validation accuracy would be too low. This allows for a certain level of safety towards overfitting.

To verify the performance of whether it is overfitted, the accuracy of the test data is determined. The accuracy of the test-set will show the model in a test situation, as it has not seen the test-set before. If the accuracy is lower than the validation accuracy by a lot, it appears to show that the model is overfitted.

5.4 Loss Function

The loss function is important part of training a model. It is responsible for determining the difference between the predicted result and the actual result. The error in estimating the actual label is then used in an optimizer to reduce the error for the data. To determine the error of the classification issue, different functions have been introduced in Table 2.1 in chapter 2.2.4.

For this classification problem, both binary cross entropy, cross entropy and focal loss are all reasonable choices for the model. As an extended application of the model would be to expand the classification problem to 3 classes, the binary cross entropy is not suitable due to being a binary classifier. This leaves both cross entropy and focal loss as possible candidates for the loss function. As the model also aims to increase generality, it is important to focus on hard to classify samples, as these will likely be present when different datasets are involved. Due to this, the focal loss function is chosen, which allows for a higher punishment for confident wrong predictions made by the model and a higher focus on the samples the model appears to have a hard time classifying, thereby reducing the overall loss.

Another option for the loss functions to achieve better estimates is to allow label smoothing. Label smoothing is the act of reducing the binary class value to a lower value instead of 1 or 0. Label smoothing of a factor 0.05 will reduce the label value from 1 to a 0.95 which can help reduce the loss. Another positive benefit that label smoothing has is that it may help preventing

overfitting of the model [41] as this is a potential consequence of focusing harder on classifying labels.

5.5 Evaluation tests

While a model is validated on a validation set under training, it is possible for a high accuracy for that specific sweep. To verify the performance and accuracy of the model, the model needs to be tested with several methods. There exists several different methods of validating, with methods such as the K-fold cross validation or testing on a separate test dataset. In this section, the used evaluation tests will be described and the reason for utilizing it.

5.5.1 K-fold cross validation

Tuning the hyperparameters to achieve a high frequency is one neccesary part of training a model, however this also opens up for an error in the form of overfitting to a specific validation set. Described in section 2.2.8, the issue of overfitting is problematic for making a generalized model that works on other data. It is a possibility to randomize the data that is used for the validation set to avoid tuning to a specific set of validation data. To determine the overall performance of the model on all the data, K-fold validation can be used. The K-fold validation will show how well the model performs on different splits of the data. Because the model is tested on different subsets of the data, K-fold validation reduces the risk of the model relying on a single, potentially non-representative train-test split. If a high variance occurs in the accuracy for the different folds it suggests potential overfitting.

5.5.2 Confusion matrix

The confusion matrix is a method of comparing the estimated labels to the actual labels for a test dataset. The comparison is visualized in a matrix-plot showing the true estimations on the diagonal. Showing the distribution of the estimated labels can help determine how the labels are split in the test-dataset and the distribution of the misclassified samples. Knowing the distribution will help determining whether the split is properly made with an expected spread of classes and whether a certain pattern appears in the misclassified samples. Knowing the distribution of correctly and misclassified classes can help determine the performance and a potential point of improvement.

5.5.3 SNR Loop

As the system needs to be applicable in multiple scenarios, it is important to test its resilience to changes in the environment, such as the amount of noise present as described in section 2.1.2. A way to test this resilience would be to take a predetermined training set from the data provided to the model, and apply noise to it in varying SNR's.

A flowchart describing this test can be seen in Figure 5.2, where a range for SNR is first defined, as well as the length between each step. From here, the test would loop through each SNR, and get an accuracy that would then be plotted against the SNR so the relationship between the two can be found. It is expected that a system which has not been trained with noise added to the training data, would perform poorly at a low SNR, where it will get better the higher the SNR becomes. The model will be tested on an SNR going from -25 dB to 60 dB, as this is below a very bad signal, and above an acceptable SNR [42].

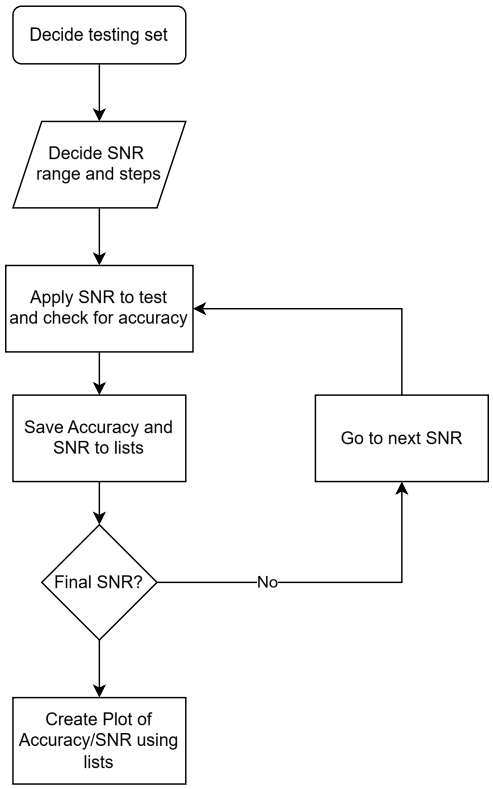


Figure 5.2: Flowchart of the SNR Loop testing the systems resilience.

5.5.4 Generalization test

The final test will be designed to check if a certain model is useful in different environments. A simple way to do this is by running the same tests, but using different datasets as the test set. The reason for using the same tests is due to the need of the results being comparable to one another. This way, it is easy to see how big an impact a change of physical location has on the performance of the model, and how general the final model is. This test can also be used to see the effect of transfer learning, or other methods that can be used for making a model more general.

The test should summarize the results of the confusion matrix and SNR. For easy comparison, the confusion matrix should be placed side by side, with their accuracy and total correct and false predictions described. The SNR test can be plotted on the same graph, giving easy view of comparison, as the graphs can be seen side by side.

5.6 Design Summary

Throughout this chapter a system has been designed with the options of three different models to be implemented. The different possible preprocessing methods that will be used and tested have been designed. The models; DNN, LSTM and SVM have been designed regarding their hyperparameters as well as their sizes. These models are going to be implemented and tested to determine the best performance between each of them. The specific methods used for validating and testing the models have been designed.

6. Implementation

Based on the system design seen in chapter 5, this chapter will focus on the actual implementation of the system. This chapter will write about the implementation of NLOS identification, while the localization implementation will be described in chapter 8. The DNN and LSTM are implemented using the Python ML framework PyTorch [43] and the SVM is implemented using the SVC implementation [27] included in the scikit-learn framework[1] [44].

6.1 Loading datasets

The dataset mentioned in chapter 3 is provided in a Comma Separated Values (CSV) format, which is loaded using the pandas framework [45] and its file structure called *dataframes*, using the `read_csv()` method. The head of a dataframe with a loaded CSV file can be seen on Table 6.1, where the metadata at the top and bottom of the CSV are removed.

Index	Time(s)	S21(DB)	S21(DEG)
0	0.000000e+00	-95.394760	-30.964529
1	1.333333e-10	-96.196541	-146.034180
2	2.666667e-10	-97.192039	100.559200
3	4.000000e-10	-98.324020	-10.398050
4	5.333333e-10	-99.433479	-118.024430

Table 6.1: Dataframe head of a CSV file from chapter 3. Dataframe does not include the meta data at the top of the CSV file and neither the bottom.

The datasets are separated into different variables i.e. LOS cases between the Original (OG) and High Frequency (HF) dataset, and the same for NLOS with the different case as seen on Table 6.2. Note that most of the HF NLOS dataset has been removed due to a lot of the measurements not being consistent with the 0 degree elevation, as in all other datasets.

[1] Often referred to as sklearn, which will be used from here on out.

Variable name	Scenario	Total CSVs
los_dataset	LOS in OG dataset	937
nlos_big_metal_dataset	NLOS in OG dataset with a big metal obstruction	937
nlos_small_metal_dataset	NLOS in Original dataset with a small metal obstruction	937
hf_los_dataset	LOS in HF dataset	858
hf_nlos_dataset	NLOS in HF dataset with a small metal obstruction	91

Table 6.2: Overview of the datasets being used and the number of CSVs in the dataset. Many CSVs from the HF NLOS datasets have been removed, to keep all measurements consistent at 0 degrees elevation.

6.2 Preprocessing

As described in chapter 5, the preprocessing functions will add noise, normalize the datasets, get group delay, Fast Fourier Transform, finding the kurtosis, RMS delay spread and peak amplitude of the provided data. An overview of how the functions can be enabled, can be seen on Table 6.3. These implementations will mostly make use of the library numpy [46], as it allows augmentation and analysis of large amounts of data.

Category	**Option**	**Type**	**Description**
Training Mode	resume	bool	Use existing model weights
	TL	bool	Enable transfer learning
Data Processing	Normalize	bool	Apply data normalization
	train_With_FFT	bool	Preprocess data using FFT
	train_With_GroupDelay	bool	Process group delay (in Rad)
	apply_Filter	bool	Apply Kalman filter
	s_Time	bool	Include time domain data
	s21_db, s21_deg	bool	Include S21 magnitude and phase
	train_kurtosis	bool	Include the kurtosis of the signal
	rms_delay	bool	Include the RMS delay spread
	PeakAmp	bool	Include the peak amplitude
Noise Handling	add_Noise	bool	Add colored noise to data
	train_With_Noise	bool	Train model with noisy data
	Desired_SNR	float	Set SNR level (in dB)
	random_Noise	bool	Enable random noise
	dB_min, dB_max	float	Set noise range
Data Selection	mixed_data	bool	Mix original with additional data
	train_With_Additional	bool	Enable additional data usage
	HF	bool	Use high frequency chamber data
	BC	bool	Use big chamber data
Model Configuration	Activate_Scheduler	bool	Enable learning rate scheduler
	labels	int	Number of classification categories

Table 6.3: Configuration options for training pipeline.

6.2.1 Noise

For adding noise to the data, two functions exists that adds noise to the S21(DB) column seen on Table 6.1. 1) `addNoise()` that takes a fixed SNR for all datasets 2) `addRandomNoise()` gives a

random SNR between a minimum and maximum SNR on a file basis. When adding noise, there will be two copies of all the datasets 1) Unedited datasets 2) Datasets with noise augmentation. This will be used for testing (See subsection 5.5.3). The result of adding noise to the signal can be seen on Figure 6.1.

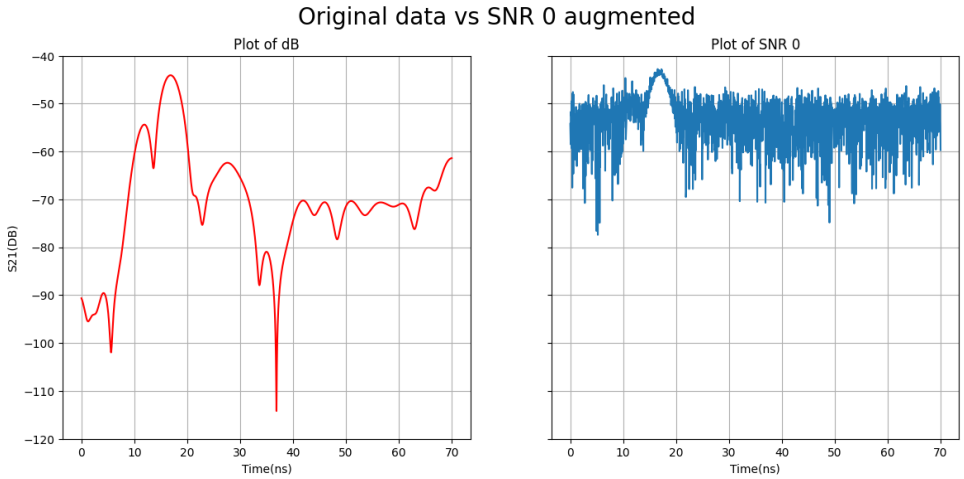


Figure 6.1: Noise function applied to data, with an SNR of 0.

6.2.2 Filtering

If noise is added to the signal, it is filtered to reduce the noise. For this a kalman filter is used to reduce the noise. Given that the data is a Channel Impulse Reponse (CIR) with a certain level of applied AWGN, the state transition coefficient and process noise coefficient are set to a constant value of 1.

The filter functions in section 3.1.1 are then iterated through the length of the data where $\hat{Y}(n|n)$ is saved for each iteration, giving the filtered response. The pseudocode for this is:

Algorithm 2 Kalman Filter Pseudocode

Input: Initial estimates $\hat{Y}(0|0)$, $R(0|0)$, system parameters, and observations.
Output: Updated estimates $\hat{Y}(n|n)$, $R(n|n)$

1: **for** each time step n **do**
2: **Prediction:**
3: Predict $\hat{Y}(n|n-1)$ based on previous estimate and system dynamics.
4: Predict the measurement: $\hat{X}(n|n-1)$ based on $\hat{Y}(n|n-1)$ and system dynamics.
5: Predict error covariance $R(n|n-1)$ based on the previous error and noise characteristics.
Update:
6: Compute Kalman gain $b(n)$ using the predicted error covariance and measurement noise.
7: Update the estimate $\hat{Y}(n|n)$ by incorporating the measurement $X(n)$.
8: Update the error covariance $R(n|n)$ after incorporating the measurement.
9: **end for**

Filtering the noised data from Figure 6.1, the aim is to determine how well the signal is filtered for noise. The values of σ_Z^2 and σ_W^2 were set to be 0.1 and 1 respectively as these values did not distort the signal. The resulting $\hat{Y}(n|n)$ is plotted on Figure 6.2.

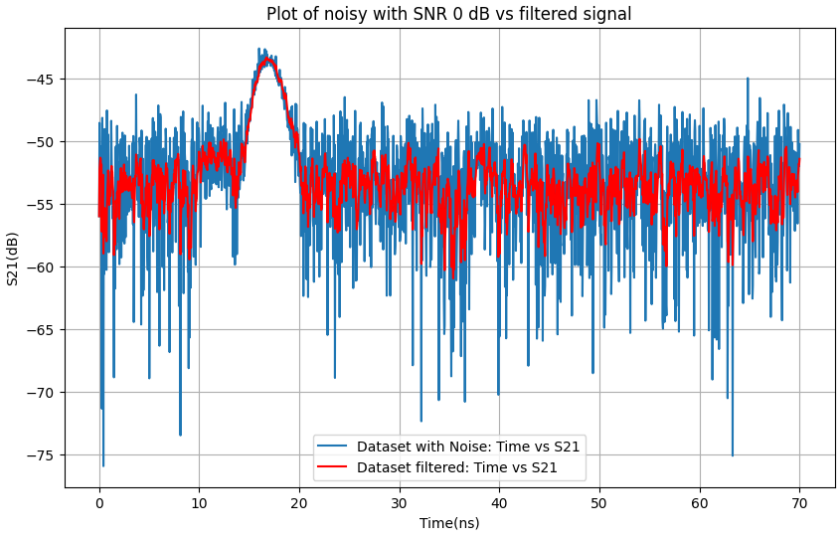


Figure 6.2: Noised signal from Figure 6.1 being filtered with a kalman filter with constant $h(n)$ and $a(n)$.

Looking at the resulting filtered graph, it appears that the noise is properly filtered reducing the variance in the data while avoiding a reduction in the strength of the peak signal. From this, the values of $h(n)$ and $a(n)$ are considered appropriate, given the filter's purpose of reducing noise without diminishing signal strength.

6.2.3 Normalization

Normalization has two different implementations. 1) L2 normalization from equation (3.7), which is calculated using the `numpy.linalg.norm()` function that by default calculates the L2-norm, $||x||_2$. 2) z-normalization from equation (3.1.2) using `numpy.mean()` and `numpy.var()` to calculate the mean and variance. To accurately represent the data, only one of the normalization techniques presented will be used, and only when FFT is not used. The original signal next to the normalization augmentee signal can be seen applied to data on Figures 6.3 and 6.4.

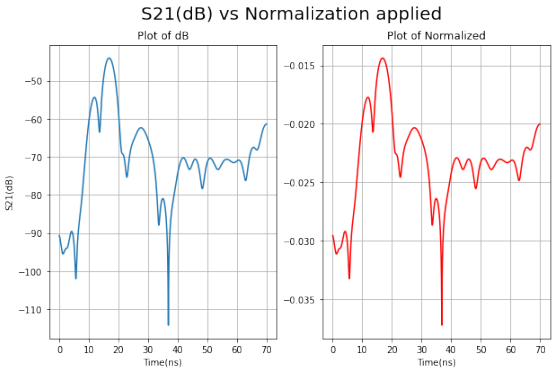


Figure 6.3: L2 normalization function applied to data.

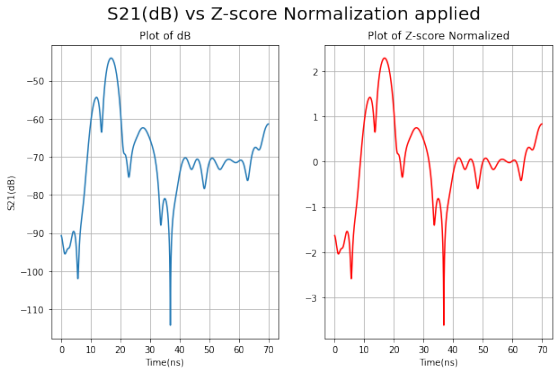


Figure 6.4: Z-Score normalization function applied to data.

6.2.4 Group Delay

To obtain the group delay of the phase, the `numpy.gradient()` function was used. The gradient function utilizes a taylor series approximation [47], as the phase values are constant and can

therefore not be differentiated with respect to angular frequency. Pseudo code that handles the calculation of group delay can be seen on Algorithm 3.

Algorithm 3 Group delay calculations

Input: Dictionary holding CSV values

Output: Dictionary holding initial CSV values with added group delay column.

```
for file in directory do
    Read dicitionary data
    Extract data from S21(DEG) column
    Convert phase from degrees to radians
    Unwrap phase in radians
    Calculate group delay using taylor series approximation
    Export dictionary with added column of group delay
end for
```

The calculated data for the full dataset of 1801 points is then assembled and added to achieve a graph similar to the one on figure 6.5.

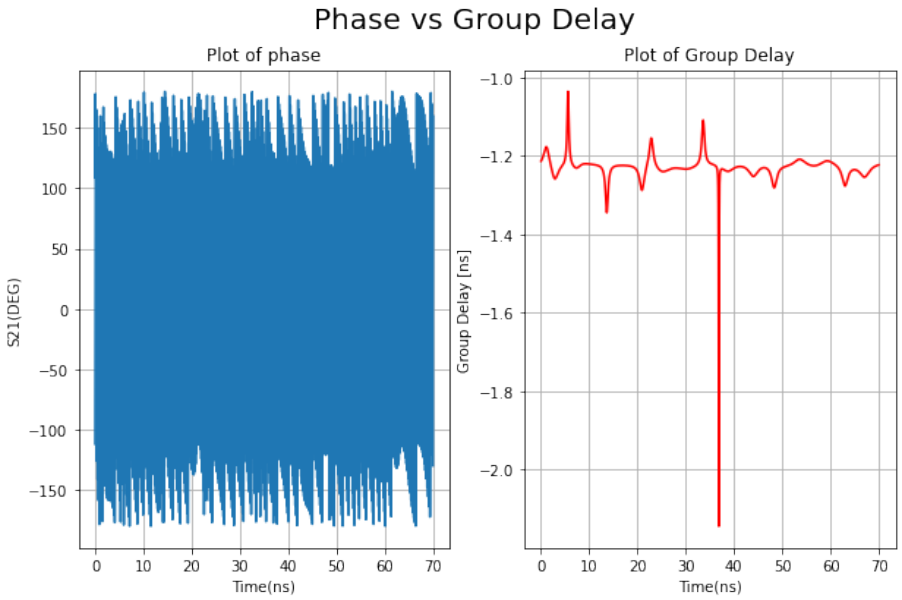


Figure 6.5: Group delay function applied to data.

6.2.5 Fast Fourier Transform

The function for the FFT is also implemented using the numpy library, which can perform a Fast Fourier Transform using the `numpy.FFT()` function. This function does not shift the data for accurate representation though, and the impulse will be placed on the first measurement, giving a skewed view for visualization. Therefore, the `numpy.fft.fftshift()` function was also used, which shifts the impulse to the center of the spectrum, hereby taking half of the samples and mirroring it, giving a more accurate representation of a Fourier Transform (FT).

To prevent loss of the frequency components of the signal used as input to the FFT, normalization is not used in combination. The FFT of an input signal can be seen on figure 6.6.

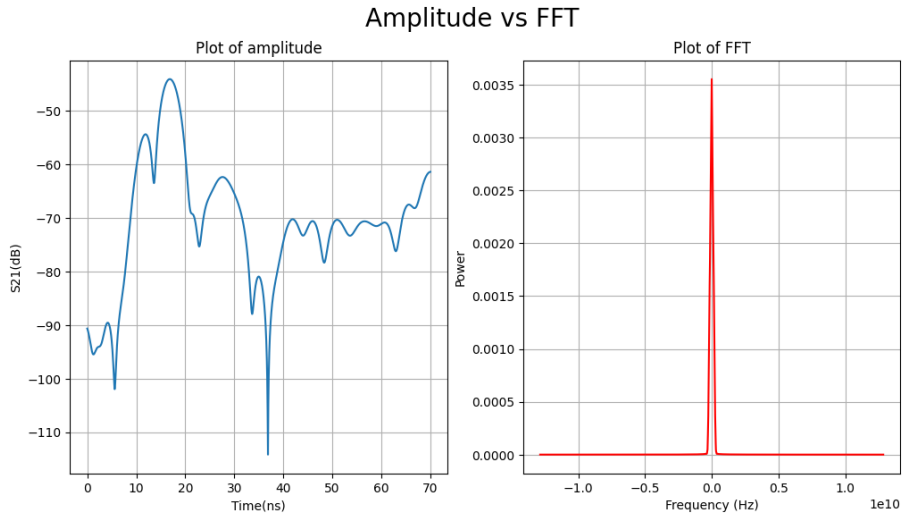


Figure 6.6: FFT function applied to noisy data with an SNR of 0 dB.

6.2.6 Kurtosis

To obtain the kurtosis, the `sklearn.stats.kurtosis()` function is utilized. The kurtosis function takes in one dataset at a time and calculates the kurtosis. The output of the kurtosis is appended to the dataset and is an excess kurtosis. The kurtosis is then found for all datasets in the dictionary and appended to each individual dataset.

6.2.7 RMS Delay Spread

To find the RMS Delay Spread of the signal, the algorithm 4 is used.

Algorithm 4 RMS Delay Spread calculations

Input: Signal and time vector

Output: Singular RMS Delay Spread value.

1: **if** Signal is non-linear **then**
2: Convert Signal to linear scale
3: **end if**
4: Find total power of Signal
5: Calculate mean delay using Signal, time vector and total power
6: Calculate RMS delay spread using mean delay, Signal, time vector and total power
7: Return RMS delay spread value

The RMS delay spread is then appended to each individual measurement in the dataset.

6.2.8 Peak power

To find the peak power of the magnitude, the PyTorch function `torch.max()` is used. This returns a the value of maximum power of the magnitude data, along with the index of the value. PyTorch utilizes argmax to find the index of the maximum value. The maximum value is then appended to the dataset.

6.3 Model implementation

The DNN models are implemented in PyTorch, a popular Python ML framework. PyTorch represents data in tensors, thus enabling PyTorch to easily handle GPU operations using tensors. Implementing the models are done over four steps. 1) Converting the dataset to tensors. 2) Splitting the dataset into training, testing and validation sets. 3) Define the model. 4) Training the model and plotting the validation and training loss for visualization of its performance. The DNN and SVM both operate through the four steps above, but the LSTM has an additional step before the first step, where the data is converted into a sequence. While the SVC implementation of SVM from the sklearn is used, it can still operate on tensors.

6.3.1 Converting to tensor

On a fundamental level, tensor operations are matrix operations, which is highly parallelizable and is therefore preferred to be moved to GPUs [48]. When copying the tensors from the host device to a GPU, the tensors are now in another memory space and therefore all tensor operations has to be performed on the GPU. See subsection 2.2.10 for more details on this.

Loading datasets is done using the `to_tensor()` function where data columns, such as those seen on Table 6.1, a dataset and the associated label is provided. The function then converts the values in the defined columns into a large tensor, and creates a matching label tensor. Another function was created being the `appendToTensor()` function. This takes a list of dataset dictionaries as inputs, and create a list of tensors with all the datasets, and a matching label tensor. Pseudocode for how data is loaded to a tensor can be seen in Algorithm 5.

Algorithm 5 Converting datasets to tensors using the `to_tensor()` function

Input: Dataset, label, data columns,
Output: 2D list of tensors

1: **for** file in dataset **do**
2: Check data columns to include
3: Create tensor with data columns and flatten to 1D
4: Create tensor with label and flatten to 1D
5: **end for**

The final tensor will have a dimension of `[csv_count, total_features]`, where `csv_count` is the sum of CSVs based on the included datasets from Table 6.2 and `total_features` is 3602 if no statistical features as mentioned in section 3.1 is added, otherwise up to 7207 with all features applied.

6.3.2 Sequencing the data for LSTM

As the LSTM is a Sequence predicting model that requires a 1-dimensional vector input, it is necessary to convert the tensor from the shape of `[csv_count, feature_length]` into a sequence of shape `[batch_size, sequence_length, 1]`, where `batch_size` will be discussed in subsection 6.3.3. As the dataset contains several dimensions of the data, such as S21(dB) and S21(DEG), it is necessary to take some considerations on this. The LSTM only want one-dimensional, and containing both dimensions in 1 sequence means mixing two different types of data, which is undesired. Thus, the dimension of the S21(DB) is determined to be the size of `feature_length`, as this is considered to be providing the most information about the signal.

6.3.3 Splits of datasets

To evaluate the ML models, all datasets chosen are combined and partitioned into a train, test and validation set. In case of using multiple datasets for training, these are combined before splitting. Splitting the datasets in these sets is done using `train_test_split()` function provided in sklearn with data shuffling set to true. Arbitrarily many datasets can be given as parameters to the function and tries to ensure all datasets are shuffled the same way, but can vary a lot. The process of generating the train, test and validation set can be seen on algorithm 6.

Algorithm 6 Data splitting

Input: Input data, Input label, test size, validation size, shuffle (default: True)
Output: Shuffled train, test and validation set

1: Load datasets
2: X_train, X_temp, y_train, y_temp = data_split(Input data, Input label, test_size, shuffle)
3: X_test, X_validation, y_test, y_validation = data_split(X_temp, y_tmp, validation_size, shuffle)

Finally, the datasets are loaded into a separate `DataLoader` that are iterable objects provided by PyTorch. With the `DataLoader` a batch size of 32 is set, as it is a good trade-off between performance and generalization. The final dimension given to the DNNs is `[batch_size, total_features]` i.e. `[32, total_features]`.

6.3.4 Defining the model

In this section, the architectures of the DNN, SVM and LSTM along with associated hyperparameters will be presented.

DNN Model

As the data is now ready to be used for training, testing and validation. The DNN is implemented using the architecture defined in subsection 5.2.1 using the `nn.Module` superclass provided by PyTorch. The amount of data columns, output classes and the dropout rate are specified when creating the model. The implemented model architecture can be seen on Figure 6.7.

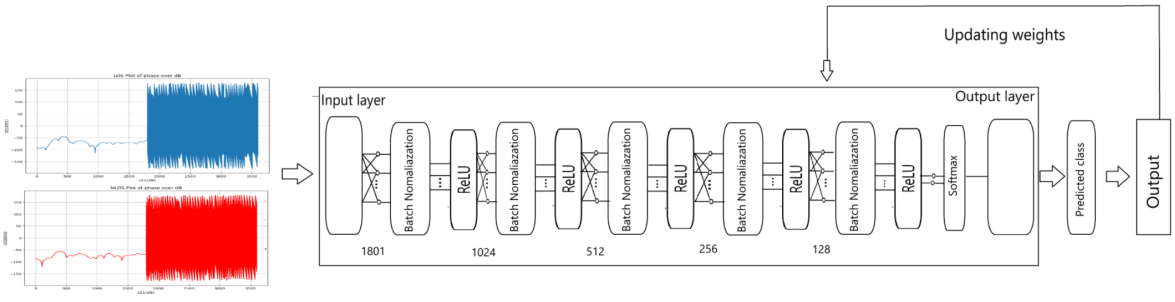


Figure 6.7: The DNN Model overview.

After defining the initial parameters, the model is defined to be sequential using the `nn.Sequential` class. The activation function used for each layer is the ReLU function, and each layer also has a batch normalization of the 1D tensor and dropout function connected using the `nn.BatchNorm1d` class.

The batchNorm1d function normalizes the data via the batches used for training. During the training, it updates the mean and standard-deviation using the new data provided.

SVM Model

When defining the SVM model, the kernel type, degree of the polynomial kernel (if polynomial), and the regularization parameter C are determined. In this project, polynomial kernel and RBF are chosen. Before defining the model, the data is standardized using `StandardScalar()` that performs z-score normalization. Then, the model is defined using the `SVC()` class provided by sklearn, where parameters such as kernel, C, and degree are set. The probability parameter is set as true, enabling the model to output probabilities for each class instead of only the class label. To train the model, the `SVC().fit()` method is used. After training, the `SVC.predict()` method will classify the data in the test set. Finally `predict_proba()` gives the predicted probabilities of each class.

LSTM Model

To define the LSTM model, nn.Module is used again, similar to how the DNN model is defined. The number of input features, output classes, and dropout rate are specified. The model is defined as a bidirectional LSTM that considers past and future values in a sequence. During the forward pass, input sequences are first processed through LSTM and produce output. These outputs are then normalized using torch.nn.LayerNorm to improve stability by standardizing each time step's hidden states. Following this, attention scores are calculated via a linear transformation function, which is nn.Linear, and the Softmax function is applied to obtain the attention weights, finding the key value in the samples. Then, the attention weight is used to calculate the weighted sum of the LSTM output, producing context vector aggregates, which is the most relevant information from the sequence. Finally, the context vector will be passed through the classifier to get the final output.

6.3.5 Training

Now that the models have been defined, training can almost begin. First, the training parameters must be defined, which are the loss functions, the optimizer and the optional scheduler. The optimizer is set to ADAM. Two loss functions have been implemented, being cross entropy loss and focal loss.

PyTorch has a built-in implementation of cross entropy, while focal loss is implemented according to section A.1. The alpha and gamma values are first defined, and the focal loss is then calculated using these values together with the cross entropy. The pseudocode training process of the DNN and LSTM can be seen in algorithm 7. The training loop keeps track of the training and validation loss used for model evaluation later on.

Algorithm 7 Simplified Model Training and Validation for DNN and LSTM

Input: Model, Data Loaders, Loss Function, Optimizer, Scheduler (optional), Number of Epochs (`EPOCHS`).
Output: Trained Model, Loss History.

for each epoch $e \in \{1, \ldots, \texttt{EPOCHS}\}$ **do**
    Training Phase:
    Set model to training mode.
    for each training batch **do**
        Compute predictions and training loss.
        Backpropagate and update model parameters using optimizer.
    end for
    Record average training loss.
    Validation Phase:
    Set model to evaluation mode.
    for each validation batch **do**
        Compute predictions and validation loss.
        Update validation accuracy.
    end for
    Record average validation loss.
    Record validation accuracy.
    Check if validation accuracy is the best so far:
    if current validation accuracy is the best **then**
        Save model as the best model.
    end if
end for
Return best model as trained model and loss history.

The final model is the model with the best validation accuracy and is saved as **best_model.pt**.

The SVM training loop is simpler, as the SVC implementation has a method called `SVC().fit()`. The pseudo code for the SVM training can be seen on algorithm 8.

Algorithm 8 Simplified Training loop for the SVM.

Input: Datasets, kernel, degree (Only for poly kernel)
Output: Trained Model.

X_train, X_test, y_train, y_test = Load datasets
Initialize SVM with kernel parameter and degree
SVM.fit(X_train, y_train)
Return trained model

6.3.6 K-Fold cross Validation

To give models a more fair comparison, K-fold cross validation has been used, where $K = 8$ using the `KFold()` class provided by sklearn. Like in the training loop specified in algorithm 7, the training loss, validation loss and validation accuracy is stored along with the best model between all folds.

Algorithm 9 Simplified training and validation loop for K-fold cross validation.

Input: Datasets, Model, K, shuffle (Default: True)

Output: Best model in K folds, loss history, accuracy history.

```
Load datasets
best_accuracy = 0
for fold in {1,.., K} do
    for epoch in {1,.., EPOCH} do
        Train model fold
        Validate fold
        if model_accuracy > best_accuracy then
            best_accuracy = model_accuracy
            Save model weights
        end if
    end for
end for
```

The KFold training loop is the exact same as in algorithm 7, but where this process is repeated for every fold. A simplified version of this can be seen in algorithm 9 and a plot of this can be seen on Figure 6.8.

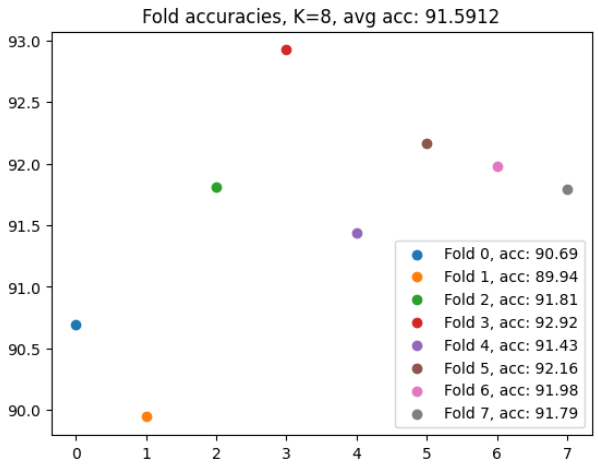


Figure 6.8: Example of K-fold cross validation with 8 splits. This figure shows the accuracies for the folds for data containing the Original data and HF data for a two-class classification problem with an SNR of 0 dB.

6.4 Test implementations

With the models trained, testing of their performance can begin. This is done by using the test data that was previously withheld from the training set. It is then used for all tests described in section 5.5, so it will all be unseen data for the model, and the tests can be comparable to one another in some degree. There will be performed three tests to see the performance of the model. The first test is an accuracy test, which is represented using a confusion matrix to visualize the results. The second test will be the SNR graph, which shows the models performance on data with varying SNR. Lastly, the model will be tested on data gathered from other locations, to see if it is general enough to be applicable in multiple scenarios.

6.4.1 Confusion Matrix

The first test is the confusion matrix, which is implemented similarly to the validation of the training part of the model. First, the trained model is imported and set to evaluation mode. Three lists are made to save the results, being all the predictions, the ones that are classified incorrectly and the correct answers to each prediction.

The confusion matrix is made using the sklearn library, using the saved predictions and correct results as inputs. This is then used in the plot_confusion_matrix function which uses seaborn and matplotlib to plot the confusion matrix for visualization. The accuracy, loss, correct predictions and wrong predictions are printed together with the confusion matrix as a part of the testing loop from algorithm 10.

Algorithm 10 Simplified testing loop.

Input: Trained model, Testing set
Output: Best model in K folds, loss history, accuracy history.

```
Load trained model
Set model to evaluation model
test_loss = 0, correct predictions = 0
for file in test set do
    Model predict
    Update correct predictions
end for
Generate confusion matrix
```

6.4.2 SNR Loop

After implementing the test to see the accuracy of the models optimal performance, it is time to see the models resilience when noise added to the test dataset. The first step is to ensure that the test data is noiseless. This is done by taking the noiseless datasets, and taking the train/test/val split with the same seed, ensuring the data is the same. The data is transformed back to dictionaries with dataframes so noise can be added to the data once again. The pseudo code for the SNR loop function called `SNRLoop()` can be seen on algorithm 11.

Algorithm 11 Simplified SNR loop.

Input: Unedited testing set, db_min, db_max
Output: SNR accuracy graph as seen on Figure 6.9.

```
for SNR in {db_min, db_min + 5, ..., db_max } do
    for dataset in Original, HF, BC do
        Add noise to dataset
        if data augmentation = true then
            Perform data augmentation (filtering, fft and/or normalization)
        end if
    end for
    Create dataloader
    Test model on augmented datasets
end for
Return: Plot of model accuracy as a function of noise SNR
```

This process running mentioned in subsection 5.5.3, sets a SNR step of 5. This is plotted, using matplotlib, with the SNR as the x-axis and the returned accuracies on the y-axis. This can be seen on Figure 6.9.

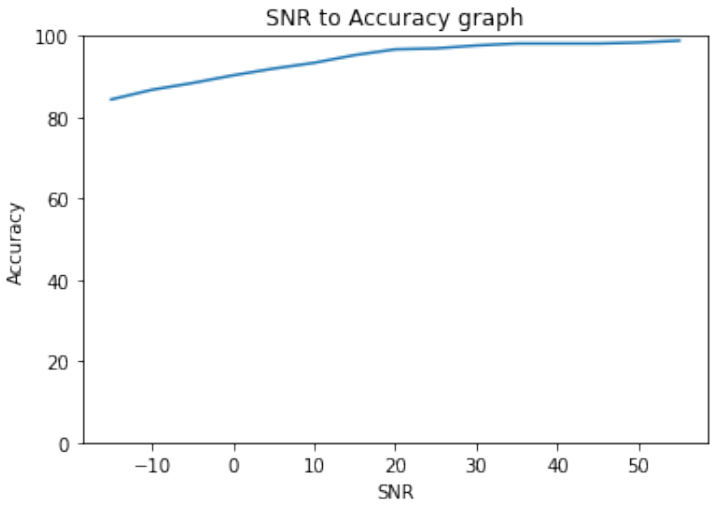


Figure 6.9: Example of SNR function output graph.

6.4.3 Generalization test

The final test is a generalization test. It is used to see if the trained model can work in different environments, and its resilience to change of locations. The way this test is implemented is by taking a different dataset than trained on, and then repeating the Confusion Matrix and SNR Loop test on the chosen dataset.

The accuracy of each scenario given in the confusion matrix test will be plotted side by side for easy comparison between each model. For the SNR loop test, the accuracy of all datasets are plotted together. The confusion matrix and SNR loop graph will also be saved as two separate figures. Examples can be seen of the confusion matrix in Figure 6.10 and of the SNR loop test in Figure 6.11.

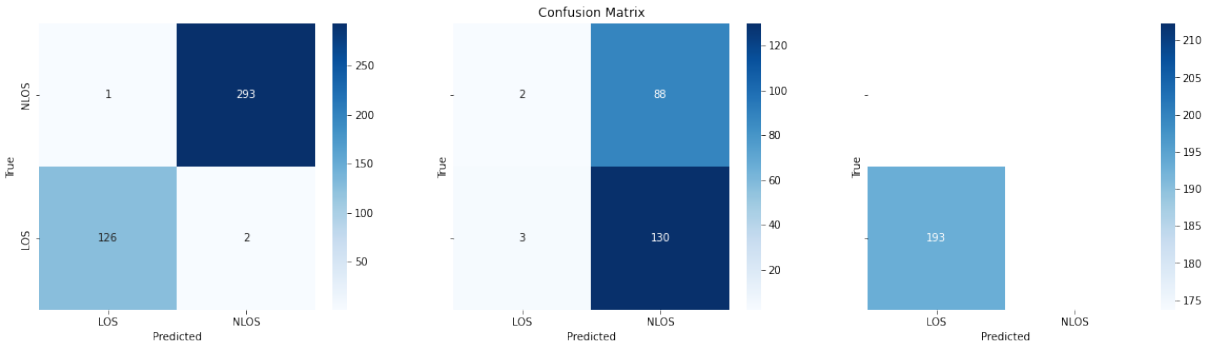


Figure 6.10: Example output of the general Confusion Matrix function.

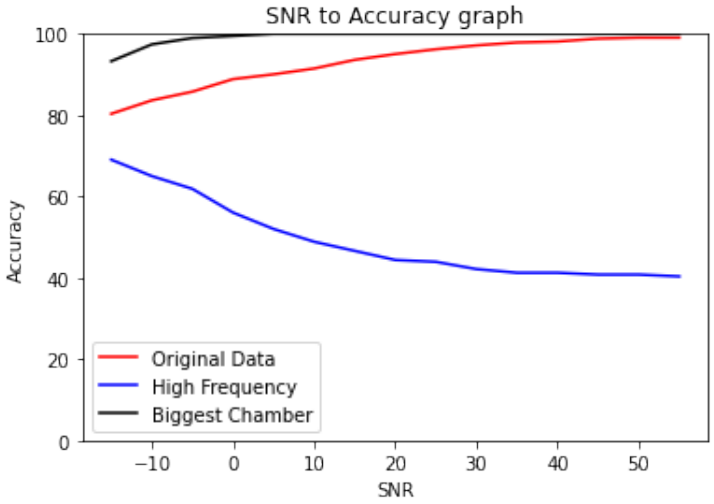


Figure 6.11: Example output of the general SNR loop function.

7. Integration Test

After implementing the preprocessing methods, each method mention in section 5.1 has been used when training a DNN model, as well as all valid combinations of used preprocessing methods. Throughout this section, the impact of each preprocessing function will be evaluated from the results of the tests made from section 5.5. After seeing the result of each preprocessing method, the five best DNN models are chosen, based on their performance in accuracy, SNR accuracy as well as their performance in the general tests. They are then compared to the SVM and LSTM models. All test results can be found in Appendix C.1.

7.1 Impact of preprocessing

To see the impact of the individual preprocessing methods, the model was first trained without any methods to determine the baseline for evaluating the model. As there is usually a degree of randomness when training models, it is hard to directly compare the preprocessing methods. A solution to this, is by setting the seed of the randomness, making sure that each "random" action is the same as the previous. After doing this, the only difference between each model should be the applied preprocessing.

7.1.1 Preprocessing summary

A more detailed analysis of the results presented in this section can be seen in the appendix D. A summary of the results can be seen in Table 7.1 and 7.2, which shows each methods accuracy on its own test set, the average SNR step accuracy, as well as its performance on other datasets.

Modification	Acc.	Avg. SNR acc.	HF data acc.	BC data acc.
Baseline model				
None	91.00%	77.89%	47.09%	100%
Model with modification				
SNR 0	91.00%	87.63%	53.36%	100%
SNR random	90.76%	91.00%	49.78%	100%
Kalman Filter	94.31%	82.88%	46.64%	100%
Group Delay	94.31%	83.42%	44.84%	100%
FFT	94.55%	83.59%	45.74%	100%
Normalization	91.94%	91.94%	47.98%	89.64%
Z-score Norm	92.89%	90.33%	45.29%	93.78%
Kurtosis	95.26%	83.47%	47.09%	100%
RMS	94.55%	80.24%	42.60%	100%
Peak Amplitude	92.65%	85.37%	45.79%	100%

Table 7.1: Table over preprocessing accuracies with a DNN trained on the OG dataset.

As can be seen in table 7.1, when comparing to the baseline DNN trained on only OG data, no methods can be seen that improve the generality of the model by a significant amount. When the model is trained on a specific dataset, it seems to classify all measurements from other datasets are of NLOS, given that the accuracy seems equivalent to the ratio of NLOS to LOS measurements. The BC data consists of pure NLOS before being made into a test set. This could be a reason for the large accuracy in generalization test, as it assumes everything but data originating from the training dataset is NLOS. The HF dataset used for the test consists of 50/50 LOS and NLOS data, as not all NLOS files with alternating elevation of the RIS were discovered when testing.

For the accuracy column in Table 7.1, it can be seen that many of the modifications increase the accuracy, with kurtosis giving the highest increase in performance of 4.26% compared to the baseline.

For the SNR accuracy in Table 7.1, it can be seen that SNR 0, SNR random, Normalization and Z-score normalization were quite effective in increasing the accuracy compared to the baseline model. When looking at the SNR graphs results of the tests on Figure 7.1-7.4, it can be observed that the SNR random function makes the DNN robust for all tested SNRs while the SNR 0 function performs best at SNR 0, which is to be expected. Both normalization methods seems to provide the same benefit, although the L2-normalization function seems very irregular with having the same accuracy at all SNR steps. This is not reasonable, and is likely a slight issue with the L2-normalization implementation.

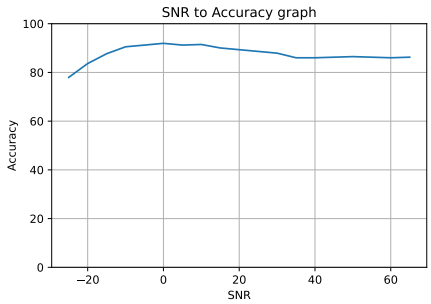


Figure 7.1: Model accuracy of different SNR, when trained on SNR 0 dB

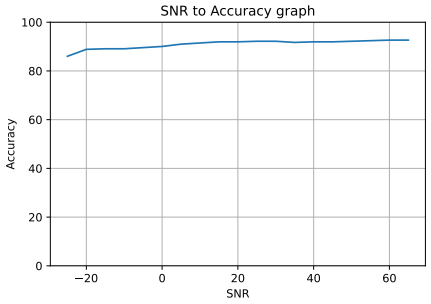


Figure 7.2: Model accuracy of different SNR, when trained on random SNR between -15 and 60 dB

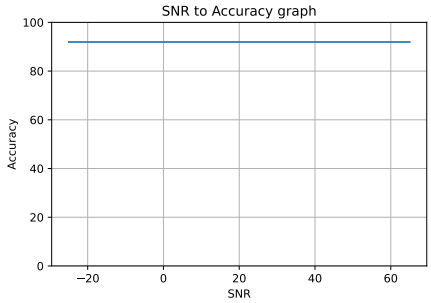


Figure 7.3: Model accuracy of different SNR, when trained with L2-normalized data

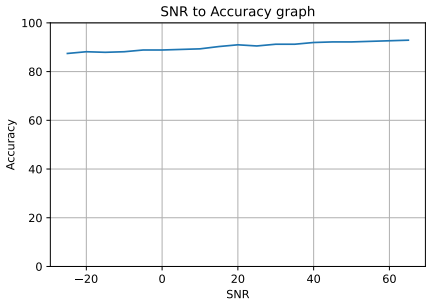


Figure 7.4: Model accuracy of different SNR, when trained with Z-Score normalized data

Train dataset	Acc.	Avg. SNR acc.	OG data acc.	HF data acc.	BC data acc.
HF	87.00%	64.79%	38.39%	N/A	100%
OG+HF	93.49%	75.87%	99.05%	94.17%	99.48%

Table 7.2: Table over preprocessing for DNN transfer learning accuracies with a DNN trained with no data augmentation.

Although the preprocessing methods did not seem to increase the accuracy of the generalization test compared to the baseline, improvements were seen when using transfer learning. The results can be seen in Table 7.2. When testing on different datasets, it could be seen that the HF data provided poor results overall, with lower accuracy for all tests. The combination of both the OG and HF dataset saw improvements to the accuracy in all tests. This shows that combining the datasets improve the generality of the model, and increases its use-case to multiple locations.

7.2 Model tests

To find the best combination of preprocessing methods used in section 7.1, many combinations of preprocessing was tested. The different combinations were limited to a certain extent due to the many plausible combination methods. This is performed for the datasets; Original dataset, High frequency lab and a combination of the two for all the tests described in section 6.4. This is performed on the DNN and to an extent on the LSTM and SVM.

7.2.1 DNN

The combinations were limited to 52 tests with the first ten described in Table 7.1. After certain test combinations were tested, the normalization method was determined to be Z-normalization (see Appendix subsection D.6.1). The noise addition method was likewise determined to be random noise after comparison-tests (see Appendix subsection D.2.2) and filter is applied on all noise (see Appendix section D.3). Table 7.3 presents the results for the top 5 performing combinations of preprocessing methods.

		CM Accuracy			SNR Loop average		
Num	**Modification**	**OG**	**HF**	**OG + HF**	**OG**	**HF**	**OG+HF**
31	SNR + Random + Filter + FFT + RMS	91.00	83.86	87.90	91.00	83.69	85.60
12	SNR + Random + Filter + GroupDelay	92.42	81.61	88.22	92.19	80.46	87.56
46	SNR + Random + Filter + GroupDelay + Kurtosis + RMS	89.57	83.86	88.37	90.62	81.35	87.49
2	SNR Random	90.76	82.96	87.44	91.00	81.35	86.63
26	SNR + Random + Filter + GroupDelay + FFT	91.47	81.17	87.75	91.33	80.98	86.40

Table 7.3: Best 5 combinations of the CM and SNR loop Average tests. These are tested on the test split to determine the best model combination The SNR loop average test consists of accuracies for the SNR loop function on Figure D.5 being averaged across the range of [-20;60] dB SNR.

The results in Table 7.3 consists of the highest average accuracy across the 6 different accuracies for each modification setup, see Appendix E for the full tables. The column "CM Accuracy" refers to the accuracy of the model when trained on a specific dataset and the resulting accuracies are from the test split data. The SNR Loop average is the average accuracy across the interval [-25;65] dB SNR on the test split. The column "CM OG+HF" refers to the accuracy of the confusion matrix when the model is trained on a combination of the OG dataset and the HF dataset. Here it is noticeable that HF tends to lack in accuracy compared to the others, but when the data is combined, the performance improves.

The highest average accuracy across the samples is for test number 31, but given that everything but the HF are higher for number 12, it will be determined to be the best model and it will be compared to the rest of the models. Number 12 consists of randomly added noise that is filtered with a kalman filter and the group delay is added as an extra dimension. This model achieves accuracies between [80.46%;92,42%] for the tests, while having the best accuracy on the "OG+HF" among the five models. Number 12 is likewise the model with the highest accuracy for the generalization test with the lowest being 91.93%. Figures of the tests for test 12 are available in Appendix E.2 and the generalization test in Appendix Table E.2.

To ascertain the accuracy is representable across different data compositions, k-fold is performed. The data is split into $K = 8$ different splits to determine the accuracy overall for the model and the accuracy over the different splits. It can be seen on Figure 7.5 that the model achieves an average accuracy of 90.38%.

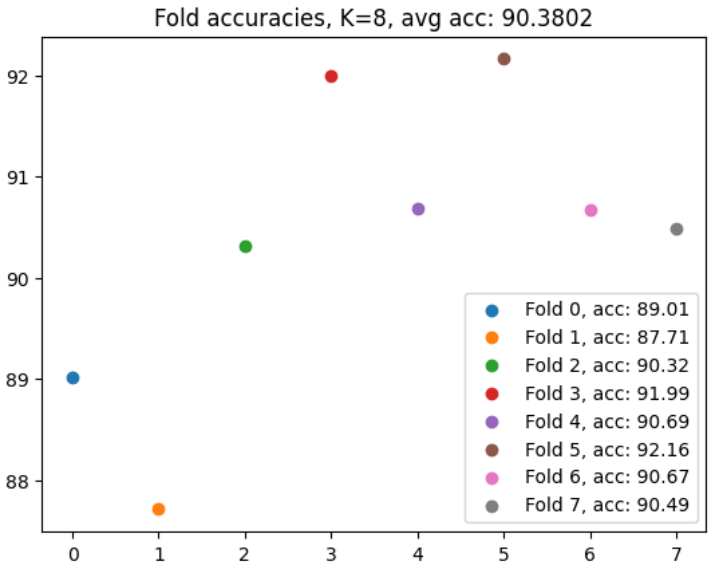


Figure 7.5: K-fold training using the data and modifications for testcase No. 12.

7.2.2 SVM

The SVM was previously determined to be run with the preprocessing methods, RMS Delay spread and kurtosis, and these are considered a constant in this case. The other combinations which will be tested is the test of randomized noise versus no noise. It is tested on two kernels, the RBF and the polynomial kernel of order 3. An overview of the results are available on Table 7.4.

		CM Accuracy		
Kernel	**Modification**	OG	HF	OG+HF
Poly	No Noise	90.39	90.21	96.04
Poly	Random Noise	90.57	88.97	97.32
RBF	No Noise	94.84	93.95	96.86
RBF	Random Noise	93.59	94.48	96.74

Table 7.4: CM accuracy results for the SVM when utilizing different kernels and noise augmentations.

The results in the table consists of the different combinations of noise and no noise tested on the different datasets and combined for two different kernels. The model is focused on the results for the "OG+HF" test results as these should allow the model for better generalization given the different environments. For this reason, the kernel RBF is determined given its consistently high accuracies. The noised kernel is chosen as the standard with its accuracy of 96.74% due to potential improvement towards new noisy data. The CMs to the corresponding accuracies can be found in Appendix E.3. A visualization of the SVM decision bound can be seen on Figure 7.6, where the data's dimensionality is reduced down to 2 dimensions for the sake of visualization and is therefore not necessarily representative.

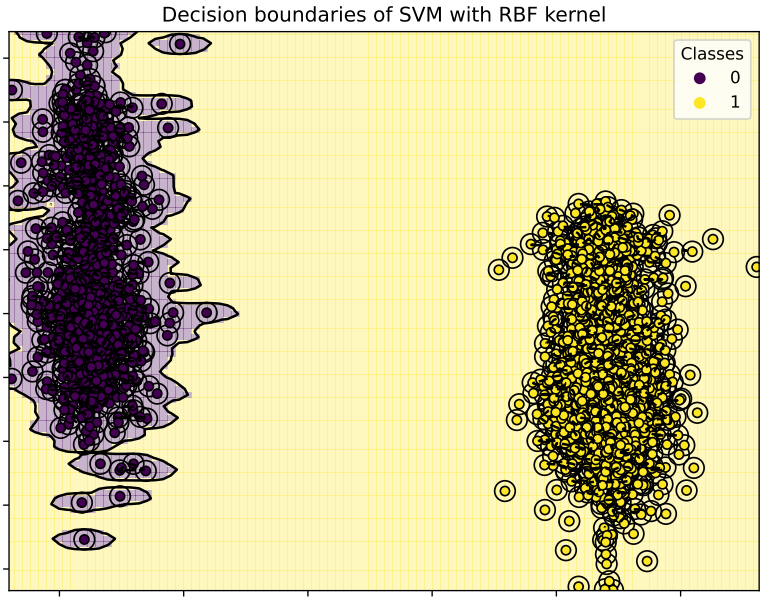


Figure 7.6: Visualization of SVM decision bound. Dimensionality reduced down to 2 dimensions.

To ascertain that the model does this across different data combinations between test and training, a K-fold cross validation is performed with $K = 8$.

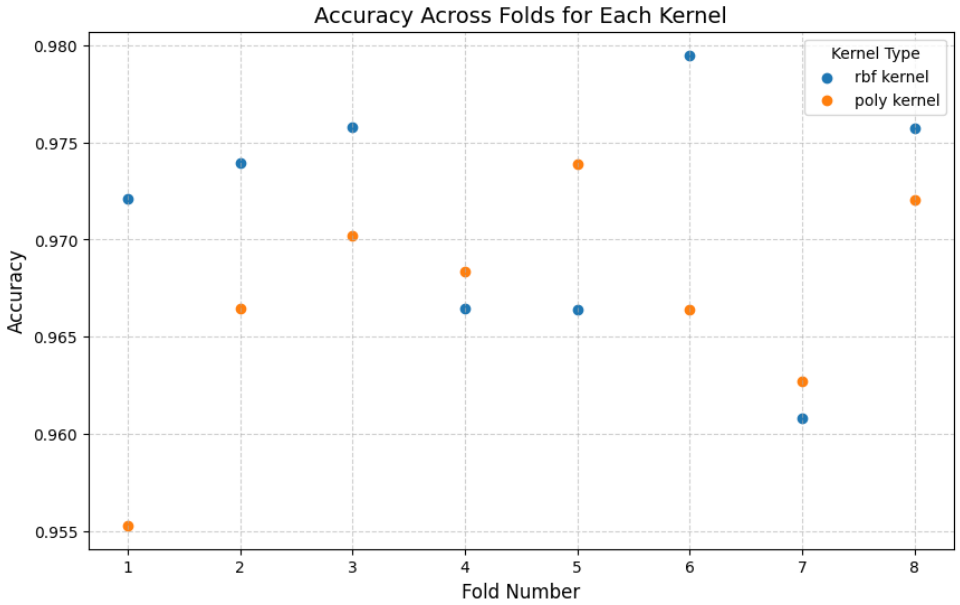


Figure 7.7: K-fold training using data on an SVM for the different kernels.

The plotted accuracies in figure 7.7 indicate that the RBF kernel performs the best across the different folds, demonstrating consistent superiority with an average accuracy of 97.14%.

7.2.3 LSTM

The LSTM is limited to attempting similar performance as the SVM. However, given the LSTM's requirement for a sequence as an input it limits the data to one column. Therefore the S21(DB) is determined as the data that is used for LSTM also given the importance of the S21(DB) given the results in appendix F.1. As this is the case, the data augmentation possibilities are limited to the S21(DB). One example using "OG+HF" is plotted on Figure 7.8.

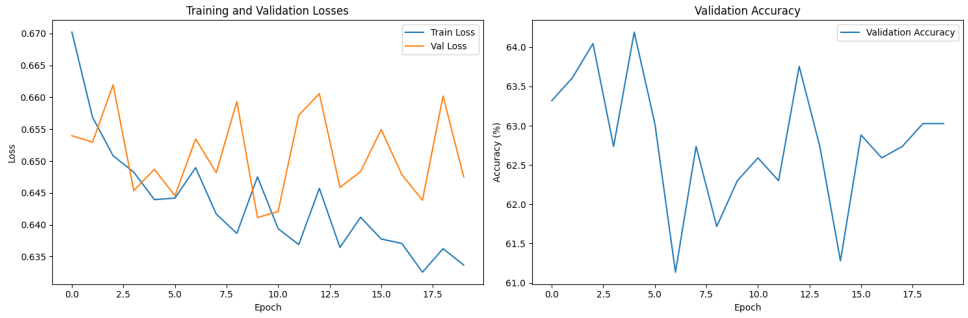


Figure 7.8: LSTM model training, showing an accuracy around 63%. It was difficult to improve the model consistently, with lower results than other models.

Given the results on Figure 7.8, it appears that the LSTM is unsuitable for achieving a high accuracy for the datasets. This appeared across the following iterations with a certain difficulty in recreating results with the same setup. The model is therefore considered unsuitable for the purpose.

7.3 Overall evaluation of the models

Three models have been made with the purpose of testing and comparing them to determine which of the models achieve the best overall performance. The LSTM model was determined to be unable to compete with the DNN and SVM models, due to the accuracy never achieving higher than 80%. The two competing models are therefore the SVM with the RBF kernel and the DNN number 12 from Table 7.3.

7.3.1 Confusion matrix

As the problem is a classification problem, the distribution of the classification is necessary to compare the SVM and the DNN. Given the kernel and model specified in previous sections, the confusion matrices can be seen in Figure 7.9.

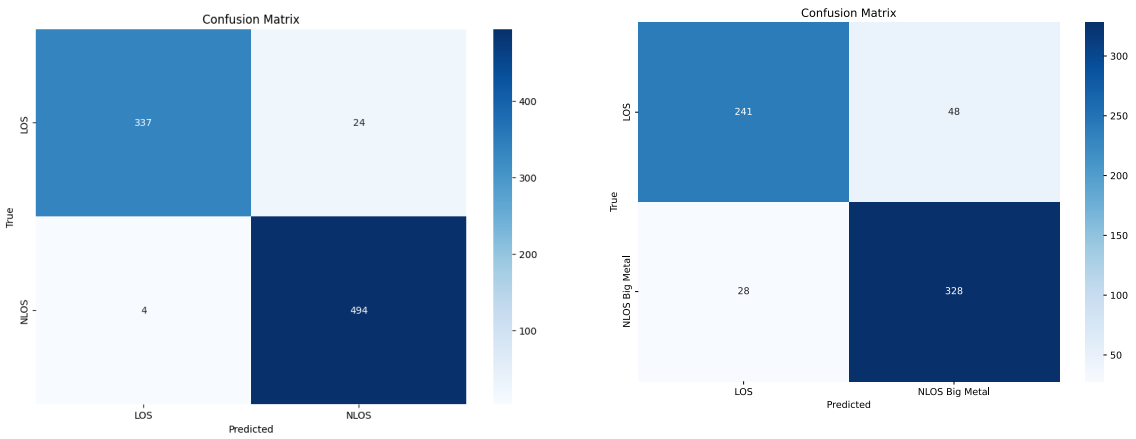


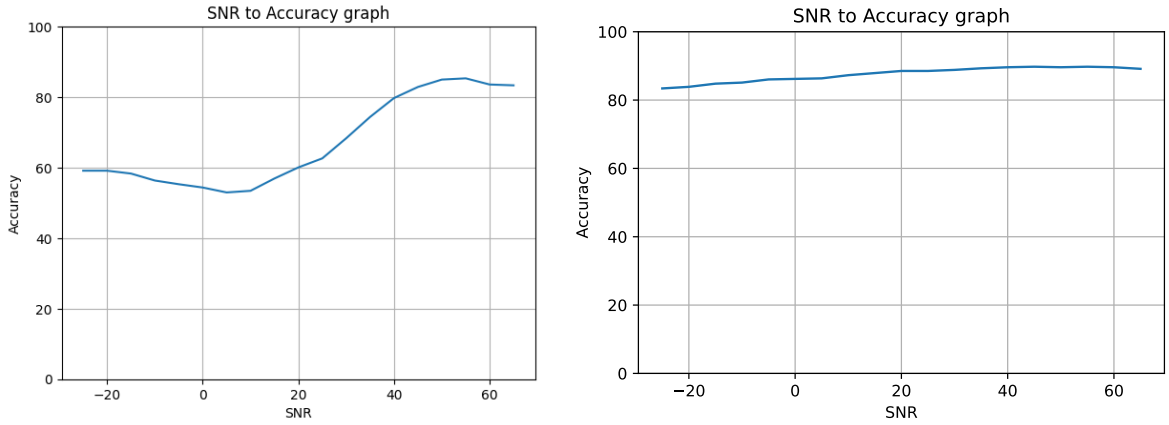
Figure 7.9: Confusion matrix of the test for the SVM and the DNN model. Left model is the RBF kernel of the SVM model. Right is composition number 12 of the DNN.

The left figure in 7.9 shows the classification distribution for the RBF kernel when random noise is added on the combined OG+HF dataset. This achieved an accuracy of 96.74%, seen in Table 7.4.

The right figure in 7.9 shows the distribution for the DNN number 12 test situation. It involves random noise filtered with a kalman filter and added group delay data. The test achieved a similar distribution of data between the SVM and the DNN, except with a lower accuracy of 87.53% meaning that the SVM is better in this situation.

7.3.2 SNR loop

Another test parameter that is for the models is determining how well the model performs at different SNR values. This is important as different environments have different SNRs.



(a) SNR loop of SVM.

(b) SNR loop of DNN.

Figure 7.10: SNR loop of respectively SVM and DNN. These SNR loop graphs are made on the test-split. It shows that the SVM has a worse resistance to noise added at different levels.

From the comparison in Figure 7.10 between the SVM and the DNN for the SNR graph, the SVM appears to be more sensitive to varying SNR levels, while the DNN is more robust across

different SNR values. Therefore for different amounts of noise, the DNN appears to perform the best.

7.3.3 Generalization

A last test for the model is to ascertain the generality. This test has only been performed on the DNN due to time constraints. Here, the model is tested on different datasets with the results seen on Figure 7.11.

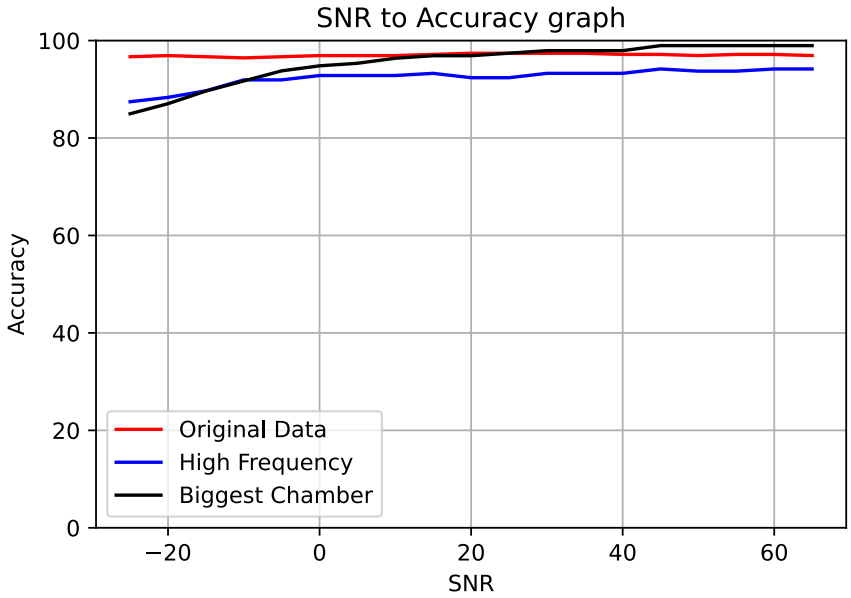


Figure 7.11: Generalization test of the final DNN model trained on a OG+HF dataset tested on all the individual datasets.

The DNN model is trained on the full OG+HF dataset and is tested on the test splits of the full OG, HF and BC datasets. As seen in figure 7.11 at lower SNR, the accuracy appears to fall off which is as expected. As the SNR increases, so does the performance up to a accuracy above 90%. The model appears to perform really well on the BC dataset, reaching close to 100%. This dataset was unseen, meaning the model achieves a fairly accurate result. From the results on the figure, it can be seen that the model achieves high general performance across different environments.

7.3.4 Conclusion

As seen in the the tests throughout this section, the SVM acquired higher accuracies when tested on the test split of the data it was trained on, making it more efficient for specific use cases. However, as can be seen in the SNR loop test, the SVM is not able to keep this accuracy in different noise environments, thus performing terribly with low SNR levels. It is here that the DNN model achieves a way higher accuracy at all noise levels, making it more efficient in real use-cases, as it is more resilient to changes in the environment. In summary, the SVM beats the DNN with up to around 8% in specific use case accuracy, but loses out a lot more in different SNRs, where the DNN has a consistently high accuracy. For this reason, the DNN model is chosen as the better performing model.

8. Localization

As stated in section 5, this chapter will explore research question 3 stated in chapter 4, and will aim to develop and test algorithms for estimating the distance between the antennas. The following section will therefore explore possible solutions for localizing objects in NLOS, hereby calculating the direct distance to set object.

8.1 Definition of localization

In this project, the definition of localization is detecting the direct distance between a transmitting antenna and a receiving antenna. This is considered for two scenarios, which are NLOS and LOS. In the situation of LOS, the transmitter and receiver have a direct path through the use of a Reconfigurable Intelligent Surface (RIS), without any obstacle. This is due to the RIS reflecting the signal directly to the Rx. In the situation of NLOS, there is an obstacle between transmitting antenna and receiving antenna, so the RIS cannot reflect the signal directly.

8.2 Known parameters

The transmitted signal in the experiment described in section 3 is a single impulse. For this reason, it is assumed that the sampling on the receiver was started at the exact same time as the transmitter starts to transmit. This makes it possible to estimate a Time of Flight (TOF), meaning that the information available is:

- Magnitude of the received signal
- Phase of the received signal
- (Estimated) Time of flight
- RIS angle
- Receivers direction
- Distances as described in Figure 3.1
- Distance between Tx and RIS

With these things known, it is possible to design algorithms which with some degree of accuracy be able to estimate the direct distance between the transmitter and receiver.

8.3 Algorithms for localization

The localization algorithms that are proposed are utilizing trigonometry on the path of the signal, or the group delay in combination with a moving average system. Trigonometry needs two parameters to calculate a distance. First, the closest wall side to the Tx in the room in which it is implemented. Second, the estimation of the time of flight, which are needed before calculations can be made. The group delay only needs the phase to estimate a distance.

8.3.1 Trigonometry

To obtain the direct distance between the Tx and the Rx, the transmitted signal will be thought of as being a direct, clean beam of energy without any form of scattering, and which only gets reflected once before arriving at the Rx. Furthermore, it is assumed that the distance from the transmitter to the closest sidewall is known. By using this method, it is possible to follow the transmitted signal to the Rx, hereby making it possible to utilize trigonometry to estimate the direct distance to the Rx from the Tx. In figure 8.1, the setup described in section 3 has been drawn with the trajectory of the signal reflecting once before arriving at the Rx. Tx will in the following section refer to the RIS's position, and not the actual position of Tx.

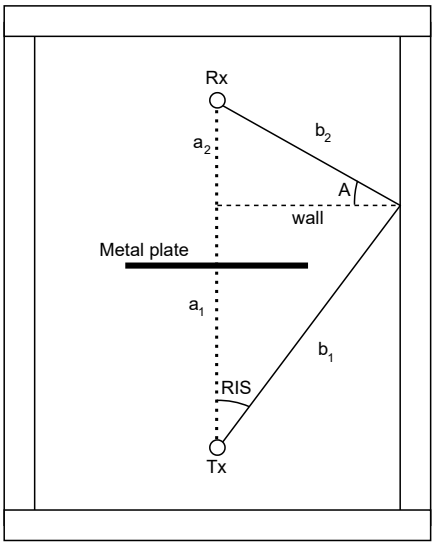


Figure 8.1: The test setup with a transmitted signal which experiences no scattering, and with only one reflection before arriving at the receiver. The signal is transmitted with an angle from Tx.

For this scenario, it is possible to calculate the direct distance by utilizing cosine relation. The path of the signal creates two 90 degrees triangles, as seen in figure 8.1. This can lead to issues, so for the necessary calculations to be valid, the following must be true:

$$|b_1| > wall \tag{8.1}$$

$$|b_2| > wall \tag{8.2}$$

$$-1 < \frac{wall}{b_2} < 1 \tag{8.3}$$

If these are not true, the triangles cannot exist, and thereby an invalid value will be returned. Requirement equation 8.3 is used to estimate the angle A seen on Figure 8.1. This angle is derived as

$$A = \cos^{-1}\left(\frac{wall}{b_2}\right) \tag{8.4}$$

which is used to estimate the a_2 length by

$$a_2 = \sin(A) \cdot b_2 \tag{8.5}$$

To estimate the distance traveled by the transmitted signal, the time of flight is used. This is found by utilizing the time arrival of the maximum received magnitude, and then subtracting it with the time the signal was transmitted. Time of flight is then multiplied by the speed of light, which is done due to the assumptions made about the signals propagation. The signals traveled path distance is then defined as

$$D = \left(TOF - \frac{d}{c}\right) \cdot c = b_1 + b_2 \tag{8.6}$$

where

D	Distance of path
TOF	Time of flight
d	Distance between transmitter and RIS
c	Speed of light
b_1	Distance from Tx to wall following the propagation path
b_2	Distance from wall to Rx

The transmitting antenna in the setup described in section 3 is closer to the Rx than the distance of the signal path. This means that the distance calculated is not the correct distance between the transmitting antenna and Rx. This can be accommodated by subtracting $\frac{d}{c}$ from TOF. This makes Equation (8.6) able to find the path distance from the RIS to the Rx. It will not make a difference in the comparison, as it is an offset. It is possible to calculate the distance of b_1, if it is assumed that b_1 is an unknown side in a 90 degree triangle. The formulation is defined as:

$$b_1 = \frac{\text{wall}}{\sin(\text{RIS})} \tag{8.7}$$

Here, RIS represents the angle at which the Tx is transmitting. By knowing the length of b_1, it is possible to calculate part of the direct distance to the Rx point from Tx, as follows:

$$a_1 = \cos(\text{RIS}) \cdot b_1 \tag{8.8}$$

With b_1 calculated, it is possible to calculate the corresponding side b_2, which can be solved as:

$$b_2 = D - b_1 \tag{8.9}$$

However, this will create a problem, as this will not guarantee that the requirements stated in equations 8.2 and 8.3 are upheld. Therefore, instead of utilizing two 90 degree triangles as described in figure 8.1, one 90 degree triangle and one arbitrary triangle will be used. The

height of the arbitrary triangle will be the remaining distance to the receiver. The new setup for calculating the direct distance can be seen in figure 8.2.

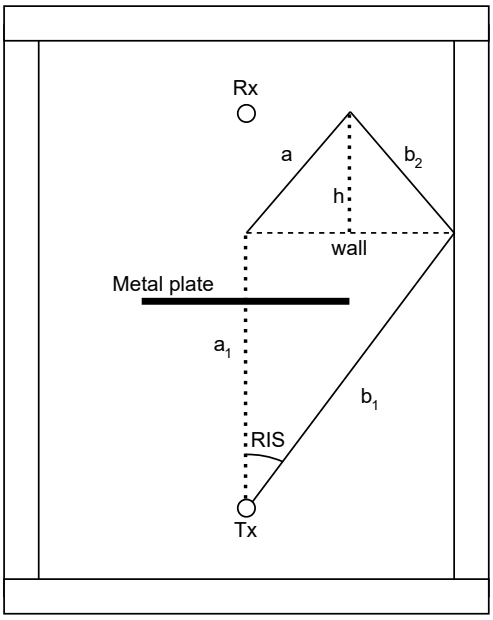


Figure 8.2: Illustrations for how the algorithm calculates the distance between Tx and Rx. The setup is when a metal plate is in the LOS, thus creating a NLOS scenario.

The variable b_2 is a side in an arbitrary triangle, which will be used to calculate the remaining direct distance to the Rx position. Calculating the height of the arbitrary triangle will allow to determine the remaining distance. The definition of calculating the height of an arbitrary triangle is

$$h = \sin\left[\arccos\left(\frac{a^2 + wall^2 - b_2^2}{2 \cdot a \cdot wall}\right)\right] \cdot b_2 \tag{8.10}$$

Here, a is assumed to be equal to b_2 as there is no way to predict what a should be otherwise with the given data. When both a_1 and h are known, they can be added together to obtain an estimation of the distance.

$$D_E = a_1 + h \tag{8.11}$$

where

D_E	Estimated direct distance between Tx and Rx
a_1	Distance estimated of the first triangle
h	height of the second arbitrary triangle

This is an estimation of the direct distance between the transmitter and receiver. A flaw in this method, is the assumption that the transmitted signal is only reflected once before hitting the

receiver, and that there is no scattering of the signal. In the real world, the signal will scatter and experience multi-path propagation before arriving at the receiver. Furthermore, the directivity of the antennas used will impact *when* the max magnitude is received. Therefore, the time of flight cannot be defined as when the max magnitude of the signal is received, subtracted by the time Tx transmitted the signal. Instead, it will be the first time that a signal is received, even if the magnitude is lower than the max magnitude, subtracted with the time Tx transmitted the signal. To find the first time a signal is received, a threshold of X dB from the max magnitude is applied. When the max magnitude is identified, a check is performed on prior magnitudes within the threshold from the max magnitude. From the local maximums within the threshold, the first received signal is used, but only if the magnitude is N samples away from the original maximum. If no prior magnitudes are within the threshold, the first maximum is used. An illustration of this can be seen in the figure 8.3.

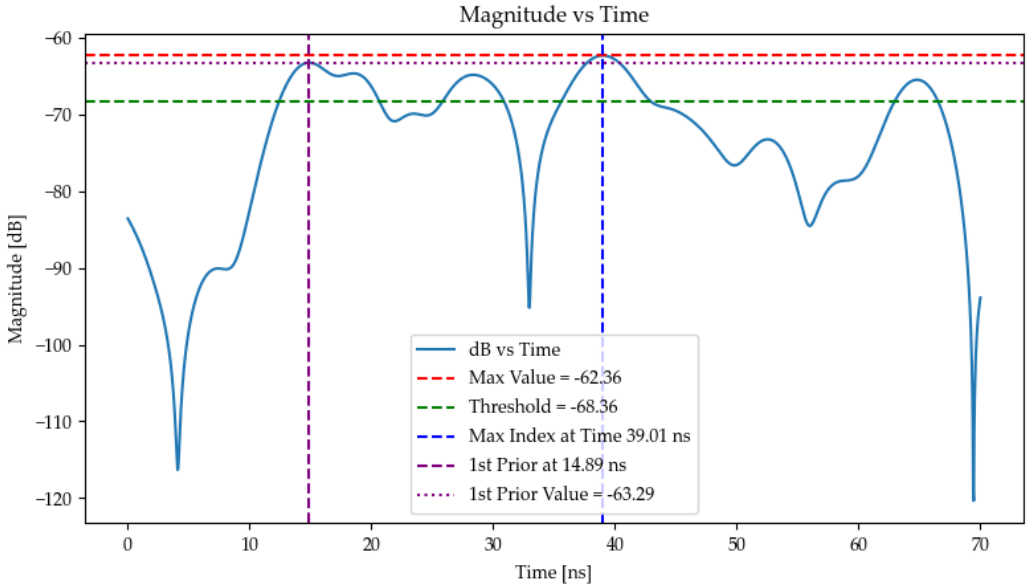


Figure 8.3: Magnitude plot describing how to possibly find the first received signal even though it is not the strongest signal. Threshold is set to -6dB from max magnitude. The 1st prior value is considered as the local maximum.

Furthermore, in the LOS scenario, the distance can be found by multiplying the first Time of Arrival (TOA) of the signal with the speed of light. This is equivalent to finding D in equation 8.6. The LOS scenario does not use the trigonometry method, instead it only uses the method for finding the first TOA.

8.3.2 Group delay distance determination

When in line of sight, the free space path loss equation(2.4) is able to give an estimation on the distance between the Tx and Rx. However, it requires the knowledge of the transmitting power. This can be difficult to obtain, as it highly depends on the hardware used. Therefore, the phase, or more accurately, the group delay of the signal will be used. The group delay as stated in section 3.1.5 is a way to describe the transmit time from Tx to Rx. This means that by knowing the group delay, it is possible to estimate the time of flight (ToF), which in LOS would be the signal's estimated path between Tx and Rx. By finding the minimum of the group delay,

the shortest transmit time can be found, which would be the most direct signal propagation path between Tx and Rx. By multiplying the minimum group delay with the speed of light, an estimation of the direct distance of the signal path is obtained.

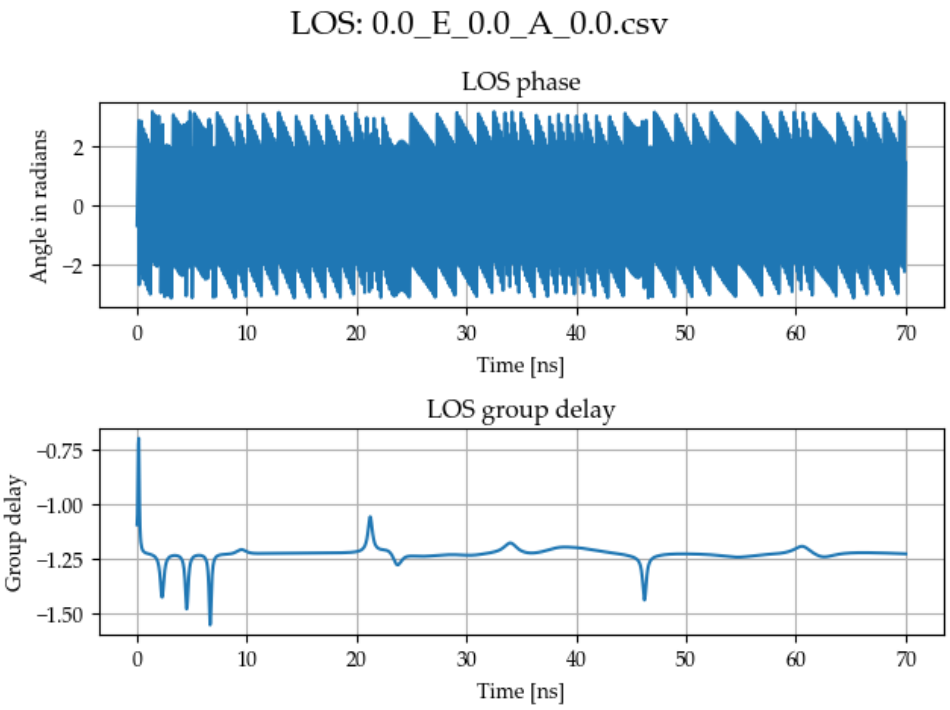


Figure 8.4: The phase and the corresponding group delay for LOS between Tx and Rx.

As seen in figure 8.4, the group delay is noisy as the signal will endure multi-path propagation, scattering, fading etc. before being received. Therefore, to obtain a more accurate distance estimation, a Moving Average (MA) filter will be used to filter out any potential outlying peeks caused by the environment the signal is propagating in. A moving average filter utilizes a window of sampled values and takes the average of these values. Mathematically it is expressed as

$$y[n] = \frac{1}{N+1} \sum_{k=0}^{N} x[n-k] \tag{8.12}$$

where

N	Window length
$y[n]$	n'th output
$x[n-k]$	Values of the input which will be averaged

This formula states that the MA filter will smoothen the measurements by computing the average between the current measurement $x[n]$ and past N measurements. The group delay plot before and after the MA filter can be seen in figure 8.5.

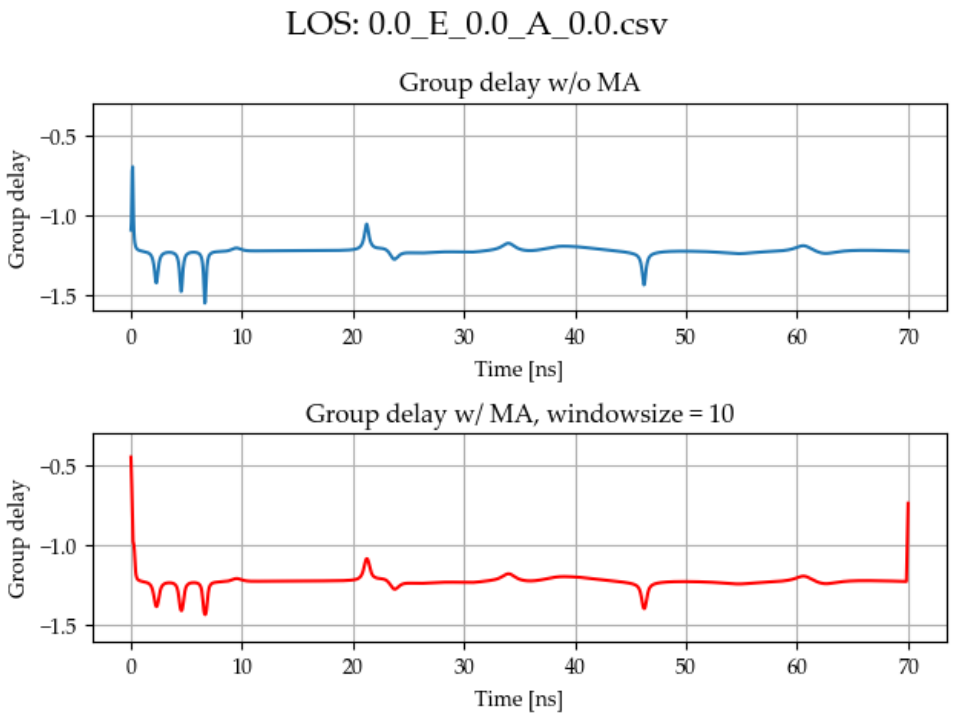


Figure 8.5: Group delay before and after the moving average filter with an order of 10.

This method will be able to calculate the distance the signal has traveled, and thereby give an estimation of how far apart the Tx and Rx is. However, a problem with the first and last data points of the filtered group delay will occur, as the filter tries to average over values which does not exist, due to the finite number of measurements. Therefore, in later iterations the first and last samples within the length of the order of the filter, will be discarded.

8.4 Implementation of localization

The methods described in section 8.3 utilizes determination cases and calculations, which can be implemented in different ways. Therefore, flowchart diagrams will be utilized to describe the behavior of these methods. These diagrams are depicted in the following sections along with a short description of these.

8.4.1 Trigonometry implementation

A flowchart of the trigonometry estimation used for localization is shown in figure 8.6. The method receives magnitude data and localizes the global maximum. The magnitude data prior to the identified global maximum is subsequently sampled to detect a local maximum. If a local maximum is detected, the TOA of the local maximum is used. If there is no prior local maximum, the global maximum TOA is used. Hereafter, the algorithm receives an input from the machine learning model, which determined if the signal received is in a case of LOS or NLOS. This decides how the algorithm should determine the distance. For LOS, the calculation uses the speed of light multiplied with the TOA to the global maximum. For NLOS, the calculation uses trigonometry as described in section 8.3.1 to calculate the distance.

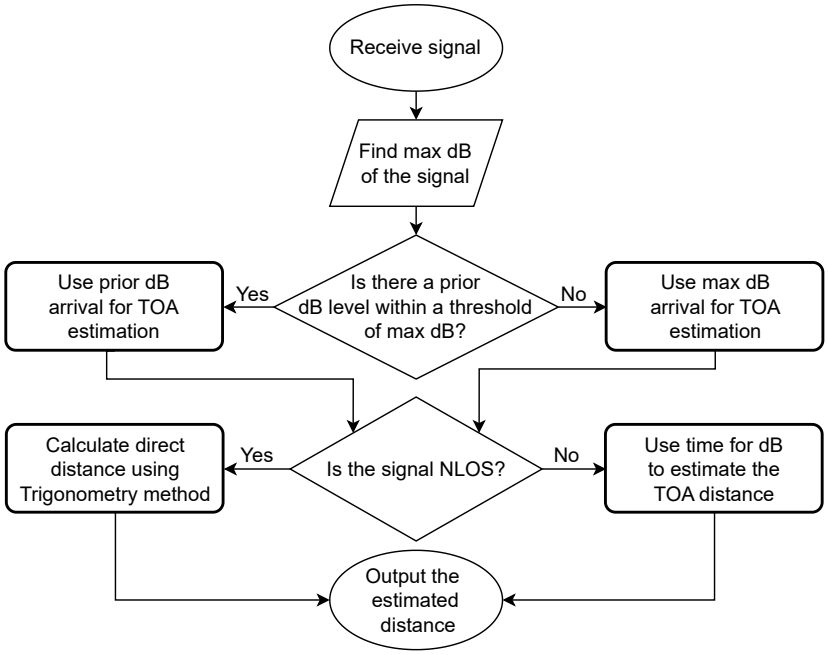


Figure 8.6: Flowchart of the trigonometry method for determining the distance between Rx and Tx.

8.4.2 Group delay implementation

A flowchart of the group delay estimation used for localization is shown in figure 8.7. The algorithm uses the phase of the inputted signal to calculate the group delay, as described in section 8.3.2. The group delay is passed through a moving average filter to reduce noise and outliers. The global minimum is determined of the filtered group delay, and multiplied with the speed of light. This gives an estimation of the shortest distance the signal has propagated.

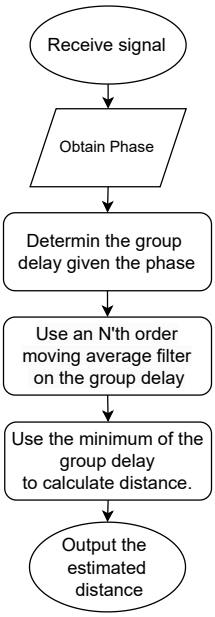


Figure 8.7: Flowchart of the group delay distance estimation method.

8.5 Comparison of methods

For the following results in this section, the distance from the transmitter to the RIS is **NOT** subtracted from the TOA, as it is described in section 8.3. This means that the term $\frac{d}{c}$ was not subtracted in equation 8.6.

By utilizing the data described in section 3, a comparison between the methods in a real life scenario is possible. Comparisons between the trigonometry method and group delay method for LOS can be seen in figure 8.8, for NLOS with big metal plate in figure 8.9 and for NLOS with small metal plate in figure 8.10. Every comparison is done with the Rx at 0.0°.

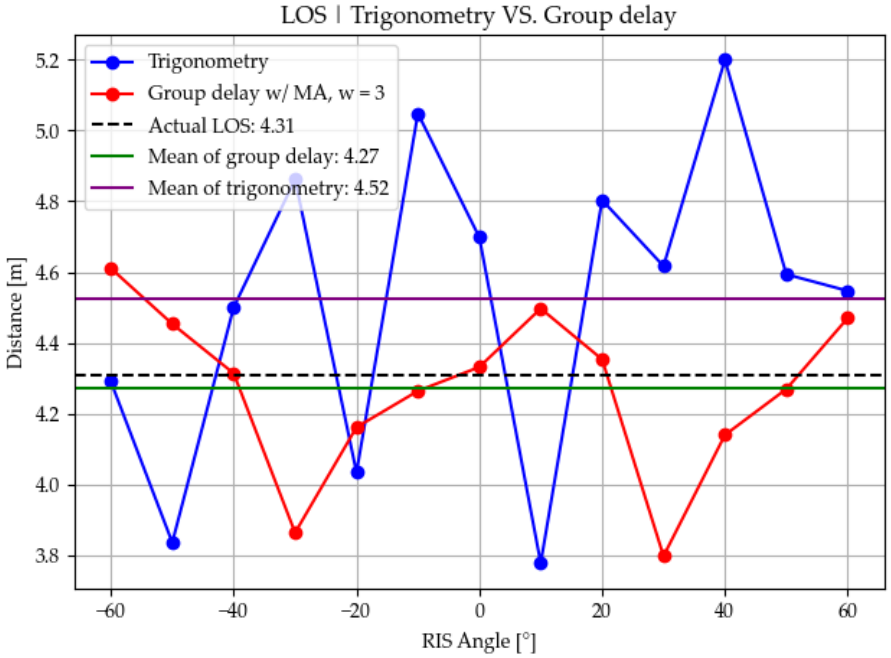


Figure 8.8: Comparison of the distances calculated for the trigonometry method and the group delay method described in section 8 for each angle. This is for the LOS case, where the Rx is at angle 0.0 deg. The angle of reflection for the RIS is mapped on the x-axis, while the distance in meter is mapped on the y-axis.

For the LOS case, five estimations (-40°, -10°, 0°, 20° and 50°) for the group delay method are within ±10 cm, while for the trigonometry algorithm (in 8.6) only one estimation ($-60°$) is within the same range. The mean for each method is also plotted, where the mean for the group delay is 4.27 m and for the trigonometry mean is 4.52 m. The errors are plotted in figure 8.11 and listed in table 8.1.

The mean calculated distance of the trigonometry algorithm indicates that the maximum peak utilized is received later than it was expected to. This could indicate that the RIS adds a delay to the transmitted signal. The distances estimated to be closer than the actual distance could indicate that the received maximum magnitude is from the Tx horn antenna directly. The direct distance between the Tx horn antenna and the receiver horn antenna is $\approx$ 3.7 meters. This means that even though the RIS is reflecting the strongest signal, it might not be received directly, and therefore appear more attenuated than the Tx horn antennas signal propagated backwards.

For the group delay method, only two estimations have error above 20 cm, which make the

method seemingly more accurate than the trigonometry method. The mean of the group delay method is 4.27 m which is 4 cm off from the actual distance. The estimations appear with a high variance around the actual distance.

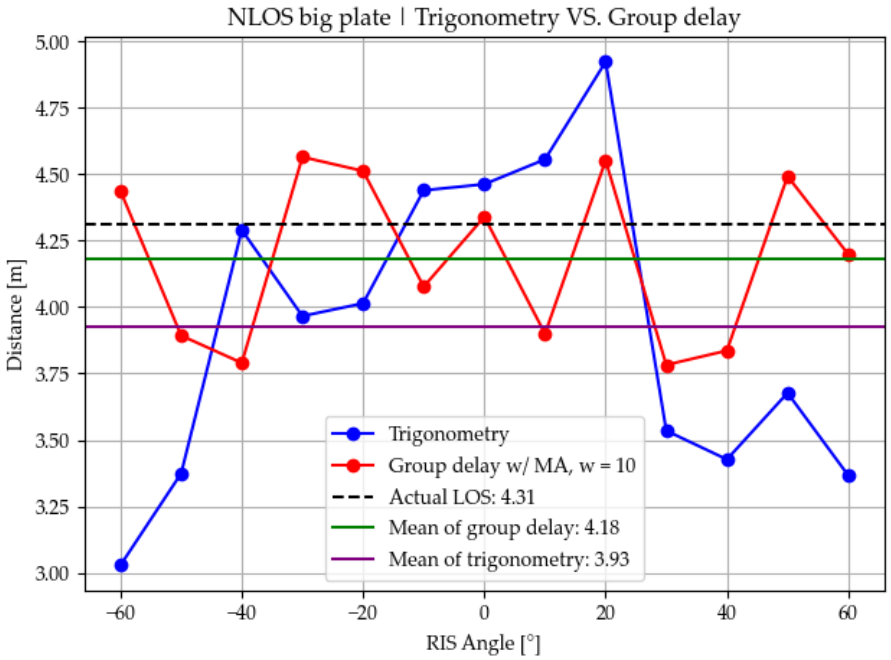


Figure 8.9: Comparison of the distances calculated for the trigonometry method, and the group delay method described in section 8, for each angle. This is for the NLOS case with the big metal plate, where the Rx is at angle 0.0 deg. The angle of reflection for the RIS is mapped on the x-axis, while the distance in meter is mapped on the y-axis.

For the NLOS case with the big plate, the trigonometry method appears more consistent at the different angles and thereby appearing less random compared to the LOS case in figure 8.8. The mean estimation is found to be lower compared to the LOS case. The trigonometry mean is at 3.93 m, while for the LOS case it was 4.52 m, which could be caused by the way the angles change in combination with when the time of the first arrival is determined.

For the group delay case, the estimations has a smaller variance compared to the LOS case, and is still more accurate than the trigonometry method. The mean of the group delay is at 4.18 m, which is lower than for the LOS case of 4.27 m.

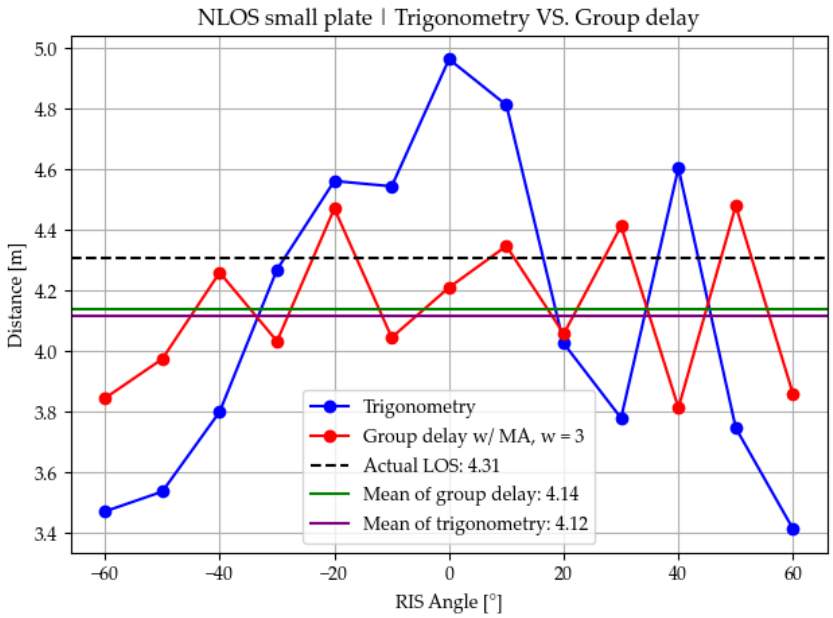


Figure 8.10: Comparison of the distances calculated for the trigonometry method and the group delay method described in section 8 for each angle. This is for the NLOS case with the small metal plate, where the Rx is at angle 0.0 deg. The angle of reflection for the RIS is mapped on the x-axis, while the distance in meter is mapped on the y-axis.

Similar to the NLOS for big metal plate and LOS cases, the trigonometry is less accurate than the group delay method. Comparing the estimations for each angle for the NLOS big metal plate and the small metal plate case, the pattern for each angle appears similar for the trigonometry method. One outlier is the 40° for the small metal plate, which is estimated to be further away the the actual distance.

The group delay had a smaller error for all cases relative to the trigonometry method. The larger errors for the NLOS cases indicates that, the plate is affecting the speed of the signal when going through.

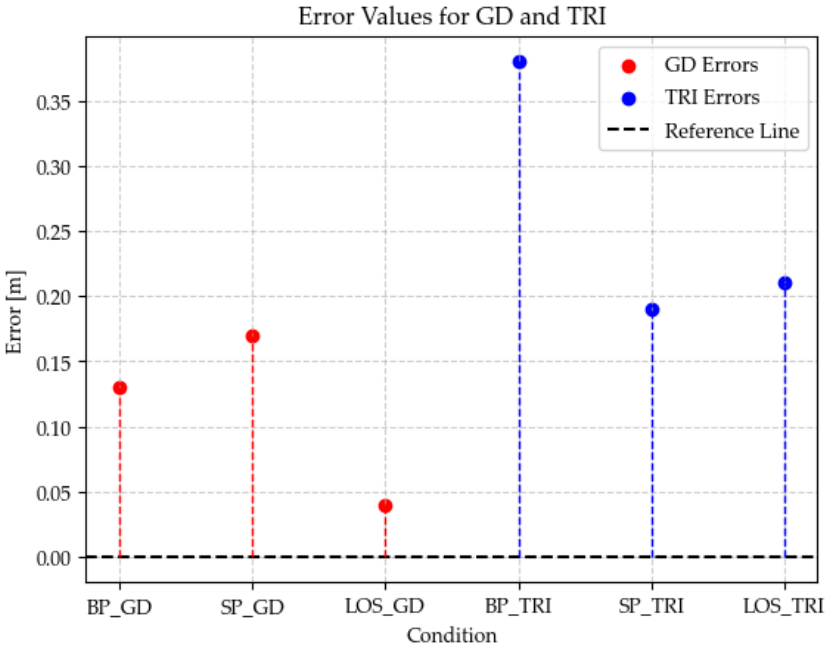


Figure 8.11: Plot of the errors of the means of both methods described in chapter 8. GD stands for Group Delay, while TRI stands for Trigonometry. BP and SP is for the NLOS cases with the big plate and small plate, respectively. The error on the y-axis is shown in meter.

The plots of each method in figures 8.8, 8.9 and 8.10 show that all mean measurements except one estimates the distance to be closer than it is. The trigonometry method for the LOS case had a longer estimated mean distance. The group delay method has a lower margin of error than the trigonometry method, as seen in Figure 8.11. The mean errors are also presented in table 8.1.

Case	BP	SP	LOS
Trigonometry mean error [cm]	38	19	21
Group delay mean error [cm]	13	17	4

Table 8.1: The mean errors for each method for each case. BP is NLOS with big plate, and SP is NLOS with small plate.

As the original test described in section 3, the receiving antenna is at a stationary position. Due to the stationary positioning of the measurement setup, it is not possible to validate the methods for dynamic distances, where the Tx is kept stationary. Therefore, a validation test will be performed, where Rx changes the distance relative to Tx.

Furthermore, the impact the RIS reflection has on the signal before it is received by the target, will have to be tested. This is to be more certain about the parameters which influences the signal. The validation test for both methods, and the RIS impact on the signal, can be seen in appendix B.1.

The choice of not subtracting the distance between the Tx and RIS from the TOA will not impact the accuracies of the calculations, as this is just a shift. However, this is only the case if the distance from the transmitting antenna to the RIS is not subtracted from the the actual signal path distance.

8.6 Summary of validation test for localization

The validation test for the two localization methods described in section 8.3 are performed in section B.1. However, as the data for the phase was invalid, it was not possible to validate the group delay calculations. Therefore, only the trigonometry method is validated. The test was performed at three different distances (i.e., 2 m, 3 m, 3.96 m) in both LOS and NLOS, where the NLOS was using a big metal plate placed in the center of the distance between the RIS and Rx. The results of the validation test showed that the trigonometry method could predict the distance between the Tx and Rx with the following deviation statistics of predicted distances:

- Overall maximum mean deviation of 27.1 cm
- Average LOS deviation of 4.83 cm
- Average NLOS deviation of 16.4 cm
- Average overall deviation of 10.65 cm
- Overall minimum mean deviation of 0.7 cm

The averages is taken from the predicted values of each distance, to get an overall estimate of the precision of the receiver. However to get a more precise deviation, more tests with different positions requires to be performed, with multiple data points from the same positioning.

Due to invalid phase data from the test, the group delay cannot be validated from the test. However, by looking at the comparison made in section 8.5 it can be hypothesized that the group delay method would have better predicted distances than the trigonometry method.

9. Discussion

From chapter 5-8, different models and localization algorithms have been designed and implemented to seek answers to the research questions from chapter 4. The following is a discussion of these designs and their results. Furthermore, the different datasets will be discussed.

9.1 Model Results

As seen in chapter 7, three different models types were tested, those being a DNN, SVM and LSTM. The LSTM provided poor results, being below 80% at all times. This was expected, given that it is usually used as a predictive model, thus being able to predict the next output. This is not really efficient in this use case, as there is not a pattern whether the data provided to the model is NLOS or LOS.

The SVM trained with random noise augmentation performed well on test data originating from the same environment as the provided training data, getting accuracies of around 96-97%. This is the highest accuracy achieved for any of the models when used on test data. When using K-fold cross validation on the SVM model, it was observed that the RBF kernel was consistently better than the polynomial kernel of order 3, providing the highest accuracy for the model. Although, the SVM model had the highest accuracy for the test set, it faltered heavily when exposed to varying noise levels, consistently having an accuracy around 60% for SNR levels below 25 dB when performing SNR Loop testing seen on Figure 7.10. This showed that the SVM model is not robust to such changes in the environment, or could suggest overfitting to the training data's noise level.

Compared to the SVM, the DNN performed very well across multiple cases, with an accuracy of around 90% in all tests, including both the test accuracy and all levels of SNR. It could be seen that augmenting the training data provided with random noise was effective for the SNR loop, increasing the robustness of the model, with no significant decrease on different SNR levels. It could also be seen that out of the five best performing models, three of them used the group delay as additional information, suggesting that the group delay has a good influence on the model results.

It could also be seen in the tests, that many of the preprocessing and augmentation techniques were effective in increasing the accuracy on their own. Both SNR and normalization augmentations, could also be seen increasing the robustness of the model, which is represented in the SNR accuracy. One issue encountered was the L2 normalization giving peculiar results for the SNR graph, as it is not realistic for the model to achieve the same accuracy across all SNR steps. This might be due to either a calculation or application error in the way the L2 normalization was implemented. In comparison, the Z-score augmentation displayed a more realistic behavior,

with slight changes in accuracy over each step of SNR. For combinations of methods, it could be seen, that the accuracy did increase slightly with multiple methods applied. Group delay seemingly provided good additional knowledge to the model, as it consistently increased the accuracy when applied.

9.2 Feature importance

During the implementation phase of the project, while experimenting with additional features to provide the ML models (See appendix F.1.1 for an in-depth description), it showed that the DNNs did not look for a peak amplitude inherently as part of the data seen in figure Figure F.2. To look further into this 3 statistical features were added to the features given to the DNN i.e. kurtosis, RMS delay and peak amplitude. It still turned out that these RMS delays on average are the most important in what the final classification as seen on Figure F.3, but peak amplitude and kurtosis proved to be insignificant seen in Table F.2. Additionally it turned out that the amplitude data was more important than the phase data.

It was discovered that supplying the 3 statistics twice i.e. the last 6 elements of each tensor are the 3 statistics two times in a row. This approach yielded higher accuracies and the amplitude was now a more significant feature as seen on Figure 9.1. Each statistic has a postfix of either **__DB** or **__DEG** depending on which order it is set in, where the **__DB** comes first.

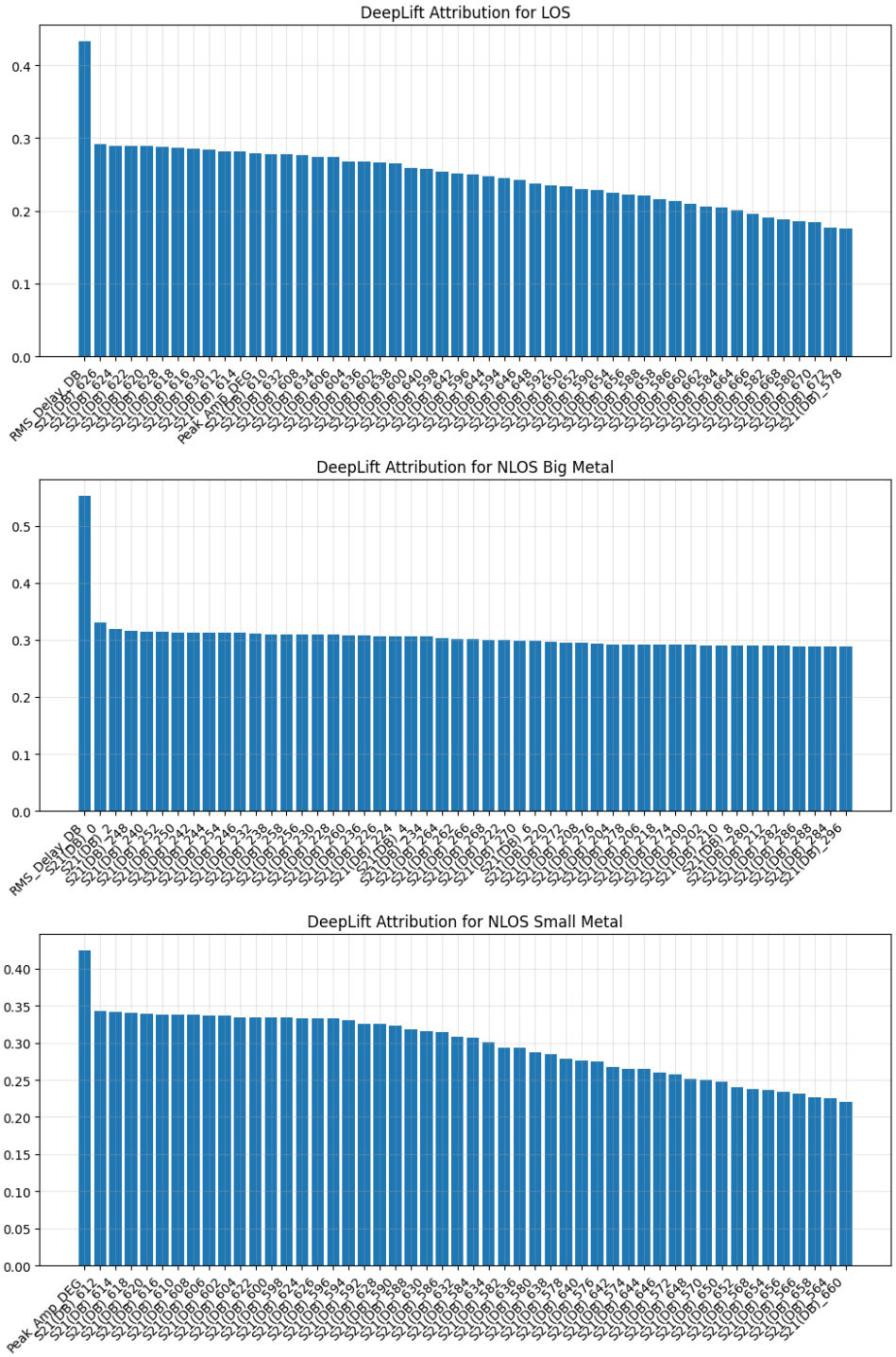


Figure 9.1: Feature importance for 3 different classes with the 2 different NLOS cases, when kurtosis, RMS delay and peak amplitude are given twice to the DNN.

Initially it was hypothesis that the reason why this worked better was that the DNN architecture was structured in an encoder style i.e. the hidden layers all acted as bottleneck layers [13, p. 486], but alternative DNN architectures yielded no better results.

9.3 Datasets

For the different datasets, it could be seen that the original data generally provided good results, while the high frequency lab data was more inconsistent. This is likely due to the big difference in the two environments, with the HF dataset also being very different from the original dataset in terms of values of the datapoints.

As they have different sampling periods, and to avoid the relying on the specific time axis of the data, the time dimension was removed. While early-on models had a good accuracy with the time axis on the OG dataset. The time dimension would have been an important part of the performance, where different lengths of the measured dataset may cause issues in determining the classification.

The difference in sampling periods for the given dataset was to take most reflections into account. The sampling period of the OG and HF datasets are respectively 70 ns and 240 ns but with the same amount of datapoints. This causes each dataset to be greatly different in the time domain, and appear stretched when compared in sample space. These datasets are therefore not directly comparable.

This is usually not an issue, as comparisons like this are not done in standard applications. As the model only uses the direct data given for its predictions, it may not be capable of accurately predicting on both datasets at the same time. Exceptions can be cases where it has been trained on both datasets from the start.

Although three different datasets were supplied, they differed greatly in results and actual data, making it difficult to check the trained models for their general performance. The generalization tests visualize this with models trained on an individual dataset seeing most of the data from other environments as NLOS data, as described in Section 7.1.1.

There is also the issue of some of the LOS data in reality being NLOS data due to the directivity of the Rx. The Rx is a horn antenna, meaning some of the data labeled as LOS is pointing away form the RIS, meaning only reflections can reach the antenna, thereby making it NLOS. This might be the cause of some of the classification errors for LOS detection.

9.4 Localization

In chapter 8, two methods were used to determine the direct distance between the two antennas from the data, as stated in research question 3 in chapter 4. The first method was to use the time for the peak of the received power multiplied with the speed of light, hereby calculating the time of flight (ToF). This in combination with trigonometric equations, was used to calculated the direct distance. The second method utilized the group delay of the signal to determine the distance by multiplying the lowest group delay value by the speed of light, thus estimating the distance between the antennas.

For NLOS with the small metal plate (Figure 8.10), the received signals magnitude is plotted for $-30°$ and $60°$ with the trigonometry parameters shown in Figure 9.2, as these are the best and the worst estimations for the trigonometry method. The distance for the data with $-30°$ is estimated to 4.27 m, which is off by 4 cm, and the distance for $60°$ is estimated to 3.41 m, which is off by 90 cm. Seen in figure 9.2, the main lobes of the two signals are almost identical with only 0.07 ns (2.1 cm) in difference, so the difference is the algorithm for estimating the distance,

as the different angles result in different triangles. This could indicate that, the algorithm for the trigonometry contains errors or is missing data on the room, such as distances to other walls.

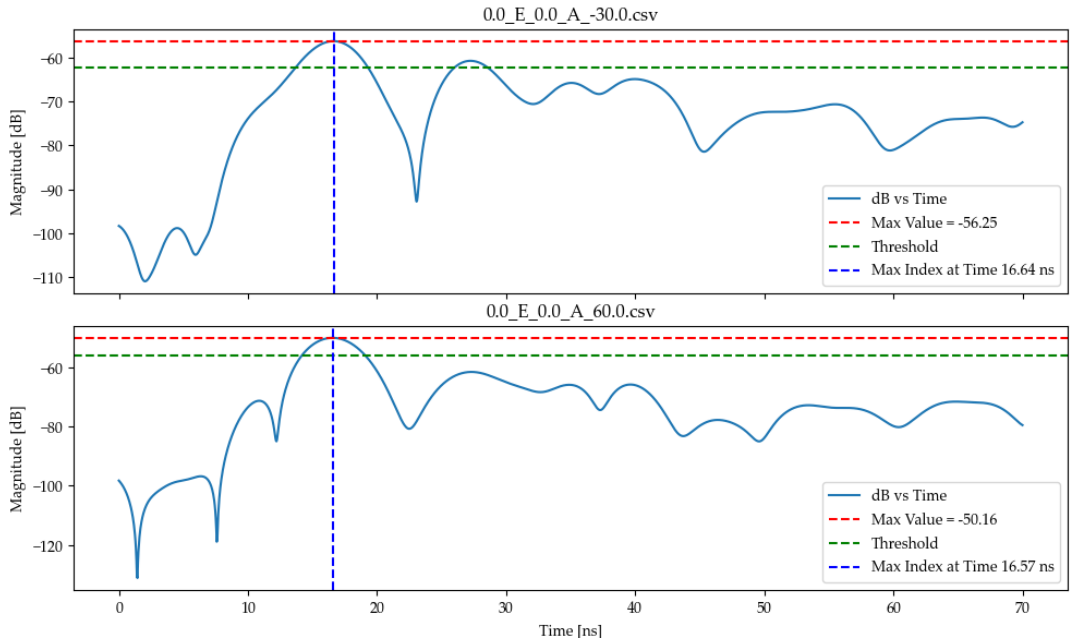


Figure 9.2: Plot of the received signal for the case of NLOS with the small metal plate for $-30°$ and $60°$, respectively, with the trigonometry parameters shown.

Furthermore, as seen in Figure 8.9 and Figure 8.10, as the angles increase in either direction, the algorithm estimates the distances to be too close relative to the actual distance.

In Figure 8.11, the group delay is shown to be the best estimator of the distance between the antennas compared to the trigonometry method. The largest error being 17 cm for the group delay for NLOS with small plate compared to 38 cm for the trigonometry for NLOS with the big plate. The smallest error being 4 cm for the group delay for the LOS case and 19 cm for the trigonometry for the NLOS case with the small metal plate.

The differences from LOS to NLOS for the group delay method could indicate that, the physical conditions are affecting the signal propagation time. The physical condition is big enough from the metal plate to alter the delay, which in future research will have to be taken into account to estimate a better distance. Furthermore, as different antennas affect the group delay differently, this has to be considered.

In real scenarios, several other devices will be present, which will result in interference at the receiver. To make the methods usable, these interferences are needed to be filtered out, especially when utilizing the group delay. This is due to other signals' phases having an affect the phase of the wanted signal, hereby making it more difficult to obtain a clear group delay. Further research is required to improve the estimation of the maximum signal magnitude for the trigonometry method.

To improve the group delay method, a sweep (or several) could be made to determine the average of all the measurements. The error for the LOS case appears to be sufficiently small for use in a real scenario. Using more data points could eliminate the error, hereby getting an even lower error.

10. Conclusion

This project aimed to determine if it was possible to develop a machine learning model capable of differentiating LOS and NLOS data. For this, the following problem statement was made:

How can a machine learning model be trained to recognize a line of sight and non line of sight signal from a transmitter, and is deep learning models better than classical machine learning models?

To fully understand the problem definition, three research questions were constructed to support the statements needed to conclude the problem statement.

The first question being the impact of different preprocessing and augmentation methods, which could be seen as having a positive impact on the performance when used both on its own, as well as in combination with multiple methods. Especially the augmentation methods of SNR application and normalization proved to increase the DNN's robustness to changes in the SNR.

The second question looked at the generality and robustness of the implemented models to test if it was possible to make a model usable in different environments, or if the usage of transfer learning would improve this. The solution found was to train the model on a dataset consisting of the combined datasets from the start, proving to give good results for both use cases.

The report demonstrated that machine learning is capable of recognizing LOS from NLOS with the use of an RIS, using the datasets provided. The DNN designed achieved an average accuracy of 94%, while the SVM performed slightly better at 97%. However, during SNR loop testing, different SNR of the noise severely impacted SVM performance (figure 7.9), showing the poor robustness to unseen noise levels. Here, the DNN only saw an insignificant decrease in performance. From this it can be concluded that the DNN is more suitable than the SVM, as it is more robust to noise, which is likely present in real applications.

The last question is to see how accurately the distance between the Tx and Rx could be found using the data provided, as well as using the knowledge of whether it is LOS or NLOS from the model. The two localization methods proposed were the trigonometry and group delay. Both used different data processing to estimate the direct distance, with testing on the OG dataset showing that group delay outperformed the trigonometry method in mean accuracy.

To validate the accuracies and the impact the RIS had on the transmitted signal, a validation test was performed as seen in B.1. The test was performed with three different distances (i.e., 2 m, 3 m, and 3.96 m). The results showed that the trigonometry method could predict the direct distance with an average deviation of 4.83 cm for LOS (found in B.3), 16.4 cm for NLOS (found in B.2) and an overall average of 10.61 cm (found in B.4). The data for the phase was invalid, and the accuracies for the group delay can therefore not be concluded.

Citations

[1] Petar Popovski. *Wireless Connectivity. An Intuitive and Fundamental Guide*. 1st. ISBN: 978-0-470-68399-6. John Wiley & Sons, 2020.

[2] R. Sarvendranath and A. K. R. Chavva. *Low-Complexity Joint Antenna Selection and Beamforming for an IRS Assisted System*. ISBN:978-1-7281-9506-3. IEEE, 2021.

[3] Andreas F. Molisch. *Wireless Communications. From Fundamentals to beyond 5G*. 3rd. ISBN: 978-1-119-11720-9. John Wiley & Sons, 2023.

[4] D. Dardari et al. *LOS/NLOS Near-Field Localization With a Large Reconfigurable Intelligent Surface*. Electronic ISSN: 1558-2248. IEEE, 2022.

[5] Ming Shen et al. "Lab to Multiscene Generalization for Non-Line-of-Sight Identification With Small-Scale Datasets". In: *IEEE* (2023).

[6] *Antenna Patterns and Their Meaning*. Tech. rep. C11-422494-00 8/07.

[7] Constantine A. Balanis. *Antenna Theory*. 4th. ISBN: 978-1-118-64206-1. John Wiley & Sons, 1997.

[8] Pan Tang et al. "Channel Measurement and Path Loss Modeling from 220 GHz to 330 GHz for 6G Wireless Communications". In: (May 2021). URL: https://ieeexplore.ieee.org/document/9444236.

[9] Hossein Pishro-NikHossein Pishro-Nik. *Introduction to Probability, Statistics, and Random Processes*. 1st. ISBN-10 : 0990637204. Kappa Research, 2014.

[10] Emil Bjôrnson et al. *Reconfiguarble Intelligent Surfaces: A Signal Processing Perspective With Wireless Applications*.

[11] Artzai Picon et al. "Why Deep Learning Performs Better Than Classical Machine Learning Computer Science Perspective Why Deep Learning Performas Better Than Classical Machine Learning 1. - Machine Learning VS Deep Learning". In: *Dyna (Bilbao)* (Mar. 2020).

[12] Luca Pietro Giovanni Antiga, Eli Stevens, and Thomas Viehmann. *Deep Learning with Pytorch*. 1st edition. ISBN: 9781617295263. Manning Publications Co. LLC, 2020.

[13] Aurélien Géron. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow. evt. undertitel*. 2nd Edition. ISBN: 9781492032649. O'Reilly Media, Inc., 2019.

[14] Chip Huyen. *Designing Machine Learning Systems. An iterative Process for Production-Ready Applications*. First edition. ISBN: 9781098107963. O'Reilly Media, Inc., 2022.

[15] Zhi-Hua Zhou. *Machine Learning*. 1st ed. ISBN: 978-981-15-1967-3. Springer Singapore, 2021.

[16] Emilija Ristic. *Learning - Nural Network*. URL: https://www.figma.com/community/file/1138558582458670232/learning-nural-network.

[17] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. First edition. ISBN: 9780387310732. Springer New York, NY, 2006.

[18] Andreas Antoniou and Wu-Sheng Lu. *Practical Optimization*. ISBN: 9781441943835. Springer, 2007.

[19] Simon J.D. Prince. *Understanding Deep Learning*. 1st. The MIT Press, 2024.

[20] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: 1412.6980 [cs.LG]. URL: https://arxiv.org/abs/1412.6980.

[21] Meta AI. *Adam*. Meta AI. URL: https://pytorch.org/docs/stable/generated/torch.optim.Adam.html.

[22] Scikit-learn. *Support Vector Machines*. Scikit. URL: https://scikit-learn.org/1.5/modules/svm.html.

[23] Andrej Karpathy. *The Unreasonable Effectiveness of Recurrent Neural Networks*. https://karpathy.github.io/2015/05/21/rnn-effectiveness/. (Accessed on 01/10/2024). May 21, 2015.

[24] Ashish Mehta. URL: https://ashish-mehta.medium.com/support-vector-machine-svm-algorithm-for-machine-learning-350fe9139a52.

[25] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-term Memory". In: *Neural computation* 9 (Dec. 1997), pp. 1735–80. DOI: 10.1162/neco.1997.9.8.1735.

[26] Sergey Ioffe and Christian Szegedy. *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*. 2015. arXiv: 1502.03167 [cs.LG]. URL: https://arxiv.org/abs/1502.03167.

[27] Scikit-learn. *SVC*. scikit-learn. URL: https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html.

[28] Asmaul Hosna et al. "Transfer learning: a friendly introduction". In: *Journal of Big Data* 9.1 (2022), p. 102. ISSN: 2196-1115. DOI: 10.1186/s40537-022-00652-w. URL: https://doi.org/10.1186/s40537-022-00652-w.

[29] *tf.Tensor*. Tensorflow. URL: https://www.tensorflow.org/api_docs/python/tf/Tensor.

[30] Meta AI. *Automatic Differentiation with torch.autograd*. Meta AI. URL: https://pytorch.org/tutorials/beginner/basics/autogradqs_tutorial.html.

[31] Nvidia. *9.1. Data Transfer Between Host and Device*. Nvidia. URL: https://docs.nvidia.com/cuda/cuda-c-best-practices-guide/index.html#data-transfer-between-host-and-device.

[32] Israel Leyva-Mayorga. "High Performance Programming - Lecture 9: "GPU programming"". Lecture powerpoint. 2023. URL: https://www.moodle.aau.dk/mod/resource/view.php?id=1509935.

[33] Troels Pedersen. "Kalman Filter lecture notes [LN]". Lecture powerpoint. 2023. URL: https://www.moodle.aau.dk/mod/resource/view.php?id=1734783.

[34] *Fundamentals of Statistical Signal Processing, Volume 1: Estimation Theory*. Pearson Education. ISBN: 9788131728994.

[35] wolfram. *L2-Norm*. wolfram. URL: https://mathworld.wolfram.com/L2-Norm.html.

[36] Zach Bobbit. *Z-Score Normalization*. URL: https://www.statology.org/z-score-normalization/.

[37] Shaun Turney. *What is Kurtosis? | Definition, Examples & Formula*. Scribbr. URL: https://www.scribbr.com/statistics/kurtosis/.

[38] Andrea Goldsmith. *Wireless Communication*. 1st. ISBN: 978-0-511-13315-2. Cambridge University Press, 2005.

[39] Pevand Bahramzy and Gert F. Pedersen. *Group Delay of High Q Antennas*. Electronic ISBN:978-1-4673-5317-5. IEEE, 2013.

[40] Vincent Vanhoucke, Andrew W. Senior, and Mark Z. Mao. "Improving the speed of neural networks on CPUs". In: 2011. URL: https://api.semanticscholar.org/CorpusID:15196840.

[41] Chang-Bing Zhang et al. "Delving Deep into Label Smoothing". In: (July 2022). DOI: https://doi.org/10.1109/TIP.2021.3089942.

[42] ETSI. *User Equipment (UE) radio transmission and reception; Part 4: Performance requirements*. ETSI. URL: https://www.etsi.org/deliver/etsi_ts/138100_138199/13810104/17.07.00_60/ts_13810104v170700p.pdf.

[43] *PyTorch*. Meta AI. URL: https://pytorch.org/.

[44] *scikit-learn*. URL: https://scikit-learn.org/.

[45] *pandas*. URL: https://pandas.pydata.org/.

[46] *Numpy*. NumPy team. URL: https://numpy.org/.

[47] *numpy.gradient*. URL: https://numpy.org/doc/2.1/reference/generated/numpy.gradient.html.

[48] *torch.Tensor*. URL: https://pytorch.org/docs/stable/tensors.html#torch.Tensor.

[49] Steven M. Kay. *Intuitive Probability and Random Processes using MATLAB®*. 2006. ISBN: 978-0-387-24157-6.

[50] Tsung-Yi Lin et al. "Focal Loss for Dense Object Detection". In: (Feb. 2018). DOI: https://doi.org/10.48550/arXiv.1708.02002.

[51] Tomás Mikolov, Martin Karafiat, and Sanjeev Khudanpur Lukás Burget Jan "Honza" Cernocky. "Recurrent neural network based language model". In: (2010). URL: https://www.fit.vutbr.cz/research/groups/speech/publi/2010/mikolov_interspeech2010_IS100722.pdf.

[52] Robin M. Schmidt. "Recurrent Neural Networks (RNNs): A gentle Introduction and Overview". In: (2019). DOI: https://doi.org/10.48550/arXiv.1912.05911.

[53] Ilya Sutskever. "Training Recurrent Neural Networks". In: (2013). DOI: ISBN:978-0-499-22066-0.

[54] Alexandre Duval. "Explainable Artificial Intelligence (XAI)". In: (Apr. 2019). DOI: 10.13140/RG.2.2.24722.09929.

Appendices

A. Extended analysis

A.1 Loss functions

The following sections describe how the given loss function works, and why it is used in its use-case.

Mean Square Error

The mean square error function can determine how far the predicted value is from the target values. The mean square error is calculated by comparing the average squared difference between the predicted and the actual target values within a dataset, and is calculated as in A.1 [49, Eq. 7.40].

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \tag{A.1}$$

Where:

Binary Cross Entropy

Binary cross entropy is a loss function used in binary classification problems, where the output has two possible values (0 or 1). The model will try minimizing the loss function to obtain a higher prediction accuracy during training. The loss function for the binary cross entropy is defined as in equation A.2 [13, Eq. 4-17].

$$\text{BCE} = -\frac{1}{N}\sum_{i=1}^{N}\left[y_i \cdot \log(p(y_i)) + (1 - y_i) \cdot \log(1 - p(y_i))\right] \tag{A.2}$$

where:

BCE	Binary Cross Entropy
N	Number of data points
y_i	The label of the data point (0 or 1)
$p(y_i)$	The probability of the point being 1

Cross Entropy Loss

The cross entropy loss function is used to calculate the difference between the two probability distributions, the true labels and the predicted probabilities. It quantifies how well the prediction probabilities fit with the true labels, where the goal is to achieve something near zero. When the model has calculated the probabilities, cross entropy calculated the deviation of these probabilities from the true labels. A clear benefit of the cross-entropy function is that it punishes confident incorrect predictions more than small errors [13]. If the model uses multiple output classes, then it is desired to improve across all the samples. To do this the multi class cross entropy function can be utilized as it is defined in equation A.3 [13, Eq. 4-22].

$$L = \frac{1}{N}\sum_{k=1}^{N}\left(-\sum_{i=1}^{C} y_i \cdot \log(\hat{y}_i)\right) \tag{A.3}$$

where:

- L Computed loss of prediction compared to true label
- k Number of samples in a batch
- C Amount of classes
- y_i True labels
- $\hat{y}_i$ predicted class probabilities

Focal loss

The focal loss is an application of Cross-entropy loss that aims to reduce the loss contribution of well-classified examples, therefore putting more focus into classifying the harder samples accurately. This is useful in case the loss has a remarkable variance despite higher epochs. Classifying the harder samples is done through the introduction of two variables, namely α and γ [50]. α is responsible for the balancing of the class imbalance, where γ is responsible for how much focus there will be on hard to classify datasets. A higher γ value will result in higher focus on the hard to determine values. The equation for focal loss is seen in (A.3)[50]:

$$L_{focal} = \frac{1}{N}\sum_{k=1}^{N}(-\sum_{i=1}^{C}\alpha(1-\hat{y}_i)^{\gamma} \cdot y_i \log(\hat{y}_i)) \tag{A.4}$$

where:

- α balancing factor for class imbalance
- γ level of focus on hard to classify datasets

By using the focal loss function, it is possible to be even harsher on wrong confident estimations compared to the cross-entropy loss, thereby evening out the loss and achieving a lower overall loss. The extra focus on the hard to determine values can result in overfitting, which is a necessary concern when deciding the amount of epochs.

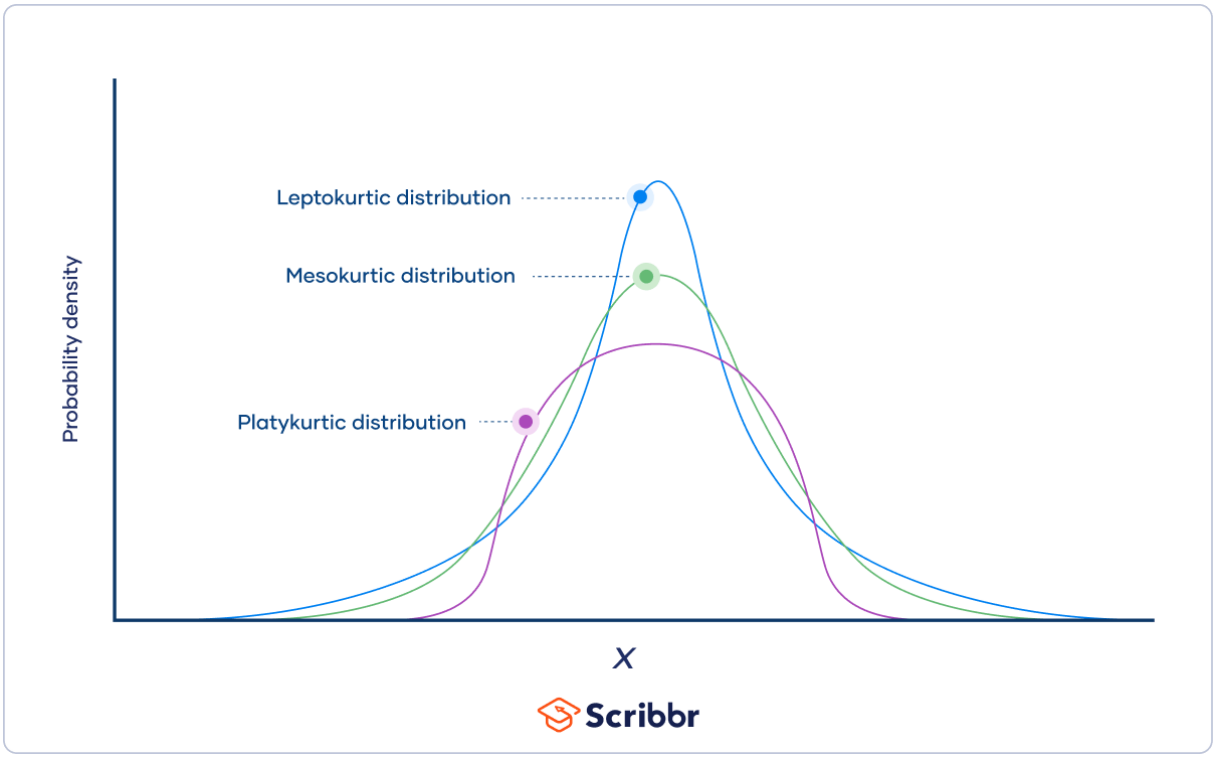


Figure A.1: Different kurtosis distributions [37].

A.2 Kurtosis Distribution comparisons

A.3 RNN

Like the DNN, the RNN consists of a network of nodes similar to that of the MLP on Figure 2.7, but unlike the MLP, the RNN architecture contains memories of prior entries that came before. This allows for the model to estimate the t value with knowledge of the prior input *t-1*. This is an especially useful feature, when the input data is sequential, like in a sentence or in a sequential signal where the order of the data matters. These sequences can even be used to predict incoming data or words. The RNN makes this possible through backwards propagation of the previous samples. A simple RNN with this behavior is visualized on Figure A.2.

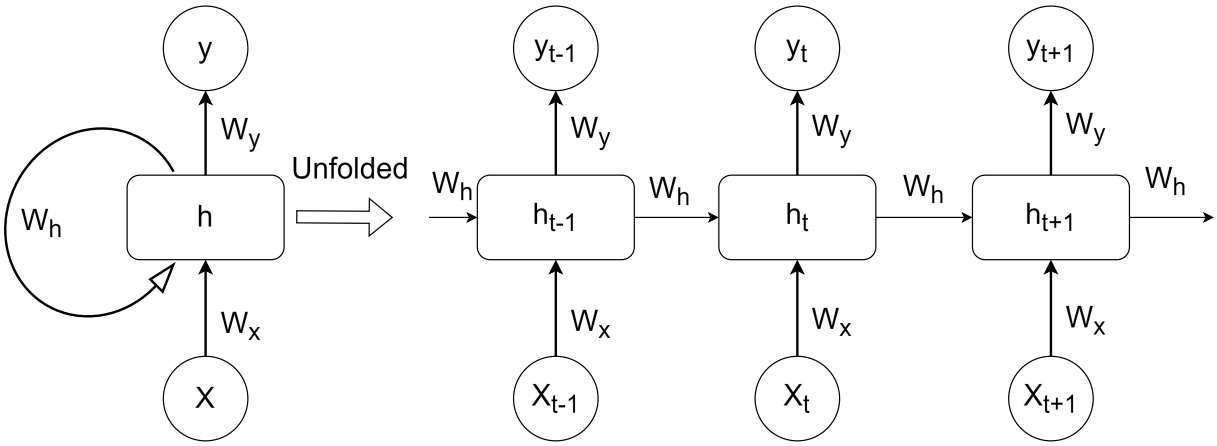


Figure A.2: The simple recurrent neural network and its construction. On the left is the simple collected RNN, with its BP pool. To the right, the RNN is unfolded to showcase how the time t receives information about the past sample and sends information to the future sample.

Where the input is computed together with the previous knowledge to determine the output. The computed input is hereafter sent to the output and the computed value is stored in the memory for the next input. The result in the hidden layer *h(t)* is a result of the previous sample, the input and the weights between the nodes and is described as [23]:

$$h(t) = f(W_x \cdot X(t) + W_h \cdot h(t-1)) \tag{A.5}$$

With the output equation for the output layer being[51]:

$$y_k(t) = g(h(t) \cdot W_y) \tag{A.6}$$

Where:

$h(t)$	Computed result of the hidden state for t sample
W_x	Weight between input and hidden layer h
$X(t)$	Input for t sample
W_h	Weight between the previous input and h
$h(t-1)$	Previous computed result of the hidden state for the sample t-1
$y_k(t)$	The output of the RNN for t sample
W_y	Weight of the hidden to output connection
$f(z)$	Activation function
$g(z)$	Softmax function

The activation function f is for the RNN a non-linear function like a sigmoid, tanh or ReLU function, while the *g(z)* function is a softmax function [51].

Through these equations, the RNN uses previous iterations of data to calculate the newest, and potentially predict future samples. The RNN however suffers from an issue over longer iterations. If the activation function $f(z)$ is either sigmoid or tanh, then as the iterations increases, the resulting value falls as it continues to approach zero and might lead to the Vanishing gradient

descent problem for the RNN. Another source of instability for the RNN is the adaptation of the BP algorithm for RNNs, called Backpropagation Through Time (BPTT).

The BPTT is a technique used to compute the gradients by unrolling the time and applying BP to it. The method is utilized to find the total loss of the gradient using calculated losses for previous samples. The loss is then sent to the BP function and from this point behaves similar to a DNN, with the weights being updated in the end. As this process proceeds, the overall loss term can become very large. When the loss-term becomes large, it reaches a numerical instability because eigenvalues smaller than 1 cause vanishing gradients while eigenvalues larger than 1 causes exploding gradients [52]. Because of this instability issue caused as the knowledge of previous terms increase, a limit on the length is typically put in place, thereby truncating. This is called a Truncated BPTT and it establishes a maximum amount of time steps that the gradient can go back to [53], thereby ignoring anything outside the limit.

The truncated BPTT limits the samples which may in some cases be undesired due to higher requirement for knowledge of prior terms. To allow for a higher knowledge of the prior terms, the LSTM was developed so that it handles the vanishing gradient problem partially.

A.4 LSTM

This section contains the equations that determines an LSTM's internal structure and how each part is calculated. Based on the RNN, the LSTM aims to solve the issue of the vanishing gradient.

$$f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f) \tag{A.7}$$
$$i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i) \tag{A.8}$$
$$\overline{C}_t = \sigma(W_{\overline{C}} x_t + U_{\overline{C}} h_{t-1} + b_{\overline{C}}) \tag{A.9}$$
$$C_t = f_t \odot C_{t-1} + i_t \odot \overline{C}_t \tag{A.10}$$
$$O_t = \sigma(W_O x_t + U_O h_{t-1} + b_O) \tag{A.11}$$
$$h_t = O_t \odot tanh(C_t) \tag{A.12}$$

Where

Figure 2.13 and formulas shows how the LSTM handles the input data through three gates. It uses the forget gate f_t to determine how much of the new input that should be used in the long term memory, which gives a value between 0 and 1. Next up, the input gate i_t calculates the long term memory value C_t. Finally, the output gate O_t determines the next short term memory value h_t, which is also the output/prediction of the current input [25].

A.5 Fine-tuning

As stated earlier, fine-tuning aims to specialize a general model, so it becomes more accurate or effective in a specific use-case. Fine-tuning is considered a subset of transfer learning, as it is also implemented by training an original model on newer data. In case of fine-tuning, there are different approaches to how this is done, such as:

1. Additive, as seen on figure A.3

2. Partial fine-tuning

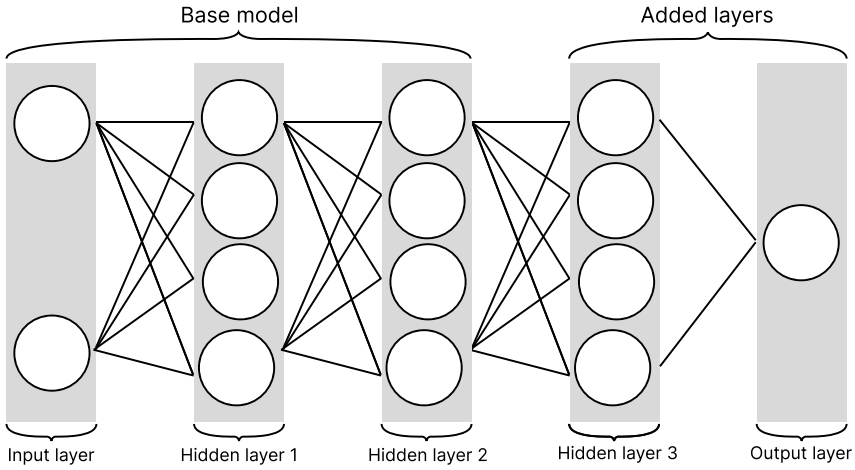


Figure A.3: A fine-tuned model using the additive approach, where an additional hidden layer has been added together with a new output layer

As the name implies, additive fine-tuning adds more parameters or layers to the model, while freezing the original layers. The model then only trains the new parameters and/or layers, while keeping the original components. This method increases stability, as the original model is still unchanged.

Partial fine-tuning, also referred to as selective fine-tuning, freezes only a part of the original model. Usually, it is the most important parameters that are changed, which in DNN's are often the outer layers, being the input and output layer. This is due to the inner layers often discerning finer features. The model is then trained on a new dataset, updating the outer layers for the given task.

A.6 Kalman filter

The kalman filter is a recursive implementation of the Linear Minimum Mean Square Error (LMMSE), which estimates an unobservable process $Y(n)$, given an observable process $X(n)$ through measurements. During this process of estimation, the error between the predicted parameters $\hat{\theta}$ used to generate $X(n)$ and the true parameters θ can be calculated using the Mean Squared Error (MSE)[34, S. 11.4]:

$$\text{MSE} = \mathbb{E}\left[\left(\theta - \hat{\theta}\right)^2\right] = \frac{1}{n}\sum_{i=1}^{n}(\theta_i - \hat{\theta}_i)^2 \tag{A.13}$$

Where:

- θ True parameter
- $\hat{\theta}$ Estimated parameter of θ

The task is now to find the minimum MSE, which can be done using the LMMSE estimator, as it is unbiased:

$$\hat{\theta} = h_0 + \sum_{n+1}^{N} h_n \cdot X_n \tag{A.14}$$

Where values of $h_0, \cdots, h_n$ are chosen to minimize the MSE from equation A.13. To estimate a dynamic system, the kalman filters makes LMMSE a recursive estimation process that utilizes previous estimated values. The KF introduces two types of equations:

1. System equations that functions as predictions rules, which describe the evolution of the state in time.
2. Measurement equations that functions as update rules to minimize errors between the predicted state and the observed state.

Prediction step

To estimate the state of the system, firstly the Kalman filter estimates the current state $\hat{Y}(n)$ as the following:

$$\hat{Y}(n|n-1) = h(n) \cdot \hat{Y}(n-1|n-1) \tag{A.15}$$

Where:

$\hat{Y}(n\|n-1)$	LMMSE estimate of $Y(n)$ based on $X(1), \cdots, X(n)$
$h(n)$	Known state transition coefficient
$\hat{Y}(n-1\|n-1)$	Previous prediction

The state transition coefficient is a known sequence $h(n)$ that weights how much the previous estimated state affects the current system state. This sequence corresponds to the h_n term as seen in Equation A.14.

Now, the MSE of the estimated $\hat{Y}(n)$ is calculated:

$$R(n|n-1) = h(n)^2 \cdot R(n-1|n-1) + \sigma_Z^2(n) \tag{A.16}$$

Where:

$R(n\|n-1)$	MSE between $Y(n+1)$ and $\hat{Y}(n+1n)$
$\sigma_Z^2(n)$	Known process noise variance

Now that an estimate of $Y(n)$ is known, the system state can be estimated using:

$$\hat{X}(n|n-1) = a(n) \cdot \hat{Y}(n|n-1) \tag{A.17}$$

Where:

$\hat{X}(n\|n-1)$	LMMSE estimate of X(n) based on $X(1), \cdots, X(n)$
$a(n)$	Known process noise coefficient

Update step

Now that the kalman filter has an estimate of the current state $\hat{X}(n)$, the next step is to take a measurement and find the error between the estimation and the observation. The estimation of $\hat{Y}(n+1)$ is updated using:

$$\hat{Y}(n|n) = \hat{Y}(n|n-1) + b(n) \cdot (X(n) - \hat{X}(n|n-1)) \tag{A.18}$$

Where:

$b(n)$	Kalman gain
$X(n)$	Observed value at time n

As seem in Equation A.18, it includes a $b(n)$ term called the kalman gain that is given as:

$$b(n) = \frac{a(n) \cdot R(n|n-1)}{a(n)^2 \cdot R(n|n-1) + \sigma_W^2(n)} \tag{A.19}$$

Where:

$b(n)$	Kalman gain
$R(n\|n-1)$	MSE between Y(n + 1) and $\hat{Y}$(n + 1n)
σ_W^2	Measurement noise variance

The kalman gain helps put emphasis on either the most recent measurements or can be set to rely on many past measurements.

Finally, the MSE of the estimated $\hat{Y}(n|n)$ is given as:

$$R(n|n) = (1 - b(n) \cdot a(n)) \cdot R(n|n-1) \tag{A.20}$$

Using these prediction and updating steps, the kalman filter can adaptively estimate even on noisy signals. To initialize the kalman filter $\hat{Y}(0|0) = \mu_{Y(0)}$, $R(0|0) = \sigma_{Y(0)}^2$, and $a(n)$ and $h(n)$ are known values set at initialization.

B. Room and Setup locations

This appendix contains the measurement setups used to obtain the datasets, Original (OG), High Frequency Lab (HF) and (Biggest Chamber (BC)). This appendix contains the measurement setups used for the HF and Biggest Chamber BC dataset that was not previously described in section 3.

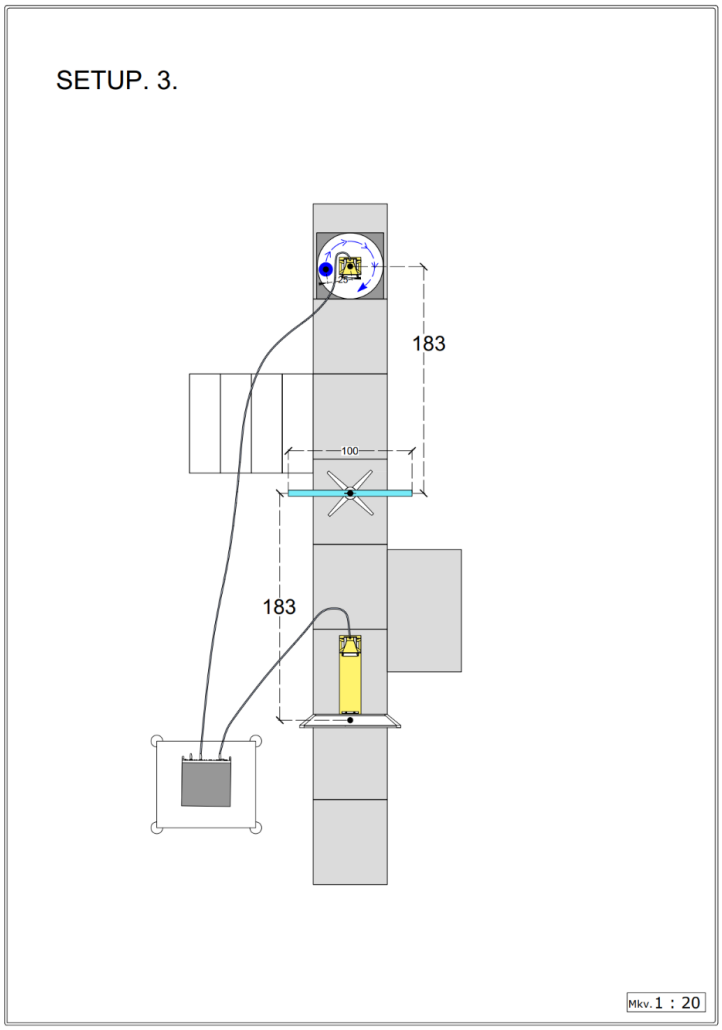


Figure B.1: Biggest Chamber room setup and dimensions.

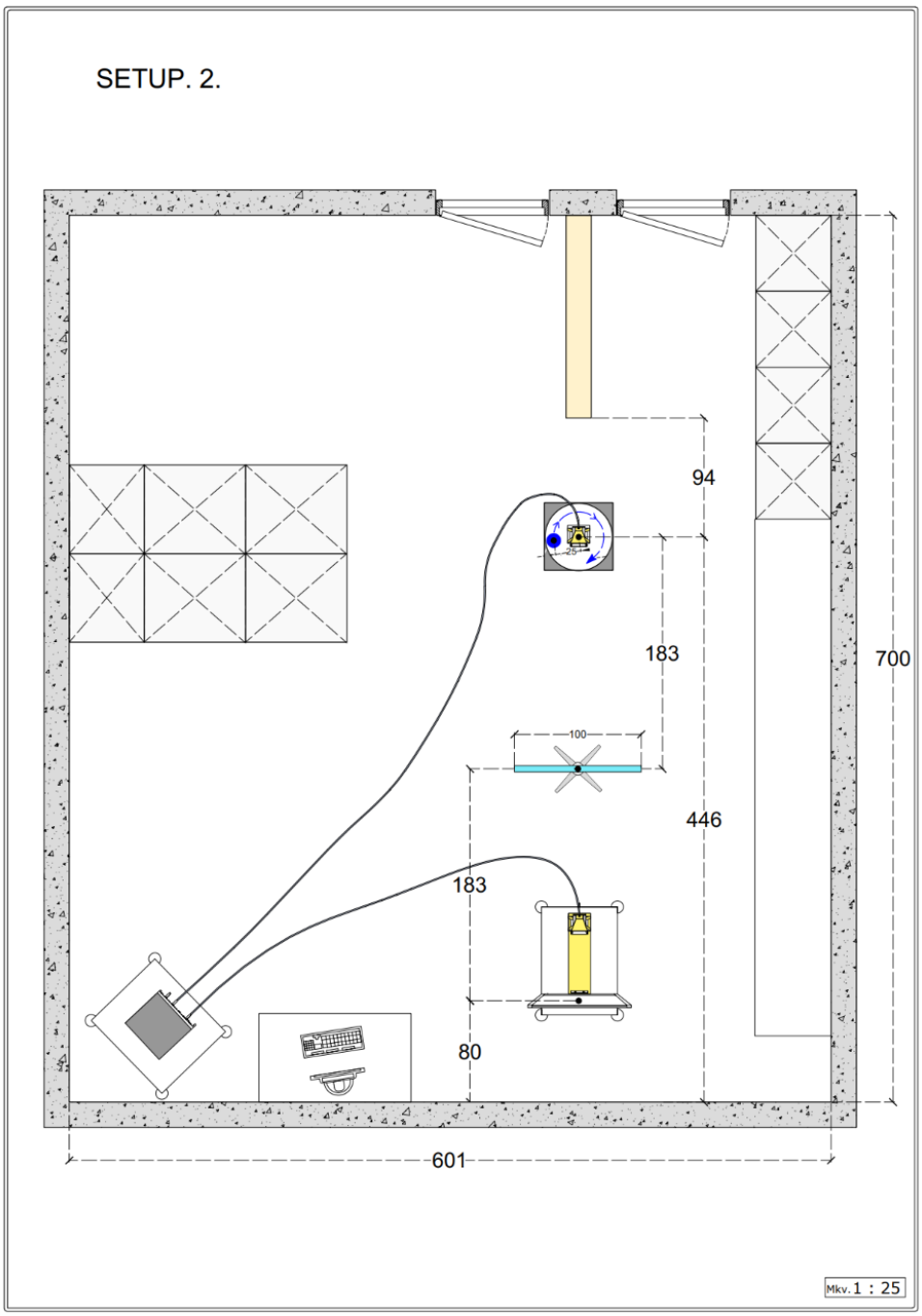


Figure B.2: Room setup and dimension for High Frequency Lab.

B.1 Test description for validating localization methods and RIS delay

Purpose
The purpose of this test is to validate the localization methods described in section 8. The test will provide data in both line of sight (LOS) and none line of sight (NLOS) cases, where a big metal plate is used for NLOS. By validating the methods, the research question in section 4 related to localization can be answered.

Furthermore, if the RIS adds a delay to the transmitted signal, it can influence the prediction of the methods. Therefore, the RIS's impact on the signal is important to discover to make the predictions more accurate.

List of equipment

Equipment	Manufacturer	Type	Specification	AAU no.
Automatic calibration device	Rohde & Schwarz	ZN-Z54	Range: 9 kHz to 40 GHz	128556
Vector Network Analyzer	Rohde & Schwarz	ZNA43	Range: 10 MHz to 43.5 GHz	128548
Reconfigurable intelligent surface(RIS)	APMS	—	Range: 400 MHz to 10 GHz	128601

Table B.1: List of equipment that was used in test.

Beyond this, the setup included standard RF equipment, including two horn antennas from AMPS and associated cabling.

The settings for the ZNA is:

ZNA Settings:	
Frequency	4.8-5.2 GHz
Ref value	0 dB
Num of points	1801
Time domain window	0-240 ns

Table B.2: The setting for the ZNA used for the measurements.

The setup
The transmitter (Tx) and receiver (Rx) is located as described in figure B.4, where the Tx and Rx have **X-length** between each other. The Tx antenna is pointing directly at the RIS and away from the Rx, as illustration in figure B.3. Halfway in between Tx and Rx a metal plate will be positioned, to simulate the NLOS scenario. For each **X-length**, the test is conducted in LOS and NLOS by either removing or placing the metal plate in the middel, respectably.

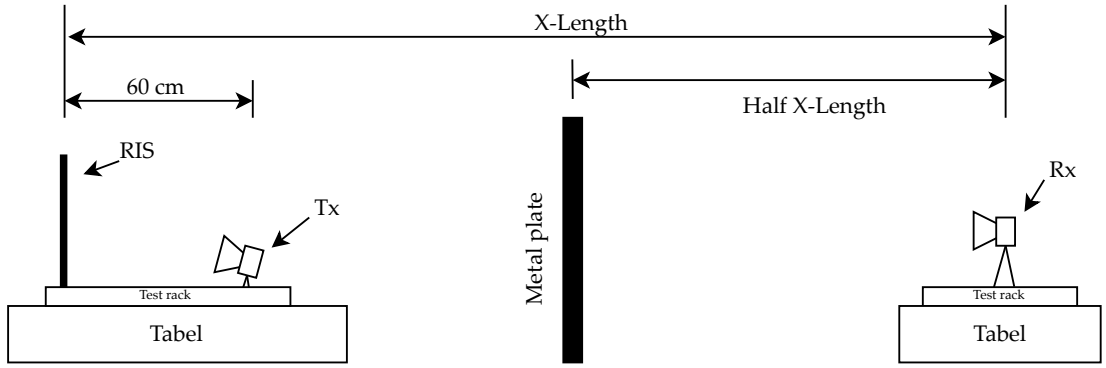


Figure B.3: The test setup as seen from the side.

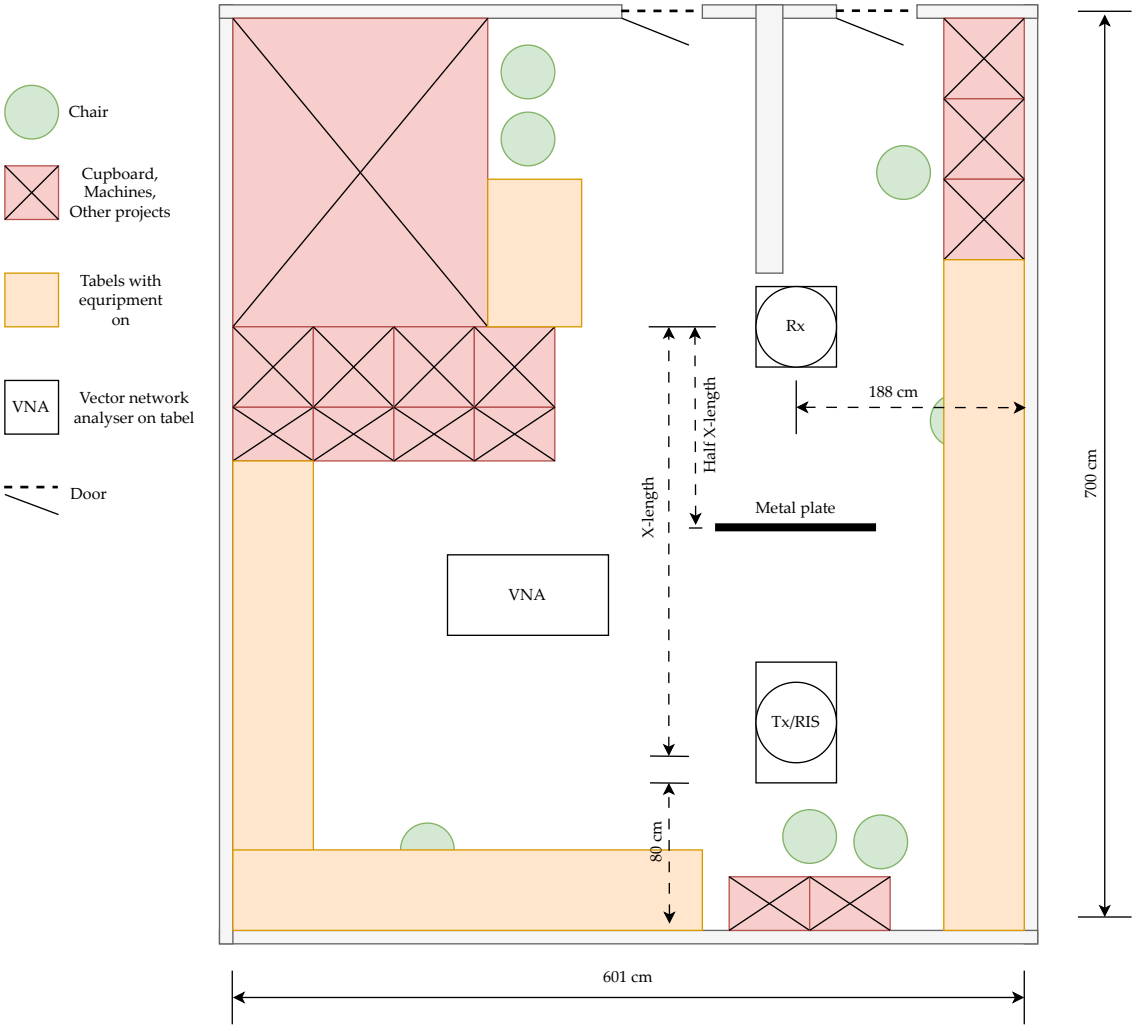


Figure B.4: Configuration of the test setup for testing the distance determination methods. The table with Tx and RIS is kept at a fixed position from the back wall, only the Rx changes position throughout the test.

The setup described in figure B.4 is for the high frequency lab. For the NLOS scenario the "Big metal plate" (1m x 1m) will be used. This metal plate is the same as the one used in the original

test, which is described in section 3.
A picture of the test setup can be seen on Figure B.5.

Figure B.5: Picture of the alignment process for the testsetup at a lenght of 3.96 meters. The antenna is aligned without a blocking object.

Procedure

The test will be conducted for the different **X-length** distances, as described in table B.3, with both LOS and NLOS scenarios. The results for the measured RIS delay are noted in the table, after the test has been performed.

1. Turn on and preheat the ZNA up to 30 minutes.
2. Calibrate the ZNA with the settings in Table B.2, except time domain window, with extended cables using the ZN-Z54.
3. Enter time domain and insert settings for time domain window.
4. Position Tx and Rx at **X-length** distance, as described in table B.3.
5. Use a laser to position Tx and Rx directly in front of each other in LOS scenario.
6. Transmit a pulse, when the RIS in turned on and turned off, and save the data. Note result in table B.3.
7. Sweep the RIS from -30 to 30 degrees with a 10 degrees step interval and make a transmission for each step for the given distance.
8. Save the data from the received data for each step of the sweep.
9. Redo step 4 and 5, but for the NLOS scenario.
10. Redo test from step 1, but with the next distance described in table B.3.

For each **X-length**, the RIS will sweep through its angels from -30 to 30 with a 10 degree step, each method will then estimate a distance for angle and distance. The obtained data must be defined as time[s], magnitude[dB], phase[deg]. Note that Rx and Tx should have equal

lengths from one of the wall, in this case utilize 1.88 meters (as seen in figure B.4), and that the **X-length** is from Rx to the RIS.

Test No.:	Distances[m]	Delay LOS [s]	Delay NLOS[s]
1	2		
2	3		
3	3.96		

Table B.3: Table for the X-length distances, and the corresponding calculated distance in LOS and NLOS scenarios, with the delay the RIS at each distance and scenario.

Results

The data for the phase of the signal was invalid and cannot be used in finding the distance using the group delay. Therefore. the following is only for the trigonometry method, described in section 8.3. Furthermore, the data for 0 degrees RIS angle in the test for 2 meters LOS was also invalid.

All calculations for the trigonometry method, have used a window of $N = 25$ samples which was discarded as possible signals first arrival, prior to the max dB found. The threshold used was 6 dB.

The delay from the RIS is calculated by subtracting the time between the two maximum peaks from the same scenarios with the RIS turned on and off. The comparsion for the LOS senario at 3.96 m is shown in figure B.6, and for NLOS at 3.96 m in figure B.7.

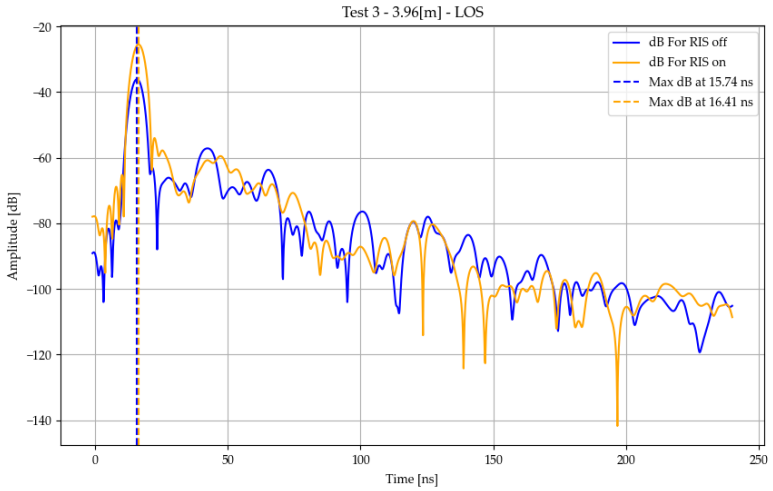


Figure B.6: The received signal in dB for the LOS case with a 3.96 m distance. The delay causing by RIS is 0.6694 ns for this case.

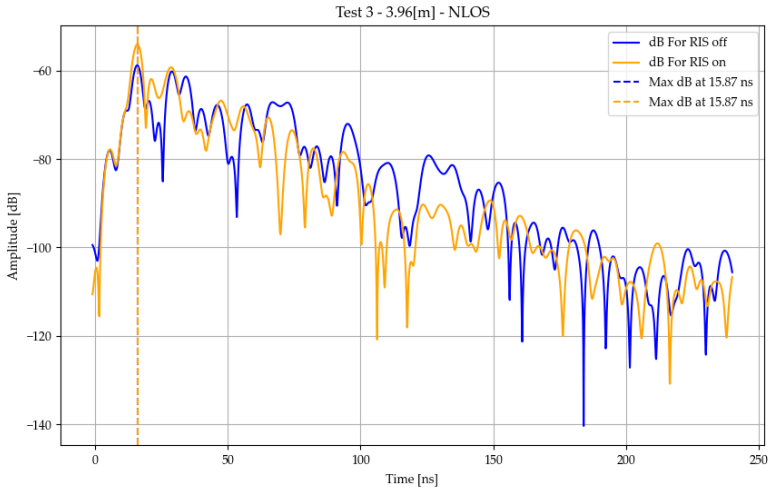


Figure B.7: The received signal in dB for the NLOS case with a 3.96 m distance. The delay causing by RIS is 0 ns for this case.

Calculations for the distance the transmitted signal traveled was estimated as

$$\text{Distance} = \text{TOF} - 0.6 \cdot c - \mathit{Delay} \tag{B.1}$$

where

$\mathit{Distance}$	The distance the signal has traveled
TOF	The time for the signal to come from Tx to Rx
c	Speed of light
0.6	Distance from Tx to RIS [m]
Delay	The delay of the RIS as seen in table B.4

The results of the RIS delay is calculated and inserted in table B.4.

Test No.:	Distances[m]	Delay LOS[s]	Delay NLOS[s]
1	2	-4.016e-10	-1.866e-09
2	3	-6.694e-10	-7.999e-10
3	3.96	-6.694e-10	-0.0

Table B.4: Table for the **X-length** distances, and the corresponding calculated distance in LOS and NLOS scenarios, with the delay the RIS at each distance and scenario.

The estimated distances will be plotted compared to the RIS angles. Results of the first test at 2 meter distance between the Rx antenna and the RIS for LOS can be seen in figure B.8 and for NLOS in figure B.9.

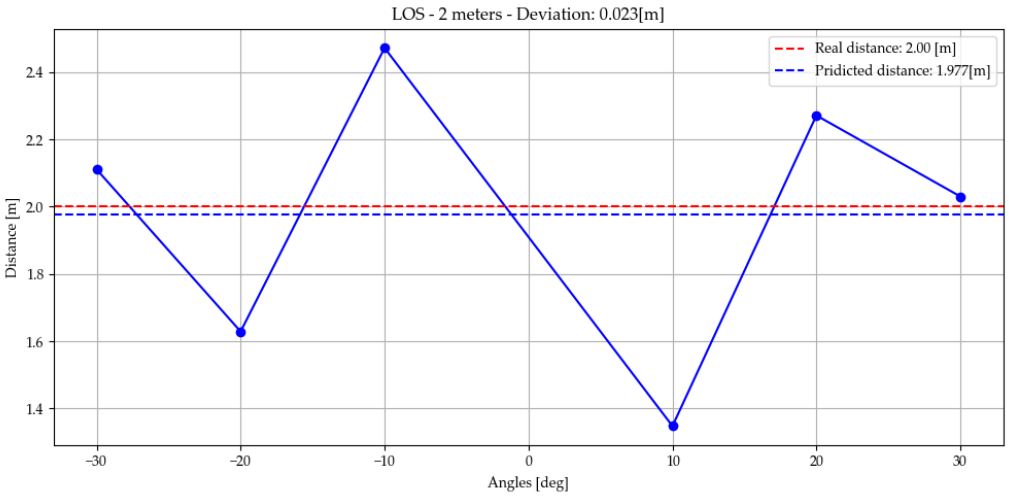


Figure B.8: The calculated distances for LOS at 2 meter, given the different RIS angles (-30 to 30°), and the maximum used amplitude in dB for each angel.

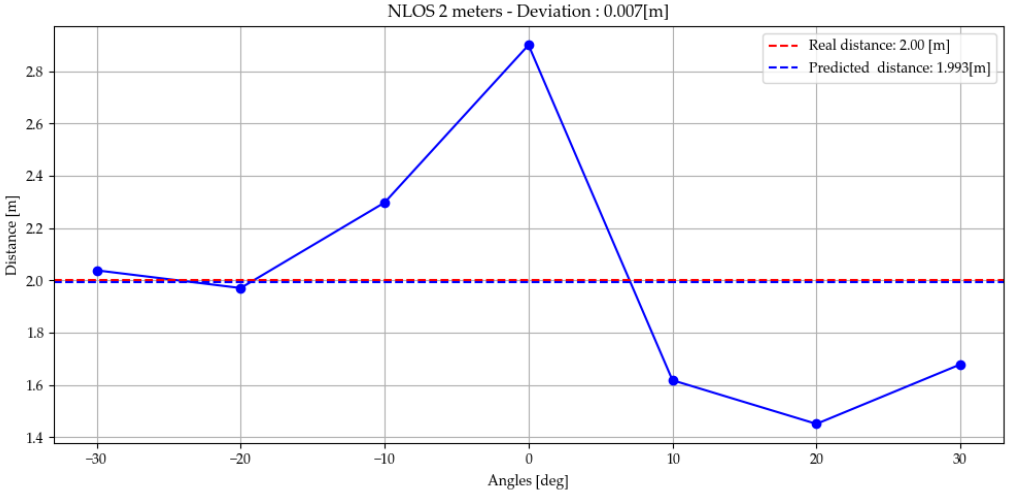


Figure B.9: The calculated distances for NLOS at 2 meter, given the different RIS angles (-30 to 30°), and the maximum used amplitude in dB for each angel.

Results of the first test at a 3 meter distance between the Rx antenna and the RIS for LOS can be seen in figure B.10 and for NLOS in figure B.11.

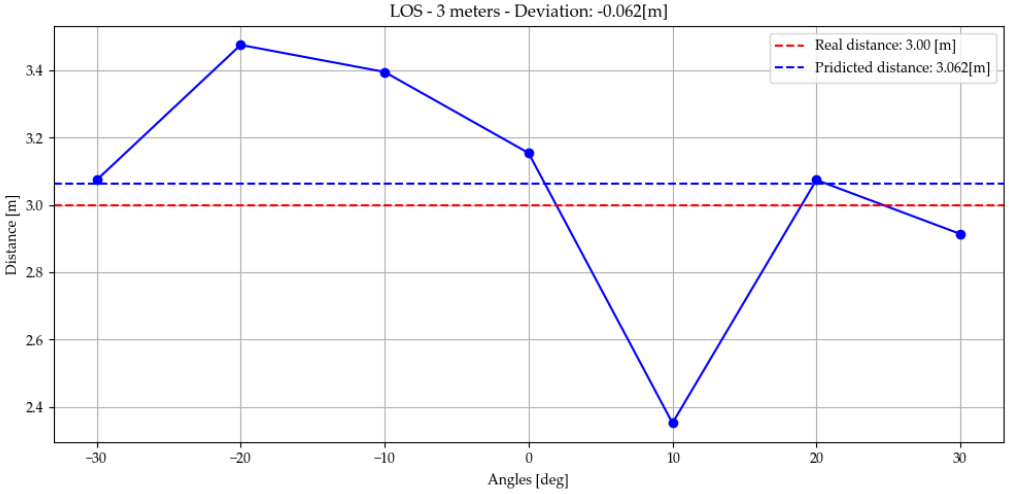


Figure B.10: The calculated distances for LOS at 3 meters, given the different RIS angles (-30 to 30°), and the maximum used dB for each angel.

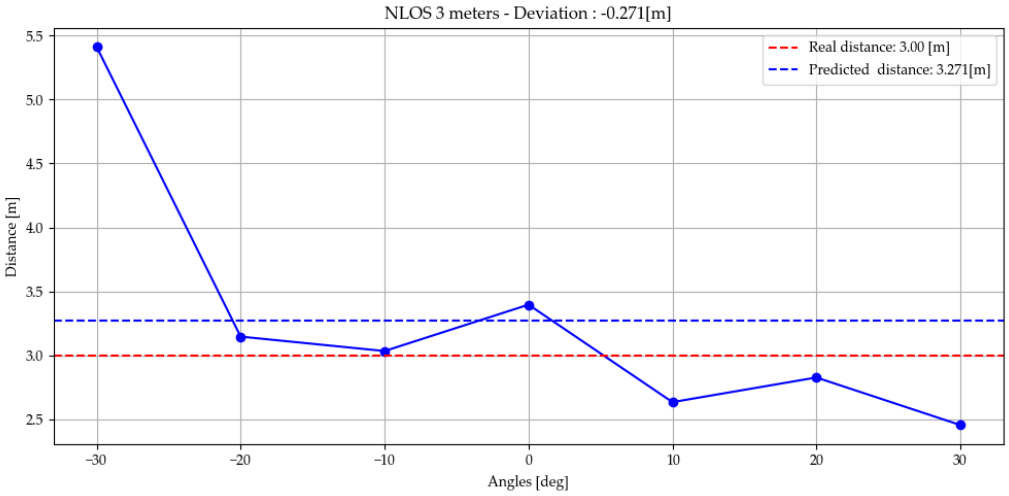


Figure B.11: The calculated distances for NLOS at 3 meters, given the different RIS angles (-30 to 30°), and the maximum used dB for each angel.

Results of the first test at 3.96 meters distance between the Rx antenna and the RIS for LOS can be seen in figure B.12 and for NLOS in figure B.13.

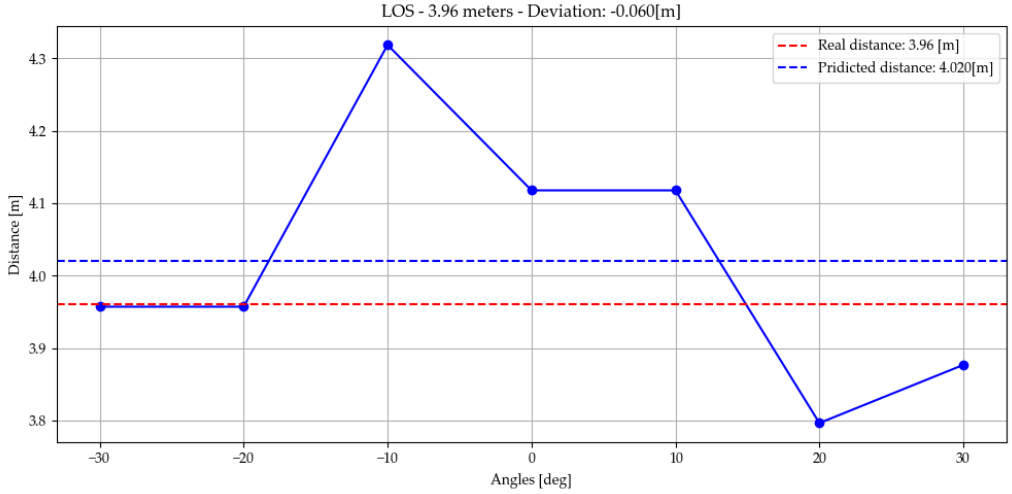


Figure B.12: The calculated distances for LOS at 3.96 meters, given the different RIS angles (-30 to 30°), and the maximum used dB for each angel.

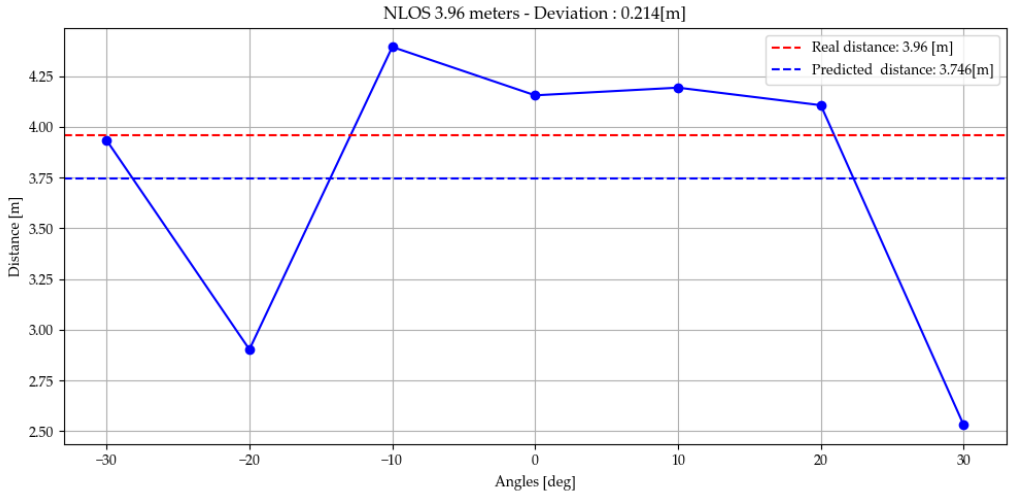


Figure B.13: The calculated distances for NLOS at 3.96 meters, given the different RIS angles (-30 to 30°), and the maximum used dB for each angel.

By combining the test results and the means of these, it can be seen that the maximum predicted distance deviation was in NLOS at 3 meters, where the deviation was 27.1 cm. The minimum predicted direct distance came out to be 0.7 cm in the LOS 2 meters scenario. By taking the average NLOS predicted distance deviation for all distances combined it results in

$$\frac{2.3\text{cm} + 6.2\text{cm} + 6\text{cm}}{3} = 4.83\text{cm} \tag{B.2}$$

and for LOS it will result in

$$\frac{0.7\text{cm} + 27.1\text{cm} + 21.4\text{cm}}{3} = 16.4\text{cm} \tag{B.3}$$

Lastly the total average predicted distance dilation for both LOS and NLOS is

$$\frac{4.83\text{cm} + 16.4\text{cm}}{2} = 10.61\text{cm} \tag{B.4}$$

Discussion

As seen in the **results** section, the calculated distance between Rx and the RIS is more precise than with the data from the the 3, as the error in B.3 range from 0.7 cm to 27.1 cm, while the errors in 8.1 range from 19 cm to 38 cm. The reasons for the errors can be manifold, as different RIS delay times, lobes of propagating signal, assumptions of the environment, etc.

The RIS might add different delay times depending on the angle of reflecting, and as the RIS delay was only calculated and measured at 0° angle, it would either positively or negatively effect the distances calculated at other angles. This would to some extend explain the behavior of the results.

Furthermore, if the RIS has a propagating beam, which at different angles (-30 to $30°$) would make a zero spot be propagated towards the Rx, it would negatively effect the calculated distance, as the strongest signal might be from a longer path than LOS due to reflections. This would lead the calculations to predict a distance further away than in reality.

Distances predicted to be closer than in reality could be due to the way the maximum magnitude was found. Because if the signal received was slower than then $N = 25$ samples to reach its maximum magnitude within the threshold, the calculations would predict the signal to be received earlier.

Lastly, it is important to state that the exact environment for the test cannot be replicated, as the environment was not controlled. This is because the high frequency room is a place students/researchers can enter and use, which lead to equipment, projects, wires etc. to be placed randomly on the floor and tables. Therefore, any replication of the test will lead to different results.

Conclusion

It can be concluded that, the trigonometry method described in section 8 can, with a certain degree of error, predict the direct distance between two points, given only the time of arrival, a sweep of the RIS angles, and the distance from the wall to the RIS. The method have potential to be used to get an estimation of the direct distance. However, the test is not a generalized setup and is very case specific, therefore it is not representative for cases where Tx and Rx does not have the same distance to the walls, and are not lined up. Therefore, more tests with different positioning and angels would be needed, to make a more specific conclusion about the trigonometry method and its use-cases. Furthermore, the group delay calculation could not be estimated due to data being saved wrongly. Therefore the group delay remains inconclusive.

C. All test results and Code for project

C.1 Test results

A folder with all test results can be seen in the attached folder: *Model Results with Kurtosis and RMS_Delay.zip*. Here the validation/training graph, confusion matrix, SNR loop, general confusion matrix, general SNR loop, accuracy for each SNR step for both SNR loops are saved. Both a jit model and a saved model are saved for each test. The folder also contains summary accuracy for the SNR loop and Accuracy for each model trained.

C.2 Code for project

The code for the project can be found on github

`https://github.com/P7-6G-NLOS/experimental-models`

or in the attached zip folder named *Project Code.zip*

D. Preprocessing results

To determine the models performance it is required to determine each preprocessing methods result on the model. Knowing the results between the actual model and application of preprocessing methods can help assist and potentially combine preprocessing methods to achieve the best model.

D.1 No preprocessing

The baseline for comparisons is necessary to see the impact of the methods. To compare each model, it is required to have knowledge about the model's performance. In this section the confusion matrix, accuracy, SNR graph and average accuracy for all SNR levels will all be determined. The generalization of the model will also be compared, using the accuracy for each dataset for both the confusion matrix as well as the average SNR. The baseline model trained on only the original (OG) data had an accuracy of 91.00% on the test set, an average SNR loop accuracy of 77.89%, with the graph seen in figure D.1.

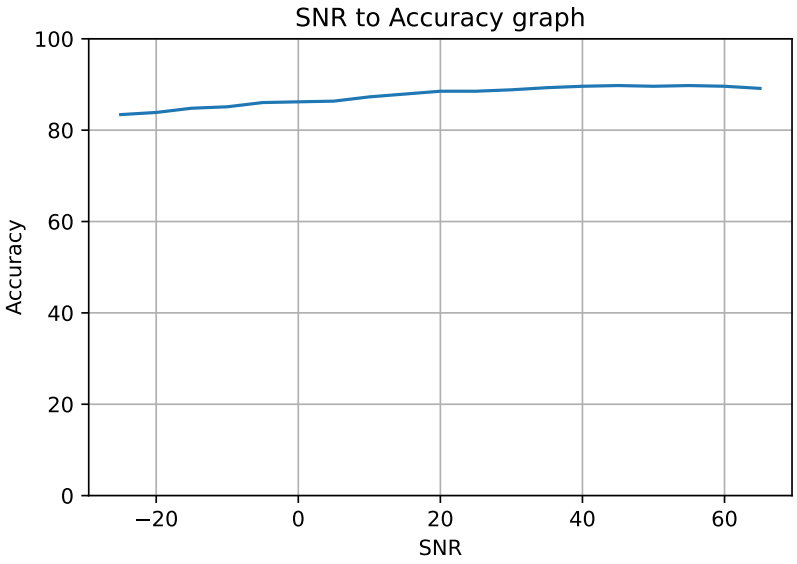


Figure D.1: SNR graph for baseline model.

For the generalization test, the model trained only on OG had an accuracy of 47.09% on HF data, and 100% on the BC data. The average SNR loop accuracy for the High Frequency Lab (HF) data is 51.78% and 98.53% for the Biggest Chamber (BC) data. The figure can be seen in

figure D.2.

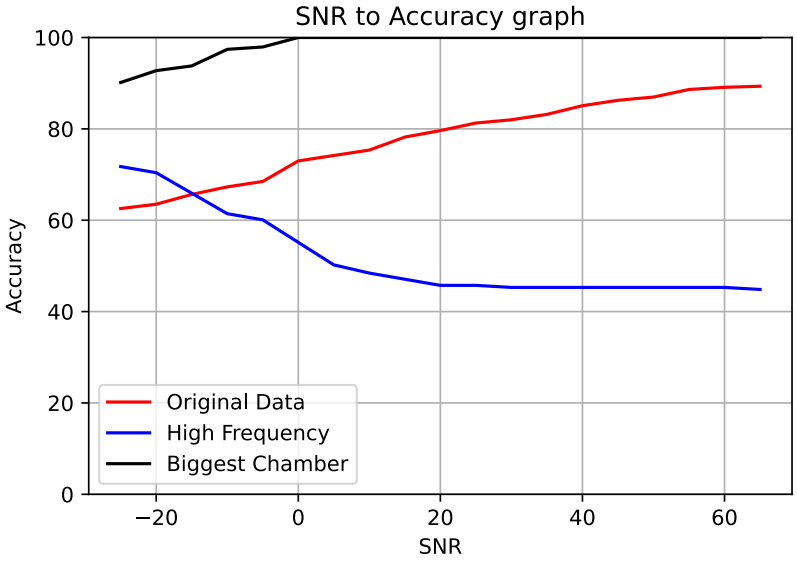


Figure D.2: General SNR test result for Default case.

D.2 Noise

Applying noise was done in two ways, with both an SNR of 0 dB for all datasets (OG, HF, BC) used for both training and testing accuracy, and applying a range of SNR in a random amount for each file in the dataset.

D.2.1 SNR 0

Adding an SNR of 0 dB to all the datasets did not seem to affect the models' accuracy in the CM test, having the same accuracy of 91.00%, while increasing its average accuracy in the SNR test by 10%, going from 77.89% to 87.63%, which is a significant increase. The SNR graph can be seen in Figure D.3.

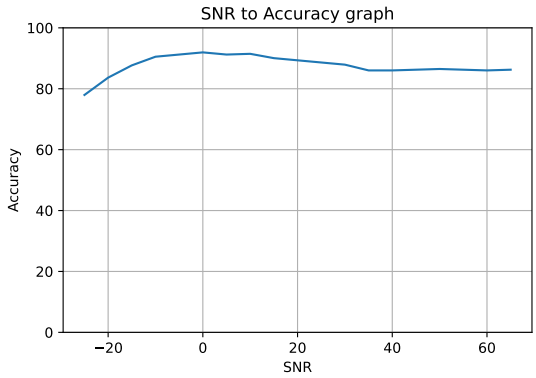


Figure D.3: SNR graph results for SNR 0 dB applied to the training data.

Here it can be seen that although the average accuracy is a lot better it has a "bump" before it

decreases in accuracy, with a peak around an SNR of 0, which makes sense since this is what the model was trained on. Although the results for the OG test set is improved, the general test does not seem to improve for the BC and HF test results, which can be seen in Figure D.4.

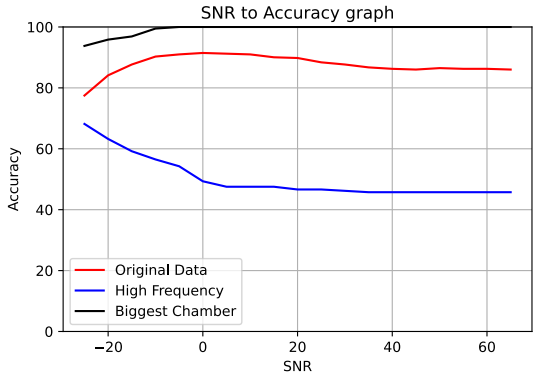


Figure D.4: General SNR graph results for SNR 0 dB applied to the training data.

D.2.2 Random SNR

For the random SNR, the accuracy was almost the same, being at 90.76%, but it has a higher accuracy on the average SNR case of 91.00%. This is likely due to the model being trained with multiple different SNR levels, causing it to be extremely resilient to changes in the SNR. For comparison, the graph can be seen in Figure D.5, where it can be seen that it is a consistently flat line across all SNR levels.

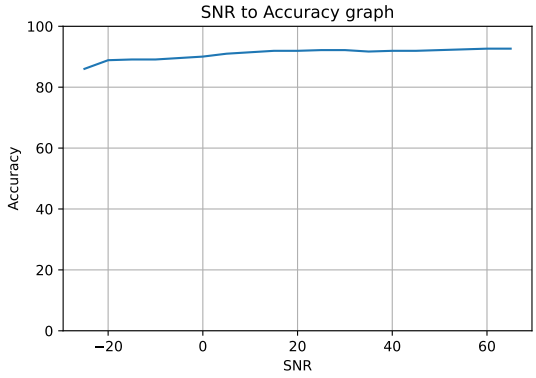


Figure D.5: The SNR graph results for a model trained on a random SNR.

Once again, the general SNR loop function sees no change for the two other datasets, meaning that adding noise to the data does not improve the generality of the function when used in other environments. The noise function does improve performance over different noise levels though, and quite a lot, with the random SNR increasing the average accuracy over the SNR graph by 12.37% seen in Table 7.1. For this reason, as both the random and SNR 0 performs about the same in the general test, but with the random SNR outperforming the SNR 0 in the SNR graph test, it is chosen as the continued preprocessing method for appliying noise.

D.3 Filter

After applying only the kalman filter the base accuracy for the model increases by 4% on the test set, having an accuracy of 94.31% seen on Table 7.1, and increases the SNR by 5% as well, being at 82% on the SNR loop compared to the baseline model. In terms of the performance over different datasets, the kalman filter had the same impact increasing the performance by around 2 percent.

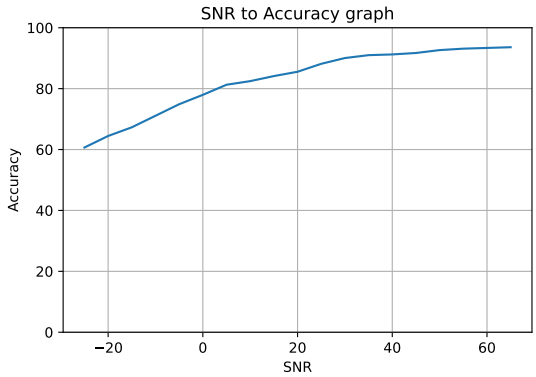


Figure D.6: SNR graph results with only Kalman Filtering

As the use of a Kalman filter is intended to reduce the importance of noise in a signal, the kalman filter was also applied to both SNR function to see the performance in tandem. The SNR 0 function's accuracy increased by 2% when also using the kalman function as seen on Table D.1, in both the accurcy for the test set, as well as the SNR test. For the random noise function, there was no increase in performance, and it performed about the same as without the kalman filter being applied seen on Figure D.8. As this is the case, the filter is applied to the SNR functions, due to the increase in performance of the SNR function seen in Figure D.7.

Modification.	SNR acc.	OG data acc.	HF data acc.	BC data acc.
Filter+SNR 0	89.66	93.38%	52.47%	99.48
Filter+ SNR random	89.87%	91.15%	53.36%	99.48

Table D.1: Table over preprocessing for DNN Filter + SNR 0 and Filter + SNR random accuracies with a DNN trained with no data augmentation.

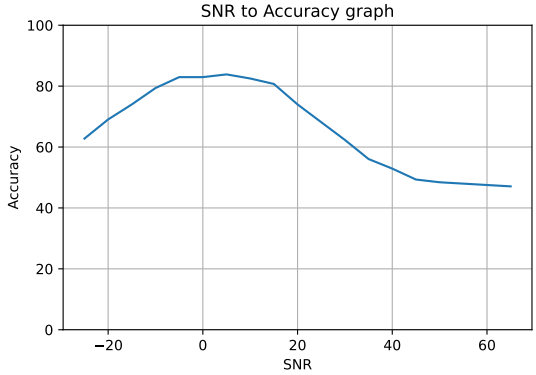


Figure D.7: SNR graph results with Filter + SNR 0

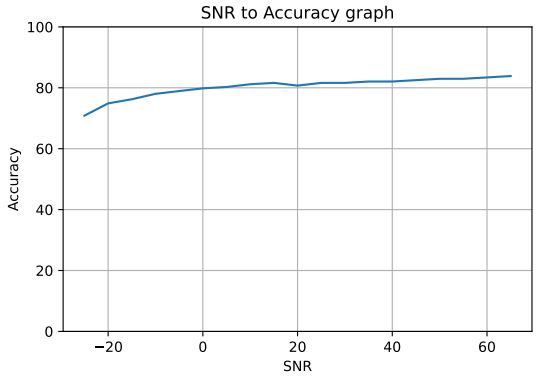


Figure D.8: SNR graph results with Filter + SNR random

D.4 Group delay

Applying group delay to the model resulted in an increase in accuracy, going from the 91.00% for the base model to 94.31% for the test data on the new model, and an average SNR accuracy of 83.42%, which is also a significant increase as seen on Table 7.1. This increase does not make it that accurate for noise alterations though, as it still follows the same curve as the baseline dataset, having a bad accuracy at low SNR, as seen on Figure D.9.

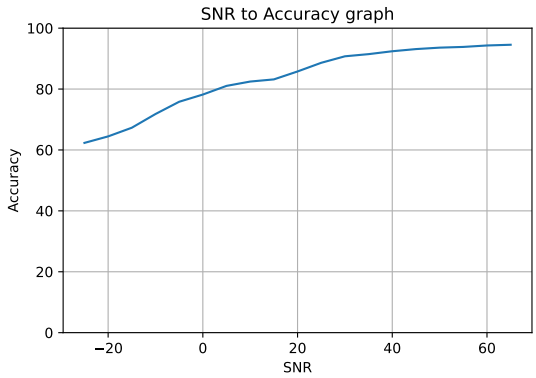


Figure D.9: SNR group results with only group delay applied.

Although the group delay increases the accuracy of the model on its own test data, it does still not improve the performance in other datasets.

D.5 FFT

The use of the FFT function increases the model performance in a similar way to both the group delay and kalman filter. The FFT only model has a base accuracy of 94.55%, and an average SNR accuracy of 83.59% from Table 7.1. Here it once again follows the curve of the original dataset, having a poor performance at low SNR as seen on Figure D.10. In terms of the general performance of the model, it does not increase in performance when compared to the baseline data.

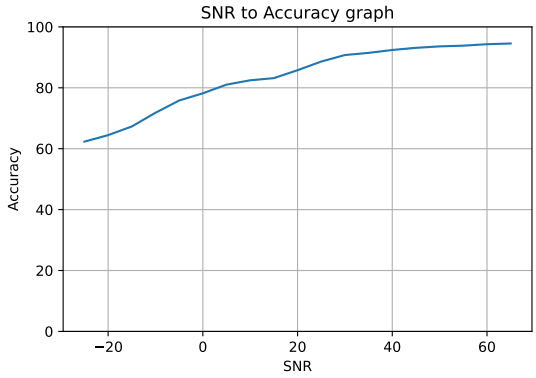


Figure D.10: SNR group results with only FFT applied.

D.6 Normalization

For applying normalization to the data, two methods can be used, although only one at a time. Therefore, the best performing method will be chosen, determined by the base accuracy and SNR average accuracy, as well as the SNR plot itself.

D.6.1 Standard normalization

First up, the standard normalization has a base accuracy of 91.94%, being a 0.94% increase relative to the base model, and an average SNR accuracy of 91.94% as well seen on Table 7.1. This accuracy is about the same as training the model with noise applied, but the plot seen in Figure D.11 represents a straight line of the accuracy. This is quite peculiar, as it is unrealistic for the model to have the same accuracy over all SNR levels.

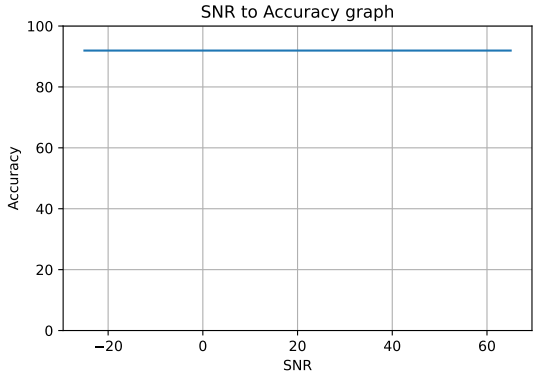


Figure D.11: SNR plot results with only normalization applied.

In case of the general tests, the model still did not perform well on the other datasets, with a decrease in performance of the BC test data when compared to the baseline by 10.34%, which is likely due to its constant accuracy being quite low.

D.6.2 Z-score Normalization

The Z-score normalization achieved slightly better results than the standard normalization function, having a base accuracy of 92.87% and an SNR accuracy of 90.33% from Table 7.1, but with

the graph behaving in a more expected manner. The plot can be seen in Figure D.12, where it can be seen it follows the same behavior as when applying the random SNR function to the data.

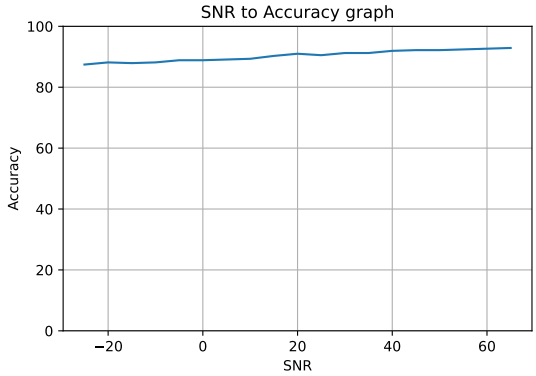


Figure D.12: SNR graph results with only Z-score normalization applied.

When using the Z-score normalization for the general tests, it also decreases to 45.29% to the performance when compared to the baseline for HF. As only one of these methods can be applied at a time, the Z-score normalization is chosen, due to more realistic results for the SNR graph, and with both having about the same performance, it is more safe to choose the Z-score method.

D.7 Kurtosis

Applying kurtosis results in the greatest accuracy achieved for the base test at 95.26%, while still achieving a 6% increase in the SNR test at 83.74% average accuracy as seen on Table 7.1. The curve here once again follows the baseline, showing that kurtosis does not provide robustness for change in noise levels seen on Figure D.13, when compared to the SNR or Normalization methods. It does however increase the performance significantly in the specific environment in which it is trained, which could lead to improvements in accuracy when used in tandem with methods to handle noise accuracy.

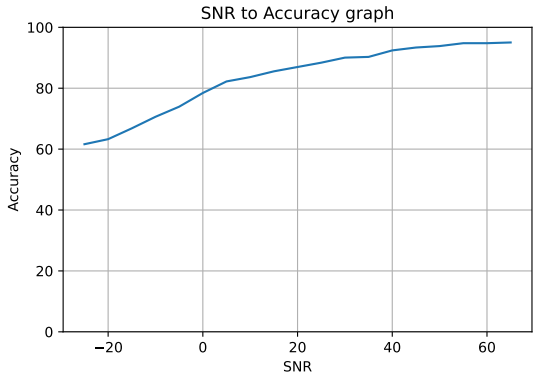


Figure D.13: SNR graph results with only kurtosis applied.

D.8 RMS Delay

Using the rms Delay function results in an increase in performance as well, having an accuracy of 94.55%, being the second highest base accuracy of the preprocessing methods. For the SNR test results, the RMS delay results in an increase as well at 83.47% seen on Table 7.1, although once again not on par with the methods developed for adapting to changes in the noise level. The RMS delay does not increase the performance for the general tests either, following the same results as the baseline for the model with no preprocessing applied seen on Figure D.14.

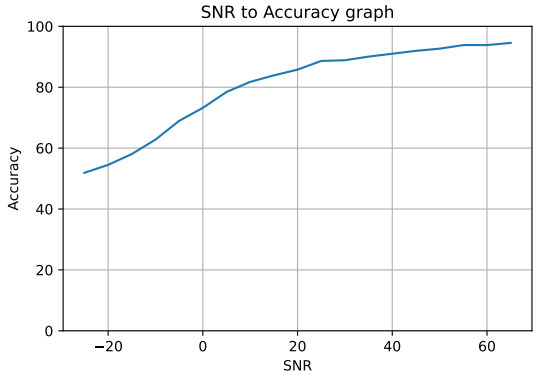


Figure D.14: SNR graph results with only RMS delay applied.

D.9 Transfer Learning

Finally, the transfer learning method is tested by seeing which dataset performs the best, being either the original data, the HF data or a combined dataset between the two as domain adaption transfer learning seen on Table 7.2.

As the original data was used for setting the baseline for the results of the preprocessing methods, the same baseline can be used for comparing which dataset to use for training the model.

D.9.1 High Frequency lab data

When using only HF data for training the model, the accuracy of the HF model falls to 87.00% from the baseline model 91.00% accuracy. For the SNR test as well, the accuracy of the HF model is reduced to 64.79% compared to the 77.89% of the baseline model. The HF SNR graph also follows quite a different curve than the original dataset seen in figure D.1, with the HF graph seen in figure D.15.

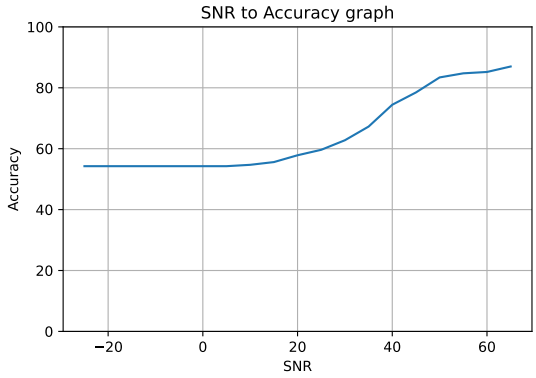


Figure D.15: HF SNR test results with no preprocessing applied.

When using the model for other datasets as in the general model test, the performance is once again very poor, with the base accuracy test having an accuracy of 38.39%, 67% and 100% for the original, HF and BC data, respectivaly. The SNR test showed poor performance as well, with the SNR test seen on figure D.16.

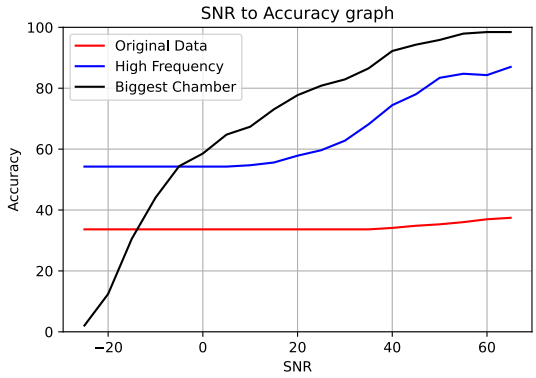


Figure D.16: General SNR graph of HF model with no preprocessing.

D.9.2 Domain adaption transfer learning

For applying domain adaption transfer learning, both datasets are mixed, allowing the model to adapt to both environments while training. Using this method was highly effective for training the model, as it achieved a base accuracy of 93.49%, although a slightly lower average SNR accuracy of 75.87% when compared to the 77.89% of the model trained only on the original data. Although this is the case, it is quickly improved when comparing the generalized test results, as an improvement here can be seen clearly, with an accuracy of around 94% for all datasets for the base test, and the SNR test having an average accuracy of around 80% for all datasets as well. The graph can be seen in Figure D.17. When comparing the transductive transfer learning model to the other models, it can clearly be seen that it increases the generality of the model a lot, letting it be effective in other environments. For this reason, the mixed model is chosen as the preferred dataset, as it can be used in all provided scenarios.

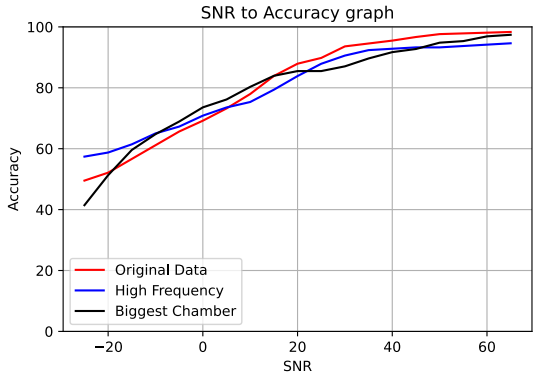


Figure D.17: General SNR test perforamed on mixed dataset with no additional preprocessing.

E. Model test

These sections describe the test results for all the different machine learning models, which are designed in section 5.

E.1 Deep neural network test results

This section contains the results for different combinations of preprocessing for the test cases described in Section 7.1. The table E.1 shows the confusion matrix accuracies for different combinations of preprocessing methods. The preprocessing method is shown in column 2 named modification. In row No. 1, the dataset which the machine learning model was trained on, can be seen, e.g. CM OG ("Confusion matrix Original data") means that the model was trained on original data. Furthermore, if OG or HF is not defined in the title, the model is trained, validated, and tested on a split of the OG dataset. When noise is added to the data, the filter will also remain present and utilized for the entire time. The rest of the tests are a combination of the different methods and their results.

		CM Accuracy			SNR Loop Average			CM OG			CM HF			CM OG+HF			SNR Loop Average OG			SNR Loop Average HF			SNR Loop Average OG + HF		
	Modification	OG	HF	Both	OG	HF	Both	OG	HF	BC	OG	HF	BC	OG	HF	BC	OG	HF	BC	OG	HF	BC	OG	HF	BC
0	Default	91.00	87.00	93.49	77.89	64.79	75.87	93.60	47.09	100.00	38.39	86.55	100.00	99.05	94.17	99.48	77.87	51.78	98.53	34.32	64.76	69.08	81.02	80.29	79.82
1	SNR0	91.00	82.96	88.84	87.63	66.56	75.76	92.89	53.36	100.00	43.60	79.82	95.85	98.10	93.27	94.82	87.59	50.15	99.26	58.87	66.44	94.49	88.45	75.17	94.08
2	SNR Random	90.76	82.96	87.44	91.00	81.35	86.63	92.42	49.78	100.00	64.69	81.17	98.96	97.87	89.69	94.82	91.01	47.60	99.43	63.34	81.24	94.93	97.51	92.92	93.18
3	Filter	94.31	86.10	91.16	82.88	67.24	79.74	95.97	46.64	100.00	47.16	84.30	100.00	98.58	94.62	100.00	82.99	53.29	98.15	35.93	67.31	74.97	82.65	82.98	85.49
4	GroupDelay	94.31	90.13	87.91	83.42	66.44	74.89	96.21	44.84	100.00	41.23	88.34	100.00	99.29	88.79	95.85	83.46	56.81	95.91	34.45	66.34	68.72	85.87	66.79	72.65
5	FFT	94.55	89.24	92.40	83.59	65.33	70.88	95.26	45.74	100.00	41.94	87.00	100.00	99.53	95.07	98.96	83.56	54.78	97.03	34.55	65.33	66.76	74.48	73.24	74.48
6	Norm	91.94	73.99	81.40	91.94	74.11	81.40	91.00	47.98	89.64	41.00	73.54	36.79	96.21	87.00	69.43	91.94	45.74	87.56	38.86	74.11	37.82	96.21	91.93	72.02
7	ZNorm	92.89	75.34	86.20	90.33	74.04	83.94	94.08	45.29	93.78	55.45	73.99	31.61	98.10	87.44	75.65	90.27	41.47	93.89	53.33	74.09	33.95	98.23	91.01	79.17
8	Kurtosis	95.26	89.24	92.87	83.47	66.06	78.74	96.21	47.09	100.00	40.28	87.44	100.00	99.29	93.27	94.30	83.47	52.68	100.00	34.61	65.92	67.47	88.33	78.50	86.80
9	RMS	94.55	83.41	92.40	80.24	66.11	74.08	95.50	42.60	100.00	45.97	81.17	100.00	99.53	92.83	98.45	80.16	56.31	93.21	35.41	66.11	72.08	78.17	77.22	76.36
10	SNR0 + Filter	92.65	83.86	88.68	89.66	65.90	78.91	93.84	52.47	99.48	46.21	83.86	96.37	97.87	92.38	97.41	89.60	50.08	99.51	59.10	65.92	94.46	91.44	83.10	96.02
11	SNR Random + Filter	90.28	81.17	88.22	89.87	80.29	88.80	91.47	53.36	99.48	68.96	81.17	97.93	97.39	91.93	93.26	89.95	50.93	98.77	62.92	80.34	93.81	98.17	93.08	91.49
12	SNR Random + Filter + GroupDelay	92.42	81.61	88.22	92.19	80.46	87.56	92.89	48.43	100.00	67.77	78.92	96.37	96.92	91.93	97.93	92.34	47.44	99.75	63.84	80.48	93.73	97.02	92.40	95.45
13	SNR Random + Filter + FFT	89.10	82.51	86.98	89.41	82.46	86.94	90.05	47.53	100.00	68.01	80.27	99.48	99.05	90.58	91.19	89.36	46.99	99.84	65.22	82.44	95.31	97.52	90.37	88.08
14	SNR Random + Filter + ZNorm	91.23	74.44	84.96	90.97	74.09	85.03	91.47	40.81	88.08	37.68	72.20	10.36	98.10	88.34	57.51	90.86	42.67	87.26	35.93	74.11	12.30	97.68	92.49	61.39
15	SNR Random + Filter + Kurtosis	91.23	82.96	87.29	89.46	81.92	85.63	91.23	54.26	98.96	67.30	80.72	98.45	97.87	88.79	83.94	89.35	48.60	97.60	65.04	81.92	95.85	97.61	91.69	88.71
16	SNR Random + Filter + RMS	91.94	78.03	89.77	91.88	76.12	88.49	92.89	48.88	100.00	56.64	76.23	86.53	98.10	94.17	95.85	91.92	46.90	99.86	58.32	76.16	84.21	96.94	93.11	91.57
17	GroupDelay + FFT	95.50	86.10	92.87	82.10	66.27	77.31	95.97	45.74	100.00	43.84	86.55	100.00	99.53	94.62	99.48	81.94	54.47	97.19	34.97	66.32	72.81	85.33	74.89	77.94
18	GroupDelay + ZNorm	90.76	71.30	85.58	88.91	70.57	83.13	91.71	40.81	99.48	61.85	73.54	35.23	99.05	88.79	84.97	88.92	41.94	99.26	61.60	70.43	42.32	97.17	90.51	86.28
19	GroupDelay + Kurtosis	87.20	88.34	93.02	81.52	62.69	76.77	87.20	42.60	100.00	36.26	88.34	100.00	98.82	94.17	99.48	81.52	50.48	98.91	33.79	62.69	63.49	81.65	74.63	70.90
20	GroupDelay + RMS	94.08	88.34	91.63	86.93	64.41	71.91	95.26	46.19	100.00	38.15	87.89	100.00	99.29	93.72	100.00	86.85	50.39	99.35	34.17	64.43	65.91	74.77	73.16	74.39
21	FFT + Kurtosis	94.31	83.41	91.47	80.74	66.72	79.37	95.97	45.74	100.00	46.68	82.96	100.00	99.29	92.83	98.96	80.73	54.12	98.39	35.35	66.75	74.01	83.05	84.85	86.75
22	FFT + RMS	95.97	86.55	92.25	84.24	64.95	75.00	96.68	42.60	100.00	37.91	87.44	100.00	98.58	93.72	100.00	84.34	48.10	98.20	34.25	64.93	67.38	78.51	78.90	79.11
23	ZNorm + Kurtosis	91.23	78.03	84.81	89.22	78.07	83.41	92.42	39.46	93.78	59.24	78.03	8.81	97.63	88.79	70.47	89.25	41.40	94.11	59.94	78.12	6.27	97.41	90.96	71.78
24	ZNorm + RMS	92.42	79.82	86.51	90.96	78.88	82.87	93.60	36.32	94.82	63.51	78.03	66.84	98.58	90.13	88.60	90.88	41.75	94.96	64.59	78.57	71.15	97.51	93.11	90.40
25	Kurtosis + RMS	94.31	91.48	93.49	79.58	67.52	73.69	94.55	44.39	100.00	43.60	89.69	100.00	99.53	94.17	99.48	79.50	44.47	100.00	34.83	67.52	71.45	76.45	78.05	78.16
26	SNR Random + Filter + GroupDelay + FFT	91.47	81.17	87.75	91.33	80.98	86.40	92.65	48.88	100.00	66.59	81.61	95.34	98.10	89.24	94.82	91.34	47.01	99.86	64.24	81.07	93.29	96.77	91.83	94.08
27	SNR Random + Filter + GroupDelay + ZNorm	91.23	75.34	84.81	90.65	75.41	85.27	91.23	42.60	87.56	48.34	72.65	29.02	97.63	88.34	69.43	90.61	43.45	87.89	44.57	75.31	31.61	97.54	91.50	74.12
28	SNR Random + Filter + GroupDelay + Kurtosis	79.86	85.20	88.22	78.87	82.30	86.41	81.04	43.50	97.41	67.54	82.06	90.67	97.87	91.93	91.19	78.79	54.35	92.86	64.98	82.39	91.90	97.04	92.19	92.53
29	SNR Random + Filter + GroupDelay + RMS	88.86	82.96	88.68	87.05	82.20	87.71	88.86	48.88	100.00	66.35	83.41	96.89	97.63	90.58	97.41	87.10	50.89	99.32	63.86	82.20	93.48	97.93	91.50	92.34

30	SNR Random + Filter + FFT + Kurtosis	86.73	82.51	87.60	87.34	79.51	86.45	88.86	50.67	99.48	63.03	78.03	94.30	98.34	90.13	92.75	87.37	51.78	98.91	62.10	79.70	92.36	97.57	90.91	91.76
31	SNR Random + Filter + FFT + RMS	91.00	83.86	87.91	91.00	83.69	85.60	91.94	47.53	100.00	64.22	82.51	97.41	96.21	90.58	93.78	91.00	46.33	100.00	63.83	83.64	94.57	95.22	90.72	91.14
32	SNR Random + Filter + ZNorm + Kurtosis	90.05	76.68	84.96	89.60	76.30	82.73	91.23	43.05	92.23	51.42	76.68	32.12	98.10	89.69	64.25	89.69	43.95	85.82	50.64	76.23	26.92	97.59	90.28	73.98
33	SNR Random + Filter + ZNorm + RMS	90.28	78.03	84.50	90.22	76.21	83.74	91.23	43.95	92.23	45.26	77.13	26.94	97.16	88.79	62.69	90.28	45.48	91.46	42.27	76.16	26.64	96.88	90.94	66.13
34	SNR Random + Filter + Kurtosis + RMS	90.05	83.41	88.37	86.90	82.42	85.75	89.57	52.47	99.48	66.59	82.96	95.85	97.63	91.48	84.46	86.90	49.85	97.33	64.44	82.30	90.46	97.76	88.88	86.47
35	GroupDelay + FFT + Kurtosis	95.26	88.79	92.40	80.52	64.13	78.30	95.97	46.64	100.00	39.10	86.55	100.00	99.53	93.27	100.00	80.57	57.66	95.72	34.34	64.20	65.56	84.07	76.37	71.39
36	GroupDelay + FFT + RMS	95.26	87.89	91.01	84.30	66.25	74.85	96.21	44.84	100.00	44.08	88.34	100.00	99.29	92.83	99.48	84.42	49.61	98.91	34.88	66.32	68.48	80.26	75.17	72.95
37	GroupDelay + ZNorm + Kurtosis	93.36	76.23	86.20	91.31	75.90	84.15	94.08	40.36	91.71	59.24	74.44	4.66	98.10	89.24	47.15	91.24	40.38	90.76	57.99	75.93	3.71	96.87	91.62	49.96
38	GroupDelay + ZNorm + RMS	91.94	73.99	85.27	88.83	73.38	82.58	93.60	39.46	95.34	60.66	72.20	38.34	99.29	88.79	87.56	88.80	41.09	94.96	61.06	73.38	40.58	97.83	90.77	90.40
39	GroupDelay + Kurtosis + RMS	94.79	87.44	92.40	84.83	65.80	74.61	96.45	46.64	100.00	41.47	87.89	100.00	98.82	94.17	99.48	84.78	53.32	99.65	34.62	65.85	70.11	82.70	71.89	66.43
40	FFT + Kurtosis + RMS	94.55	87.89	92.71	81.59	64.20	76.42	94.31	46.64	100.00	40.05	85.65	100.00	99.05	94.62	98.96	81.54	55.11	98.01	34.41	64.17	65.94	79.17	81.61	76.79
41	ZNorm + Kurtosis + RMS	92.18	73.99	86.51	90.07	73.99	84.53	93.36	46.19	97.41	53.55	74.44	4.15	98.82	87.00	36.79	90.07	43.97	97.76	53.49	73.85	2.51	98.22	90.02	37.63
42	SNR Random + Filter + GroupDelay + FFT + Kurtosis	85.31	82.96	88.37	82.86	80.88	87.12	85.31	43.95	97.93	65.40	76.23	96.89	98.34	90.58	90.16	82.84	50.39	97.38	64.37	80.93	93.40	97.69	93.70	92.96
43	SNR Random + Filter + GroupDelay + FFT + RMS	92.65	80.72	83.10	92.07	81.43	82.86	93.36	47.98	99.48	67.54	82.06	98.45	97.16	83.41	100.00	92.13	44.32	99.51	63.83	81.43	94.41	96.63	88.77	97.44
44	SNR Random + Filter + GroupDelay + ZNorm + Kurtosis	90.76	73.99	86.67	91.47	75.57	85.88	91.94	45.74	94.30	53.08	78.92	28.50	98.34	89.69	61.66	91.53	44.73	90.35	54.27	75.62	19.88	98.02	90.65	72.43
45	SNR Random + Filter + GroupDelay + ZNorm + RMS	92.18	74.44	84.19	91.86	72.83	84.39	93.60	38.57	88.60	63.27	70.40	69.43	96.92	90.58	57.51	91.82	41.99	87.16	64.79	72.95	71.23	97.03	93.08	57.51
46	SNR Random + Filter + GroupDelay + Kurtosis + RMS	89.57	83.86	88.37	90.62	81.35	87.49	91.94	47.98	99.48	62.80	83.41	88.60	98.34	93.72	92.23	90.61	45.88	99.54	62.95	81.45	89.77	97.63	92.80	93.89
47	SNR Random + Filter + FFT + Kurtosis + RMS	90.76	81.17	88.68	89.41	80.72	86.06	91.71	55.16	98.45	67.54	84.75	92.23	98.10	90.13	90.67	89.44	52.87	96.10	63.06	80.81	90.70	98.03	92.00	94.30

48	SNR Random + Filter + ZNorm + Kurtosis + RMS	90.28	77.13	84.34	89.85	74.25	83.44	89.57	47.53	98.45	60.43	73.99	41.45	98.34	90.13	60.62	90.00	44.96	99.40	59.59	74.18	38.12	97.74	90.58	78.07
49	GroupDelay + FFT + Kurtosis + RMS	93.36	87.44	91.63	83.05	66.08	79.83	94.55	43.50	100.00	42.65	87.00	100.00	99.05	93.27	95.85	83.04	49.49	99.95	35.08	66.06	70.79	88.10	77.67	71.86
50	GroupDelay + ZNorm + Kurtosis + RMS	93.84	77.58	85.58	90.10	77.27	84.30	93.84	45.29	91.19	60.66	80.27	6.22	97.87	89.24	32.64	89.99	43.17	89.96	60.09	77.25	4.39	97.74	91.93	31.88
51	SNR Random + Filter + GroupDelay + FFT + Kurtosis + RMS	90.28	84.30	84.65	88.73	83.15	82.08	88.86	48.88	100.00	65.88	83.41	95.85	97.87	90.58	93.26	88.73	46.99	99.75	65.89	83.17	94.25	96.56	87.37	96.07
52	SNR Random + Filter + GroupDelay + ZNorm + Kurtosis + RMS	91.71	78.92	87.13	91.42	76.09	84.40	91.94	39.91	94.30	59.48	78.03	49.22	98.10	89.24	64.77	91.33	42.60	90.84	63.96	76.07	50.45	97.79	89.52	67.60

Table E.1: Full list of test combinations and their accuracy results for the different test cases.

		CM OG+HF			SNR loop average OG + HF		
Num	**Modification**	**OG**	**HF**	**BC**	**OG**	**HF**	**BC**
12	SNR + Random + Filter + GroupDelay	96.92	91.93	97.93	97.02	92.40	95.45
16	SNR + Random + Filter + RMS	98.10	94.17	95.85	96.94	93.11	91.57
46	SNR + Random + Filter + GroupDelay + Kurtosis + RMS	98.34	93.72	92.23	97.63	92.80	93.89
29	SNR + Random + Filter + GroupDelay + RMS	97.63	90.58	97.41	97.93	91.50	92.34
2	SNR Random	97.87	89.69	94.82	97.51	92.92	93.18

Table E.2: Best Generalization test models. Both tests are composed of the "OG+HF" dataset combination in order to attempt to train a model to become more generalized. The SNR loop average test consists of accuracies for the SNR loop function on Figure D.5 being averaged across the range of [-20;60] dB SNR. The order of the best average model is from top to bottom if the accuracy is averaged across the 6 different test results.

E.2 Number 12 Figures

Following the results in table E.2, the model determined to be best model is testcase number 12. This appendix contains the figures for all the tests of number 12. The tests consisted of the data being applied 0 dB of SNR on it, then it is being filtered and applied group delay data on it.

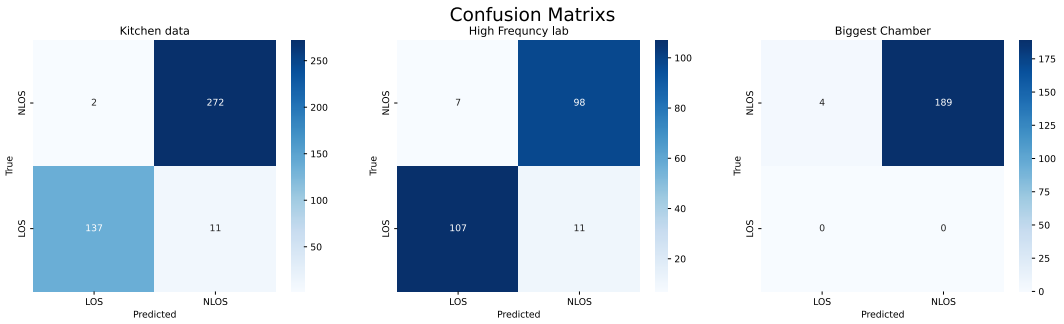


Figure E.1: CM, when the model is trained on both datasets and compared to each dataset individually.

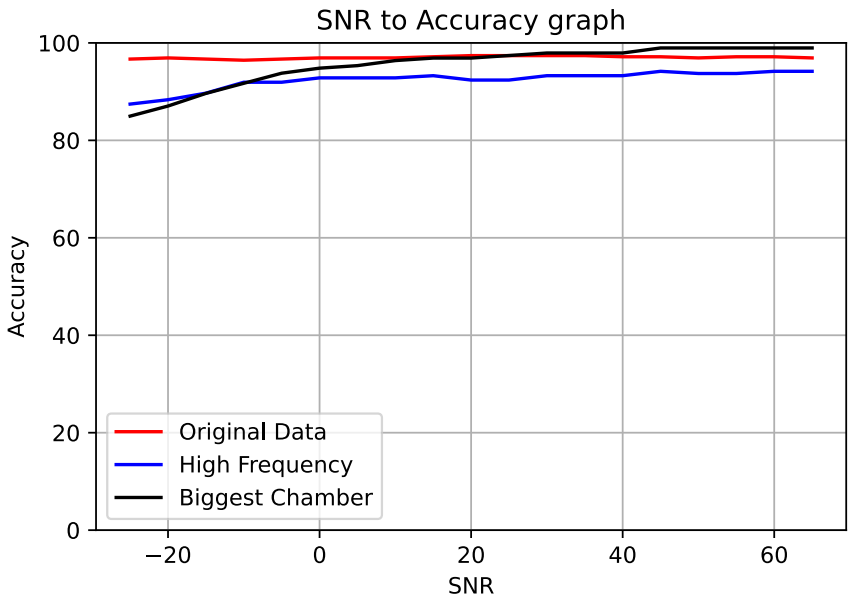


Figure E.2: SNR graph for the HF + OG .

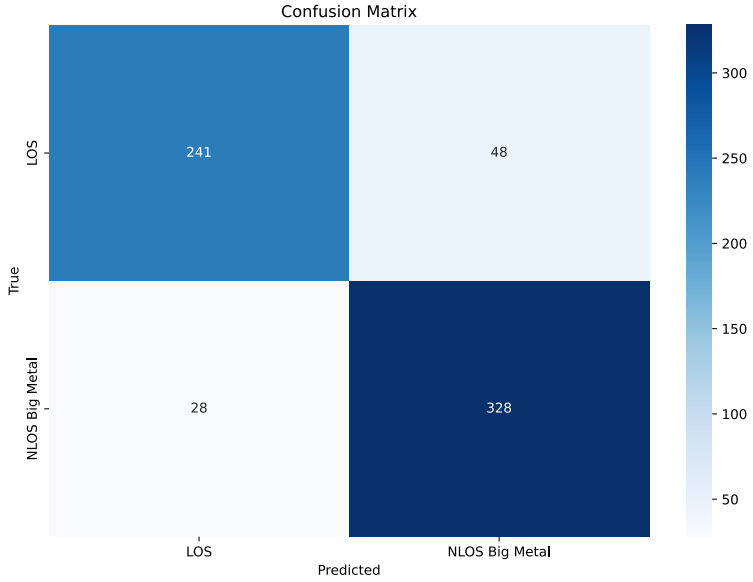


Figure E.3: CM of OG+HF combined. The accuracy is found to be 88.22%.

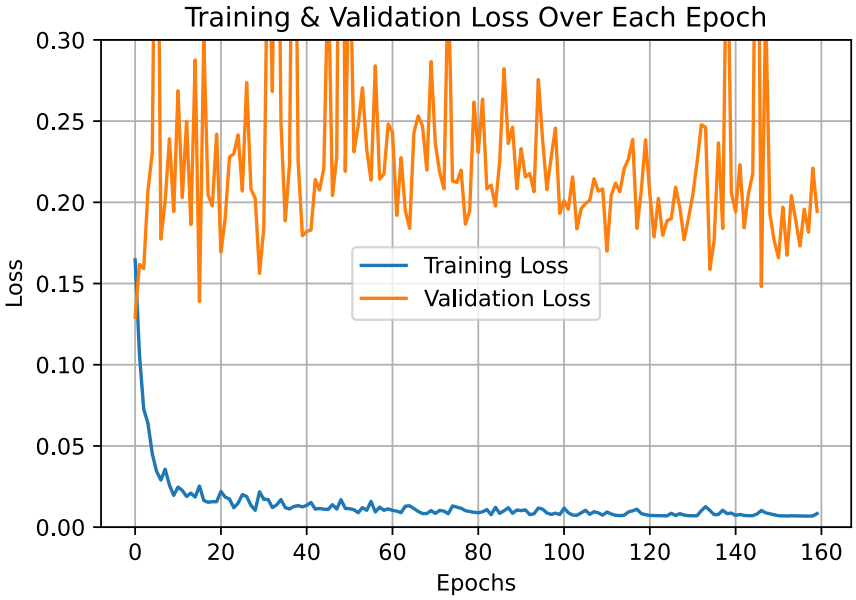


Figure E.4: Validation and Training loss graph for test number 12, for OG+HF training dataset.

E.3 SVM confusion matrices

The confusion matrices belonging to Table 7.4.

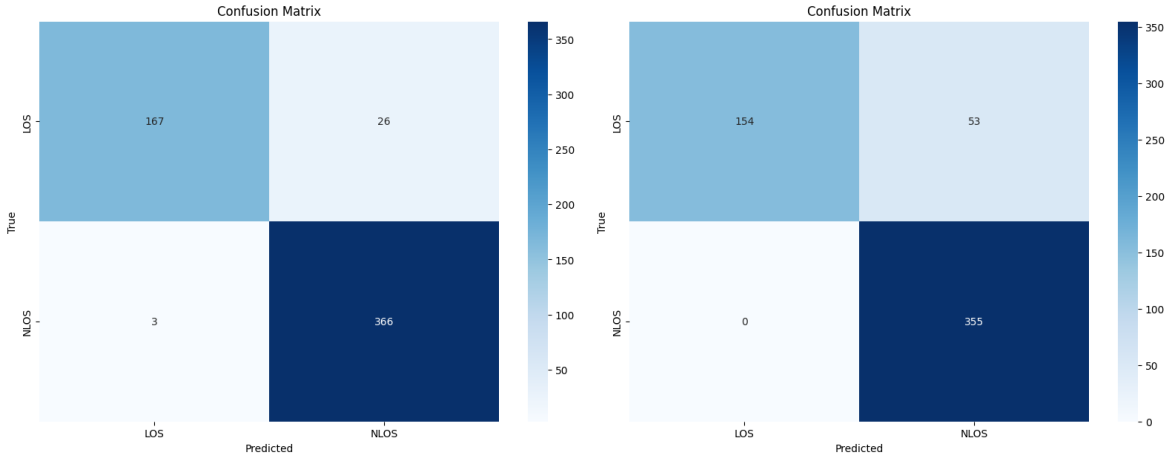


Figure E.5: CM of the Poly kernel with OG data with a degree of three. Left figure is without noise, while right CM is with randomized noise between -20 and 60 dB SNR.

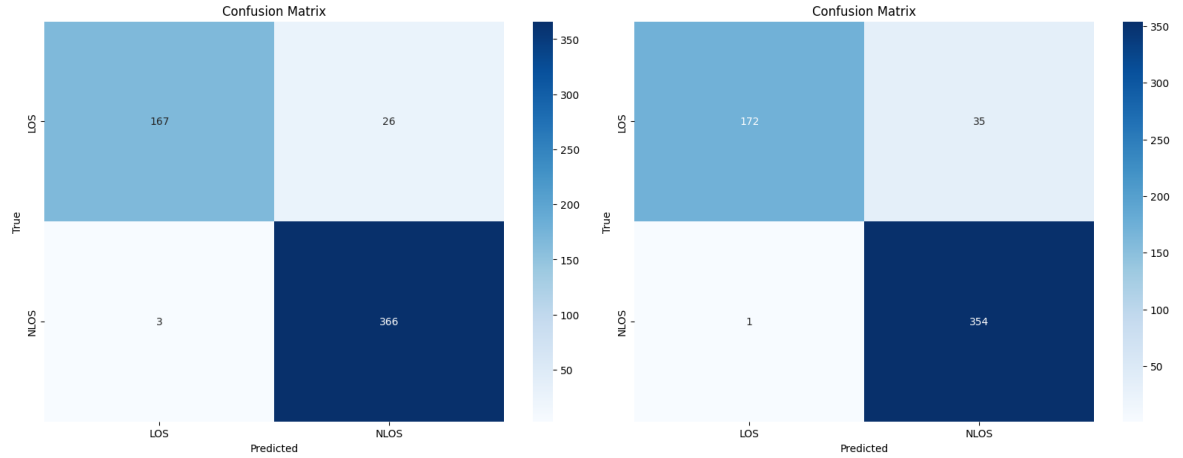


Figure E.6: CM of the RBF kernel with OG data. Left figure is without noise, while right CM is with randomized noise between -20 and 60 dB SNR.

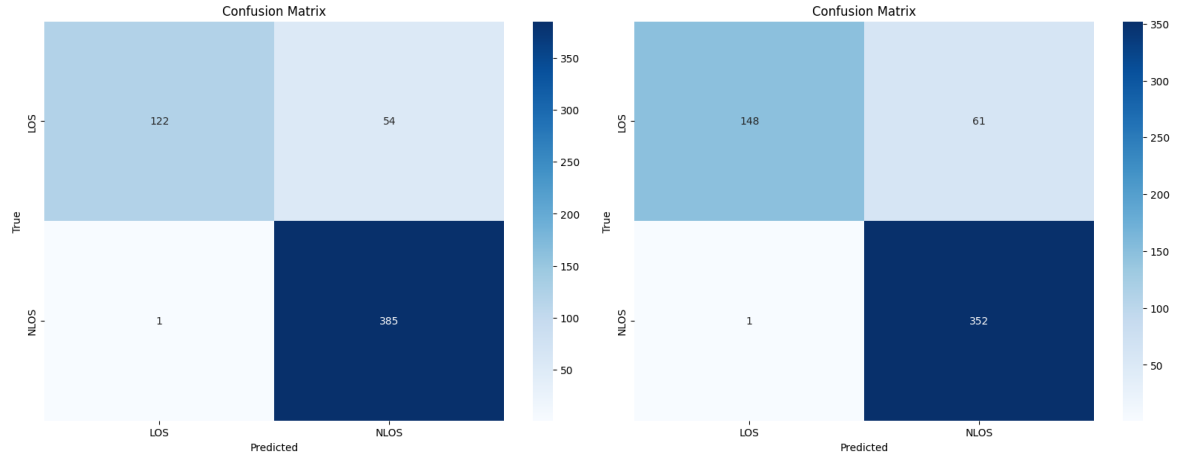


Figure E.7: CM of the Poly kernel with HF data with a degree of three. Left figure is without noise, while right CM is with randomized noise between -20 and 60 dB SNR.

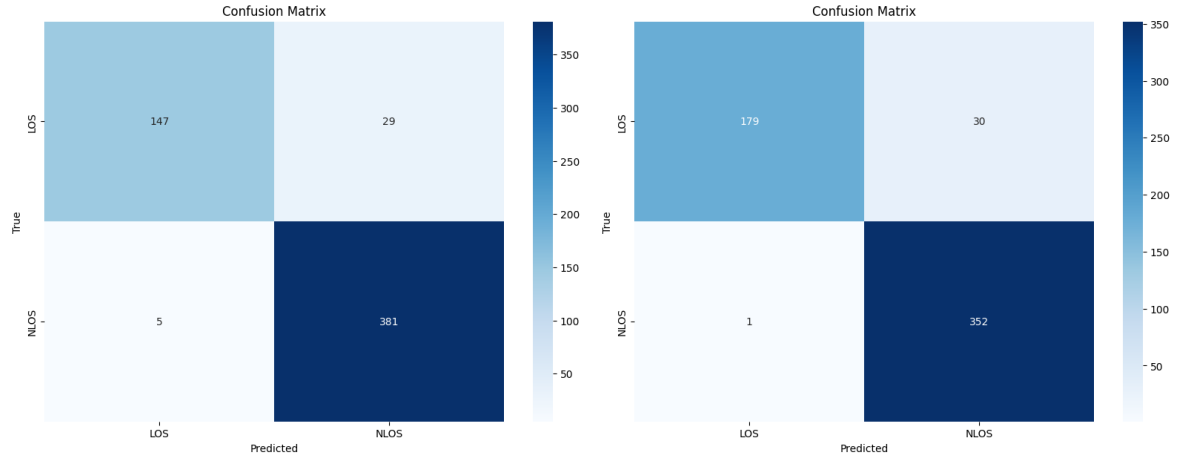


Figure E.8: CM of the RBF kernel with HF data. Left figure is without noise, while right CM is with randomized noise between -20 and 60 dB SNR.

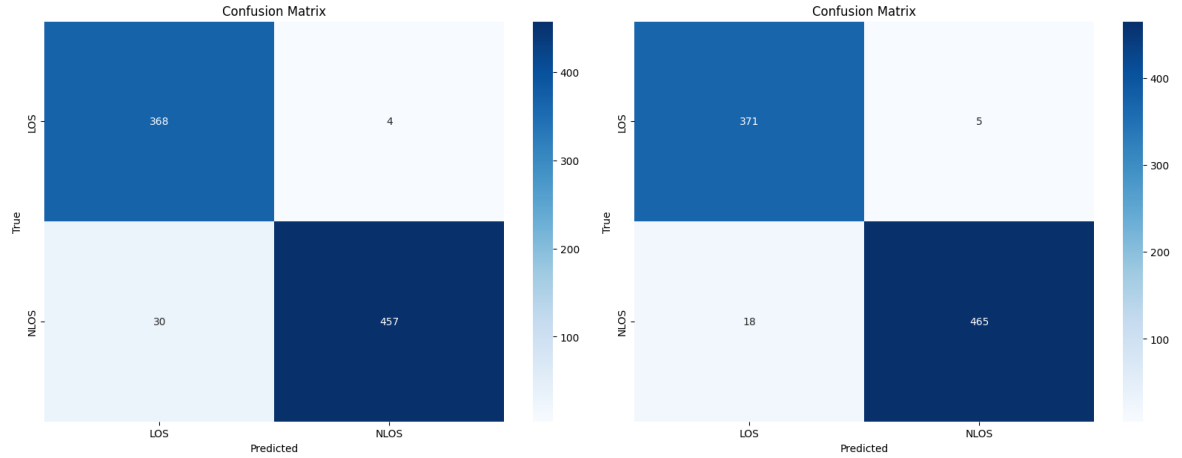


Figure E.9: CM of the Poly kernel with OG+HF data with a degree of three. Left figure is without noise, while right CM is with randomized noise between -20 and 60 dB SNR.

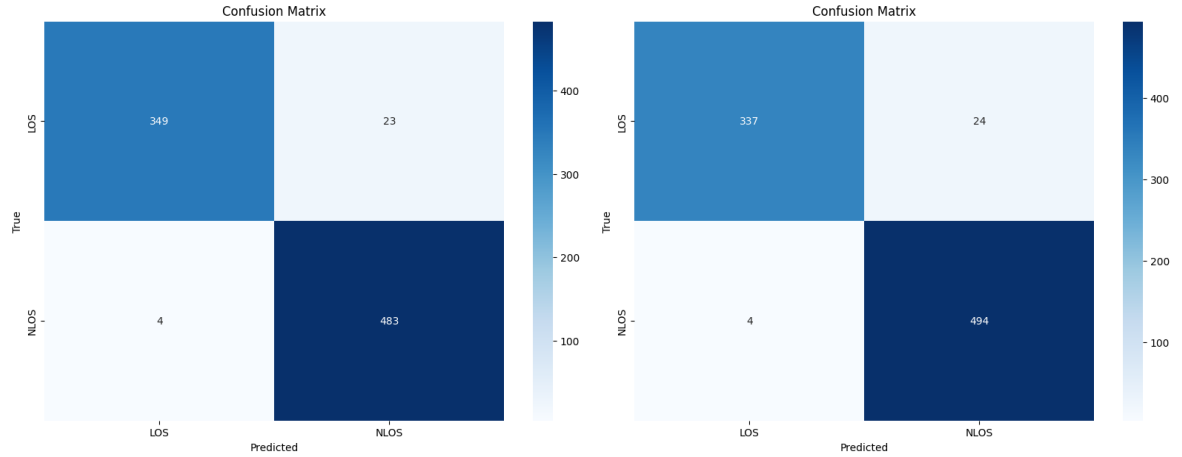


Figure E.10: CM of the RBF kernel with OG+HF data. Left figure is without noise, while right CM is with randomized noise between -20 and 60 dB SNR.

F. Model explainability

When training ML models, it is important to carefully choose what data is given to the model. Relevant features will improve accuracy, while irrelevant features will decrease accuracy. When looking at the channel impulse response (CIR), it is intuitive to look at the peak amplitude alone to differentiate between LOS and NLOS. Biases like these are something that is hoped that a ML model could use to tell NLOS and LOS apart without having to explicitly give the data[1]. One thing is, that the ML models might be able to find some other more reliable relationship in the data. Therefore, being able to explain what the ML models are really focusing on, and what contributed to the final classification can be a great benefit. This is called *model explainability* [13, p. 533]. The more complex a model is, the more difficult it generally is to explain the model behavior as illustrated in Figure F.1.

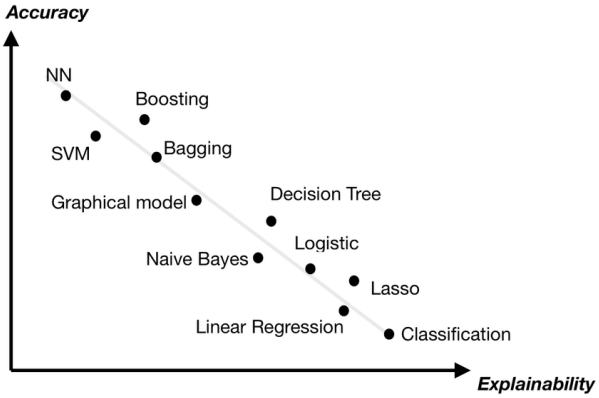


Figure F.1: Illustrative of how complexity of a model typically increases the accuracy, at the cost of making it harder to know what leads the model to its decision in the end [54, Fig. 1].

From the CIR plots of the different datasets, it is easy to tell NLOS and LOS apart just by looking at the peak amplitude. In the HF dataset, it becomes rather difficult to tell it apart by only looking for the peak amplitude. Note that the HF data is sampled over a longer time period.

By utilizing the domain knowledge in RF and adding more data to the model, a hypothesis could be that, a collection of peak amplitude, RMS delay, kurtosis and Ricean K-factor data would be adding good features for NLOS identification.

It can also be hypothesized, that not all the data points needs to be used when training the model. Just a portion of the data could prove to be enough for a somewhat similar accuracy.

[1] In this case, the peak amplitude is implicitly in the data due to the very nature of the amplitude data in the S21(DB) column.

The standard DNN model that will be used for showing the feature importance will have the parameters shown in Table F.1.

Layer	Input Features	Operations
Input	3606	None
Hidden 1	2048	BatchNorm, ReLU, Dropout
Hidden 2	512	BatchNorm, ReLU, Dropout
Hidden 3	256	BatchNorm, ReLU, Dropout
Hidden 4	128	None
Output	2	Softmax

Table F.1: DNN architecture for LOS/NLOS classification. Input features consist of 1801 S21(DB) values, 1801 S21(DEG) values, and 4 statistical features. Dropout rate is 0.3.

F.1 Feature importance

It is often said that ML models are a *blackbox*, but there are ways to figure out what features given to a ML model that are actually important for the final prediction. For an SVM with a non-linear kernel, all feature permutations have to be bruteforced to calculate the important features, while for a DNN many methods exists - Deep Learning Important FeaTures (DeepLift) will be used here.

F.1.1 DNN feature importance

Using the DeepLift algorithm, it is possible to find the importance of the input features, i.e. how big of a contribution a feature provides to the final classification for a DNN. In Figure F.2, the top 30 most significant input features can be seen on the left plot, where it appears that the most important features are S21(DB) data located around index 300. On the far right, the plot of the channel impulse response shows the start and end of the interval with the 30 most significant features that combined have the biggest consecutive sum of feature importance. It would be expected that the peak amplitude of the signal would have the highest importance, but from analysis of the biggest contribution window, it seems like the model does not look at this implicitly. As it is not the case, it would be worth providing other features about statistics, such as the peak amplitude for the signal. It can also be seen from the figure, that the signal amplitude is generally more important information than the phase of the signal.

The model used to generate the graphs is a default DNN (from Table F.1), with no data augmentation or any additional features other than the pure CIR data.

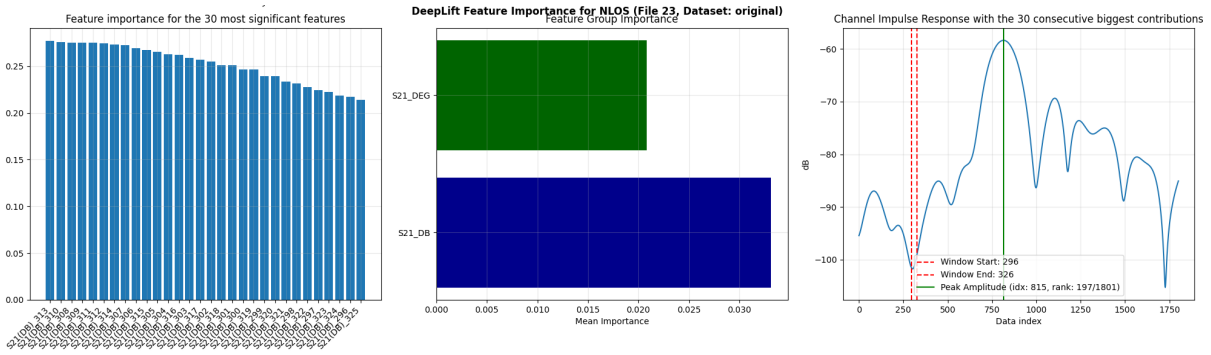


Figure F.2: DeepLift feature importance plots for a NLOS file, for a model trained with no augmentation or preprocessing steps. Generally, the magnitude of the signal is more important for the final classification compared to the phase of the signal. On the right, the plot of the channel impulse response shows the start and end of the interval with the features that combined have the biggest sum of feature importance, where the top 30 are plotted on the left most plot.

A new DNN model has been trained that got the peak amplitude, kurtosis, RMS delay and Ricean K-factor, but no other forms of data augmentation. The hypothesis is that peak amplitude should show up as one of the most significant features now as it is given explicitly. In Figure F.3, it turns out that these statistical features have a higher average feature importance compared to the amplitude and phase of the signal.

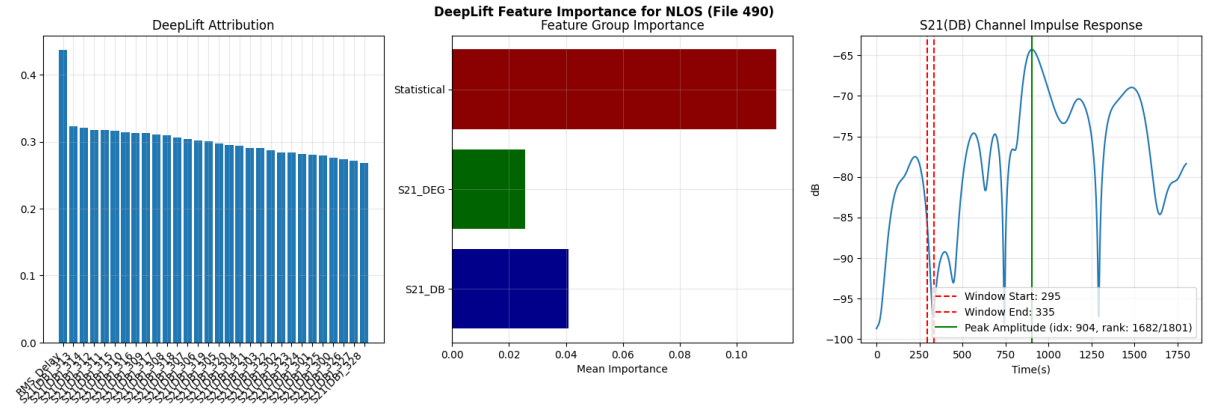


Figure F.3: Model trained on OG+HF, where 4 additional features are given.

For the new model, the feature importance weights can be seen on Table F.2. It can be seen that for NLOS cases, the magnitude of the signal is more important compared to the LOS case. Also, generally what can be seen is that the most significant statistical feature is the RMS delay, while the peak amplitude and the kurtosis could be omitted and not severally affect the final classification. A graphical representation can be seen on Figure F.4.

Feature	LOS	NLOS
S21_DB	0.029230	0.040942
S21_DEG	0.017422	0.025790
Statistical Features	**LOS**	**NLOS**
Kurtosis	0.000050	0.000200
Peak Amplitude	0.000051	0.000203
RMS Delay	0.132326	0.436818
Ricean K-factor	0.005395	0.018960

Table F.2: Feature Importance comparison between LOS and NLOS cases for a DNN trained on OG+HF dataset with 4 additional statistical features added together with the CIR.

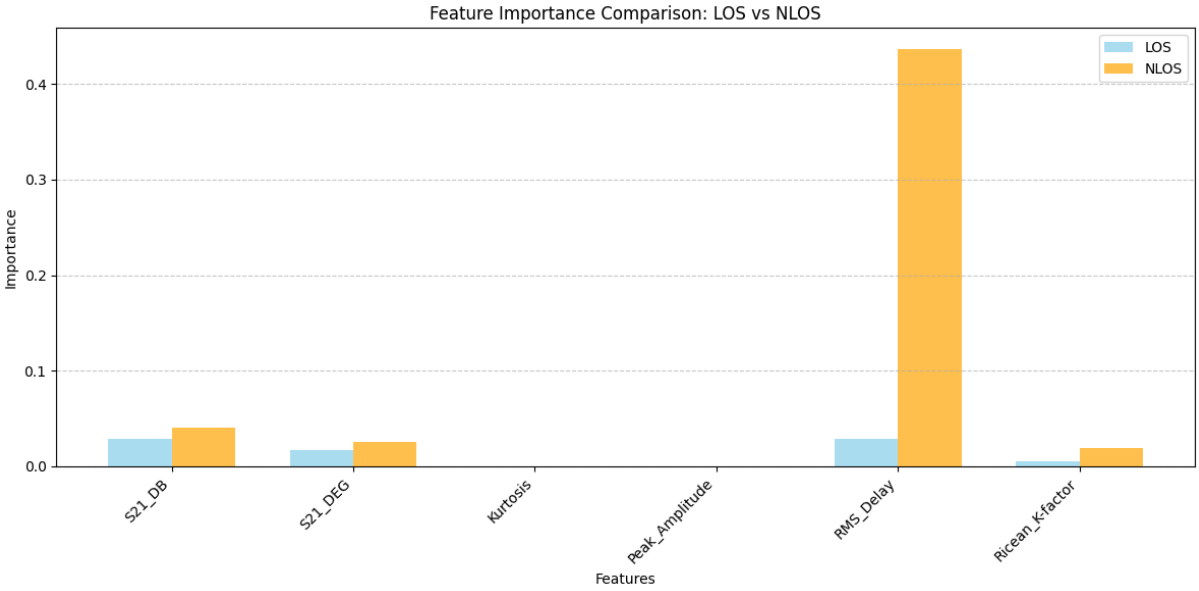


Figure F.4: Feature importance comparison for the LOS and the NLOS data for the new model.

To test whether or not peak amplitude is a deciding factors, a similar DNN model to the model seen in Table F.1 have been designed as a modified version with an attention block. The attention architecture can be seen on Table F.3, where the product of the num heads and embed dimension are greater then the tensor dimension (3606) to make sure that all input features can be "watched" by the attention heads.

	Value
Num heads	8
Embed dimension	512

Table F.3: Architecture of the DNN with an attention block at the start.

The attention mechanism should focus on the statistical features and make the influence of the statistical features of more importance than found for the DNN trained with the four statistics seen in Figure F.4. The attention + DNN model feature importance plots can be seen on Figure F.5. The conclusion again is that it seems like even with the attention mechanism, the model does not care much about neither the implicit peak amplitude that is directly in the CIR,

nor the explicit peak amplitude feature given. Table F.4 and F.6 show the same story as earlier, RMS delay is the most significant statistic.

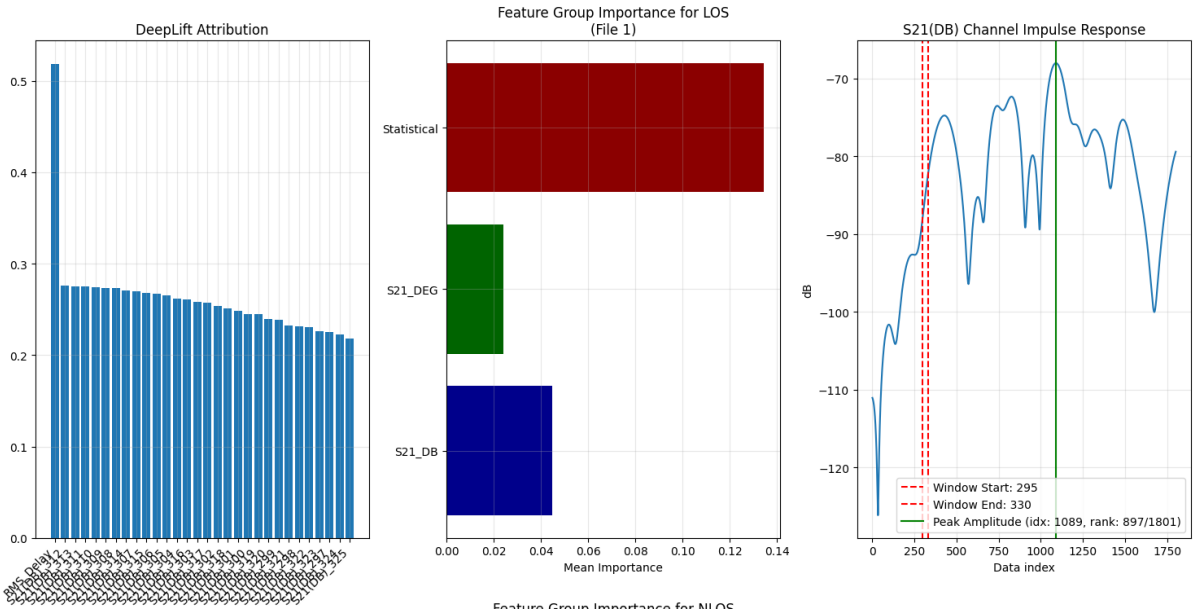


Figure F.5: Attention+DNN trained on OG+HF added 4 additional features.

Feature	**LOS**	**NLOS**
S21_DB	0.044754	0.053156
S21_DEG	0.024061	0.034802
Statistical Features	**LOS**	**NLOS**
Kurtosis	0.000798	0.000143
Peak Amplitude	0.000788	0.000146
RMS Delay	0.018122	0.601522
Ricean K-factor	0.018301	0.013302

Table F.4: Feature importance values for LOS and NLOS scenarios.

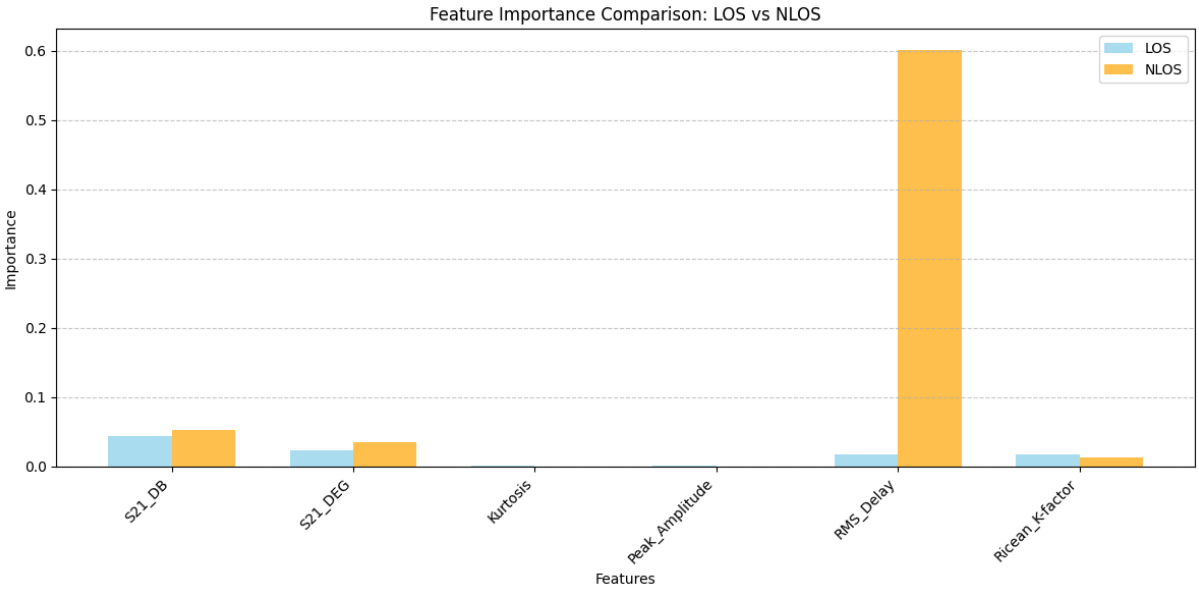


Figure F.6: Feature importance between different features given to the model using attention+DNN, trained on OG+HF.

To look further into this, a plot of how the peak amplitude is distributed for the original dataset can be seen on Figure F.7 and for the HF dataset on Figure F.8. The red lines shows the interval [280; 330], which is the interval often observed in feature importance plots to be the most significant, based on the DeepLift feature importance values.

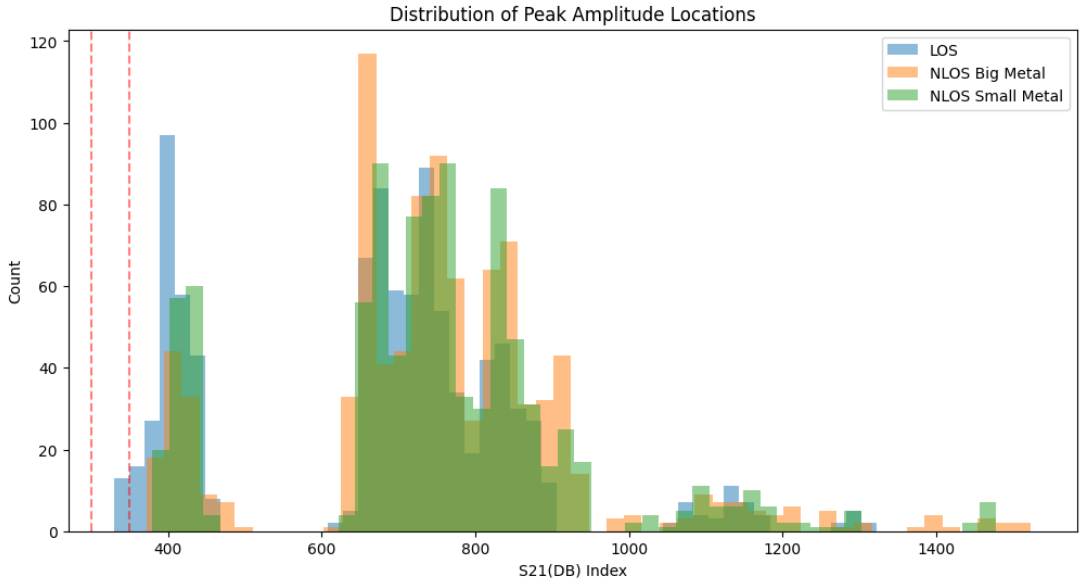


Figure F.7: Peak amplitude distribution over all files in the original dataset with an elevation of 0 degrees.

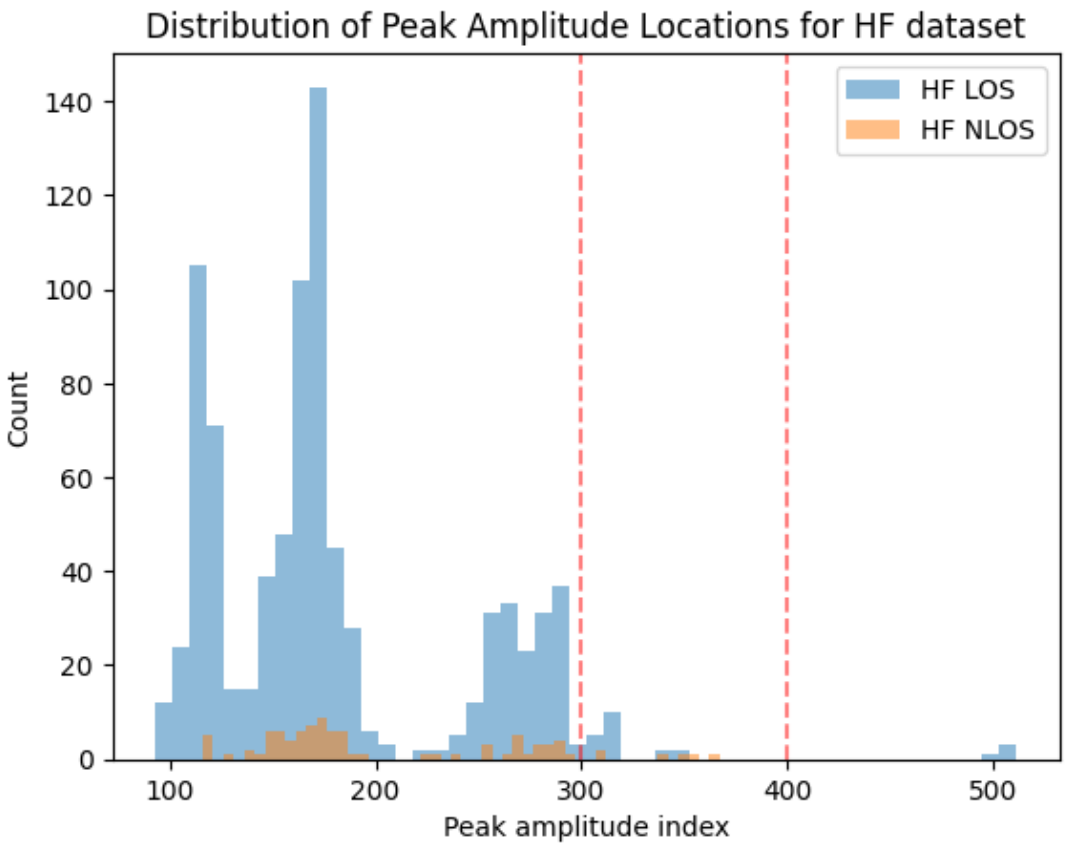


Figure F.8: Peak amplitude distribution over all files in the HF dataset with an elevation of 0 degrees.

It seems like there is no correlation between the 30 most important features produced by DeepLift and neither a LOS or NLOS scenario, across the original dataset and HF dataset.

Similarly, an analysis was done for the RMS delay. On Figure F.9 it seems somewhat indistinguishable to tell LOS and NLOS cases apart based on the original dataset. The same for the HF dataset seen on Figure F.10, but this is mostly due to all HF NLOS measurement with an elevation different from 0 degrees is filtered out.

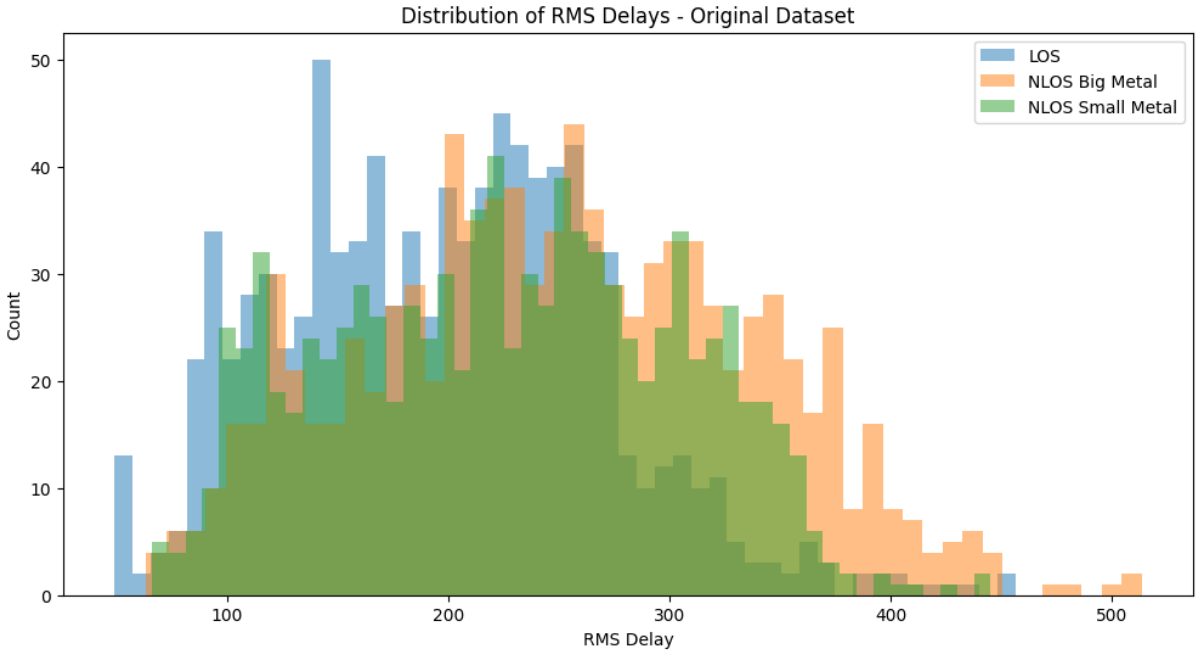


Figure F.9: RMS delay distribution over all files in the original dataset with an elevation of 0 degrees.

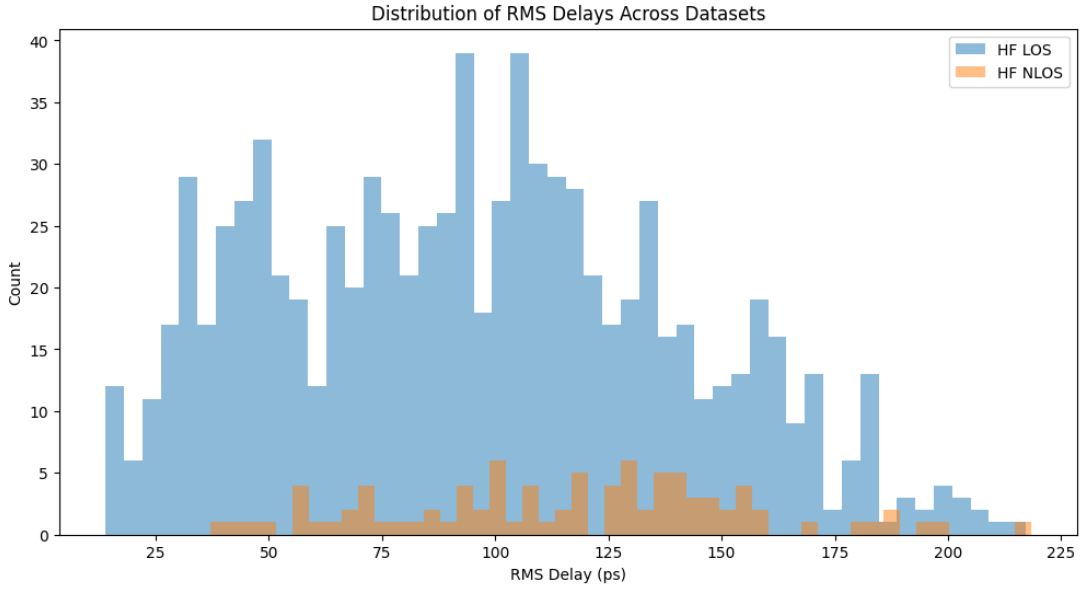


Figure F.10: RMS delay distribution over all files in the HF dataset with an elevation of 0 degrees.

Fixing feature under utilization

It seems like the DNN is suffering from feature under utilization - It simply just does not use all the provided data. It turns out that giving the statistical features twice solves this issue as seen on figures F.11-F.13 such that the tensor dimension is 368 (without ricean K-factor).

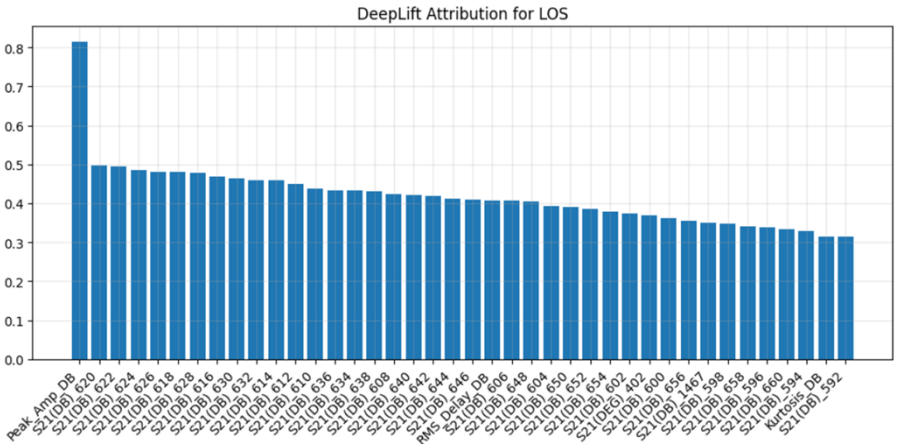


Figure F.11: Feature importance with statistical features given twice for LOS on original dataset.

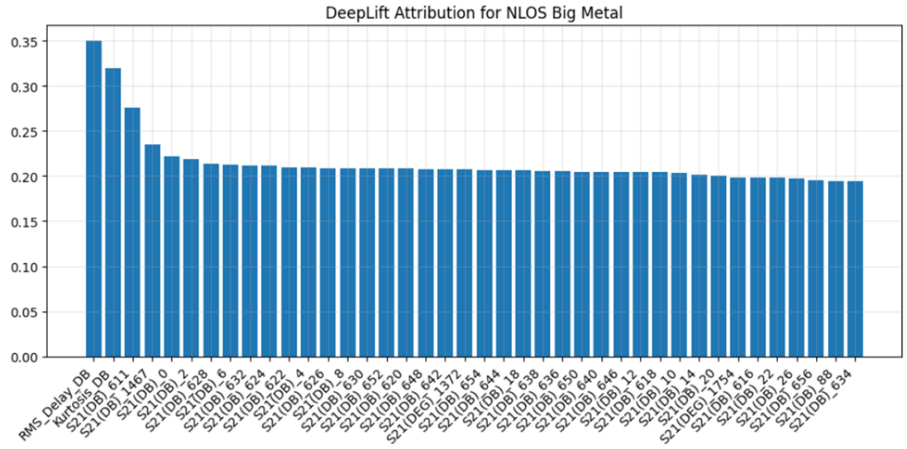


Figure F.12: Feature importance with statistical features given twice for NLOS with big metal plate on original dataset.

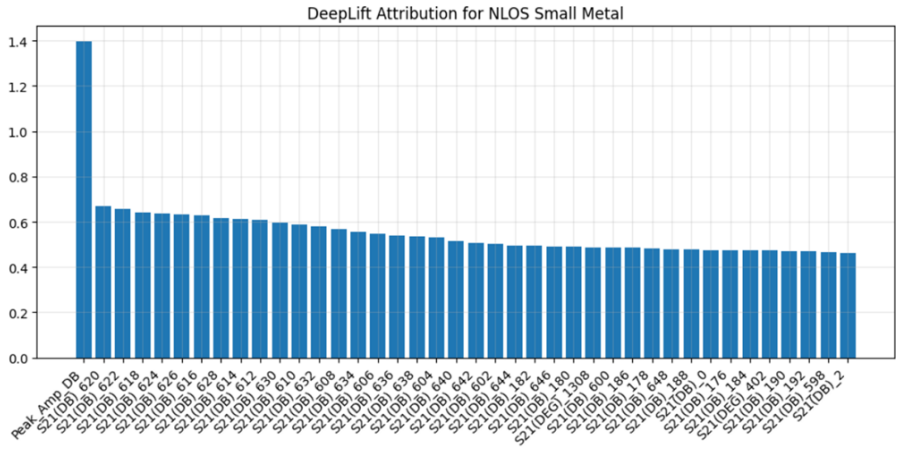


Figure F.13: Feature importance with statistical features given twice for LOS on original dataset.

This model ended up proving to perform very well on OG dataset with an accuracy of 97%, where the confusion matrix can be seen on Figure F.14. The model was also trained on OG+HF got an accuracy of 95%.

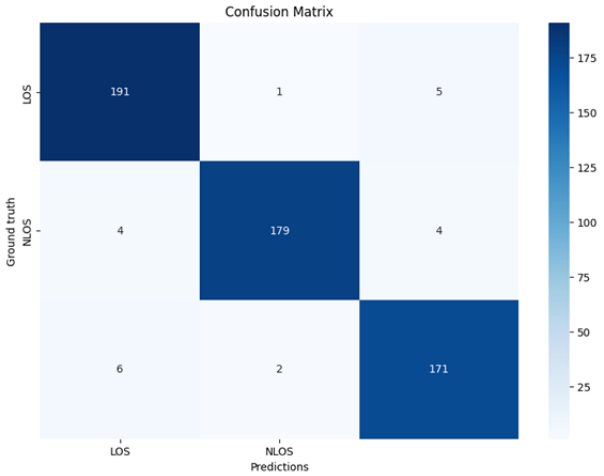


Figure F.14: Confusion matrix of newly proposed DNN trained on OG dataset that uses 3 statistics features given twice.

Thus as expected with the original hypothesis, the peak amplitude it indeed a very important factor for NLOS identification for the given dataset.