Markus Gallacher, BSc

NLOS Ranging Error Mitigation using Machine Learning on UWB Systems with Constrained Resources

MASTER'S THESIS

to achieve the university degree of

Master of Science

Master's degree programme: Information and Computer Engineering

submitted to

Graz University of Technology

Supervisor

Carlo Alberto Boano, Assoc.Prof. Dott. Dott. mag. Dr.techn. MSc

Institute of Technical Informatics

Technical Supervisor

Michael Stocker, Dipl.-Ing. BSc

Institute of Technical Informatics

Graz, December 2021

AFFIDAVIT

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used. The text document uploaded to TUGRAZonline is identical to the present master's thesis.

13.12.2021 

Date, Signature

Abstract

Ultra-wideband (UWB) technology is becoming an increasingly popular technology for range estimation, localization, and tracking. Thanks to its high bandwidth, UWB allows for decimetre-level ranging accuracy in unobstructed line-of-sight (LOS) environments. These accurate ranging measurements can be used in a variety of applications, such as asset tracking, theft detection, and keyless car access security.

Despite the advantages UWB has to offer over other ranging methods, it is still prone to large ranging errors when obstacles block or attenuate the direct signal path, i.e., in non-line-of-sight (NLOS) conditions. These obstacles can indeed increase the estimated distance up to a few metres. While the latter is still better than the achievable accuracy with traditional narrow-band technologies (e.g., Wi-Fi, Bluetooth) it is often insufficient for several of the aforementioned applications. The NLOS problem has been investigated intensively over the past years, and several different approaches have been proposed. The two main approaches are either to identify LOS situations or to directly mitigate the ranging error. Most approaches analyze channel statistics derived from the channel impulse response (CIR). Other more advanced machine learning methods can automatically extract information from the CIR. In this thesis, the focus lies specifically on assessing the feasibility and performance of machine learning methods, such as support vector machines and neural networks to identify LOS via classification and to mitigate ranging errors via regression, on constrained embedded devices.

To evaluate the performance and resource requirements of different NLOS detection and mitigation techniques, a measurement campaign has been conducted in 12 different locations at the university and recorded on 749 measurement points. This data is used to train the machine learning classifiers and regressors, as well as to evaluate these different approaches in terms of decreasing residual ranging error. In many situations, it is desired to run inference directly on resource-constrained devices. The contribution of this thesis is the evaluation of several machine learning models and the implementation of the best models on a constrained device. The results showed that a neural network approach performs the best for NLOS mitigation, achieving a median absolute error improvement by up to 39.4% and an improvement of the 90^{th} percentile absolute residual ranging error by up to 50.7%. The neural network classifier used, achieves an F1-score of 92%. By exporting the models as a TensorFlow Lite model, the lightweight versions of the models can be used on a microcontroller to evaluate the resource requirements. The non-quantized model takes 242 ms to run classification and regression, as well as 88 kB of RAM, and 334 kB of ROM. Furthermore, a full integer quantized model was evaluated, where the runtime could be reduced by 43% and the ROM by 19%.

Kurzfassung

Die Ultrabreitband (UWB)-Technologie zählt zunehmend zu den beliebtesten Technologien, um Distanzmessungen, Lokalisierungen oder Nachverfolgungen durchzuführen. Dank seiner hohen Bandbreite, erlaubt UWB eine Distanzmessung im Dezimeterbereich in Umgebungen mit line-of-sight (LOS). Diese genauen Distanzmessungen können in mehreren Aufgabenbereichen, wie zum Beispiel für die Güterverfolgung, die Diebstahlerkennung und den sicheren schlüsselosen Autozugang, verwendet werden.
Trotz der Vorteile, die UWB gegenüber anderer Distanzmessmethoden bieten kann, ist sie anfällig für große Messfehler, wenn der direkte Signalpfad von Objekten geschwächt oder blockiert wird, d.h., in non-LOS (NLOS) Bedingungen. Solche Objekte können sogar zu einer Verschätzung der Distanz um einige Meter führen. Auch wenn UWB in NLOS Bedingungen genauere Distanzmessungen als traditionelle Schmalband-Technologien (z. B. Wi-Fi, Bluetooth) erzielen kann, ist es oft für die genannten Anwendungen nicht exakt genug. Das NLOS-Problem wurde in den letzten Jahren eingehend untersucht und einige Ansätze wurden vorgeschlagen. Die zwei Hauptansätze beziehen sich, entweder auf die Identifikation der LOS-Bedingungen oder die direkte Reduktion der Distanzfehler. Der Großteil der Ansätze analysiert die Kanalstatistiken, die von der Kanalimpulsantwort (CIR) berechnet werden. Andere fortgeschrittenere maschinelle Lernmethoden, können automatisch Informationen aus der CIR gewinnen. In dieser Masterarbeit wird der Fokus speziell auf die Beurteilung der Durchführbarkeit und der Leistung maschineller Lernmethoden, wie Support-Vektor-Maschinen und neuronaler Netzwerken, auf ressourcenbegrenzten eingebetteten Geräten, um LOS-Bedingungen mittels Klassifizierung zu identifizieren und um Distanzfehler durch Regression zu reduzieren.
Um die Leistungen und Ressourcenanforderungen mehrerer NLOS Klassifizierungs- und Regressionstechniken zu evaluieren, wurden Messungen in 12 verschiedenen Bereichen der Universität durchgeführt und an 749 Messpunkten aufgenommen. Diese Daten wurden für das Trainieren der Klassifizierungs- und Regressionsmodelle benützt. Die Ergebnisse zeigen, dass der Ansatz mit neuronalen Netzwerken unter NLOS-Bedingungen am effektivsten und eine Verbesserung des Median des absoluten Distanzfehlers von bis zu $39,4\%$ und eine Verbesserung des 90^{sten} Perzentil des absoluten Distanzfehlers von bis zu $50,7\%$ erzielt. Der neuronale Netzwerk-Klassifizierer erreicht eine F1-Wertung von 92%. Das Exportieren der Modelle als TensorFlow-Lite-Modelle, ermöglicht die Anwendung der leichtgewichtigen Modellversionen auf einen Mikrokontroller und die Evaluierung der Ressourcenanforderungen. Die Klassifizierungs- und Regressionsmodelle weisen eine gemeinsame Laufzeit von 242 ms auf und benötigen 88 kB RAM und 334 kB ROM. Darüber hinaus, wurde ein Full Integer quantisiertes Modell evaluiert, dessen Laufzeit um 43% und dessen ROM-Speicherbedarf um 19% reduziert werden konnten.

Acknowledgements

This thesis has been conducted at the Institute of Technical Informatics at Graz University of Technology.

First and foremost, I would like to thank my technical supervisor Michael Stocker, as well as my supervisor Carlo Alberto Boano for their support and help throughout this thesis. Without them, the current scope of this work would not have been possible.

Furthermore, I want to thank my partner, family, and friends for their constant support and motivation. Not only during the course of this thesis, but beyond that. Without them, I would not have come this far.

Graz, December 2021 Markus Gallacher

Contents

List of Figures

List of Tables

List of Acronyms and Symbols

UWB Ultra-Wideband

LOS Line-of-Sight

WLOS Weak-LOS

NLOS Non-LOS

MAE Median Absolute Error

RRE Residual Ranging Error

CIR Channel Impulse Response

PRF Pulse Repetition Frequency

ToA Time of Arrival

ToF Time of Flight

SVM Support Vector Machine

SVR Support Vector Regression

SVC Support Vector Classification

CNN Convolutional Neural Network

RSSI Received Signal Strength Indicator

FPPL First Path Power Level

PCA Principal Component Analysis

1 Introduction

As IoT applications are increasing rapidly in popularity and as the number of connected devices that need accurate distance measurements is ever-growing, the ability to accurately and quickly measure the distance between two objects is of great value to many services of modern days. In the past, mostly narrow-band radio frequency (RF) technologies, like Wi-Fi and Bluetooth, were used to estimate the distance between devices. These ranging methods use the signal strength to determine the distance between two objects. However, this is prone to errors as the signal strength varies drastically depending on the obstacles and properties of the surrounding environment [12]. Either the signal is directly attenuated or blocked by the obstacle or the signal's reflections cause constructive or destructive interference that alter the received signal [13]. All these factors lead to metre-level accuracy in the best case [14]. In recent years, however, ultra-wideband (UWB) technology has emerged as one of the most promising ways to estimate distances between devices, as it uses short repeated radio pulses with a bandwidth greater than $500MHz$ that lead to a high time resolution and can achieve an accuracy level in centimetres [15]. UWB signals are also less prone to fading, as there are far less constructive or destructive interferences from reflections [16] [17]. Furthermore, the high time resolution of UWB signals can be used to create a channel impulse response (CIR) that depicts the magnitude and timing of the first path and each detected reflection. The CIR can be used to analyze the signal behaviour in different environments and to extract statistical data to solve various problems like activity recognition [18], or position estimation [19]. UWB distance measurements can be used for many different applications, such as asset tracking and building control [20], while large companies like Apple and Samsung are also incorporating UWB technology into their high-end smartphones to enable accurate localization applications. Similarly, car companies like Volkswagen and Tesla are improving their keyless vehicle access by including UWB distance verification to combat signal relay attacks. Nevertheless, UWB distance estimates are not flawless and the reliability of a distance measurement correlates with the environment and is best when there is a clear line-of-sight (LOS) between the two devices. However, a LOS situation cannot be guaranteed, as obstacles can partially or fully block the LOS, as the devices may move between multiple rooms. Therefore, UWB signals can be subjected to attenuation, reflection, refraction, and diffraction. [21]

1.1 Problem Statement

Ranging errors greater than a few decimetres can have a significant effect depending on the application at hand. If an application, such as localizing a tag, requires $< 20cm$ accuracy, then a ranging error of $1m$ is unacceptable. For example, if a warehouse

deploys robots to automatically collect items, it is necessary to precisely know where the robot is and where the item is it needs to gather. If the localization error is too high, the robot might bump into other obstacles or it collects the wrong item. In general, for any localization application it is crucial to have as accurate distance measurements as possible, as ranging errors can propogate on any multilateration method used to calculate the exact location of a device. As different materials can block, attenuate, and slow down the travelling signal pulses, there is still an error introduced which needs to be corrected. In LOS measurements, the two ranging modules have a clear sight of each other with no obstacles in the way, so that the first detected signal pulse (first path) is not obstructed. In these situations, the ranging error is $< 10cm$, when the modules are calibrated correctly [15]. In non-LOS (NLOS) measurements, the first path and many of the reflections are attenuated, blocked, or refracted by large objects (which infers a large positive ranging error as signals need to traverse different media), or a reflection is mistaken as the first path when the actual first path is blocked [22]. We also define situations, where small objects attenuate the first path and infer a smaller ranging error as weak-LOS (WLOS) conditions. There are two main approaches to tackle this problem: Either the goal is to identify, detect and ignore incorrect measurements [23] [24], or the goal is to directly mitigate incorrect measurements [10] [11] [24] [25]. The first approach is viable if there are many ranging devices to choose from. However, some infrastructures might not have an overwhelming number of devices to use, and therefore ignoring measurements is not always an option.

Supervised machine learning models are gaining attention to tackle the classification and regression problem [26] [27] [28] [29]. By training on a data set with known reference values, these models can find and learn relations between the input data and the reference value (target) and thereby predict the outcome on a new unseen data set. Two popular models for regression and classification are support vector machines (SVMs) and convolutional neural networks (CNNs), which both classify LOS measurements or directly predict a ranging error. The SVM uses handcrafted features, which are statistical insights extracted from the CIR that are found to be useful. The CNN is fed directly with the magnitudes of the CIR, which does not require any prior knowledge on the signal's properties, and no statistical analysis of the CIR is needed. The ultimate goal is to assess the feasibility and performance of ML models for NLOS classification and mitigation on constrained embedded devices.

1.2 Contributions

This Master thesis lays the foundations and extends the work of Stocker et al. [30]. Specifically, it builds upon mitigation and classification techniques from [10], [11], [24], and [25] and contributes by:

- Creating a data set with larger ranging errors than the papers used as a basis. These papers typically use a data set with small ranging errors or leave out measurements that go through walls (e.g., [11] with a mean ranging error of $12.4cm$).

Our data set includes both WLOS and NLOS cases, such as through-wall measurements and other large objects in between the devices of interest.

- Benchmarking the performance of different classification and mitigation strategies and showing how LOS mitigation can be improved by classifying the measurements beforehand.
- Combining the neural network's mitigation model described in [11] with the output layers of [24] in order to perform classification with the REMNet model.
- Analyzing the mitigation and classification results of our data set. The overall best results are achieved by first classifying measurements, and, only then, performing mitigation on WLOS and NLOS measurements. LOS measurements are only improved by a bias correction, which describes a ranging bias caused by signal level drops over distances using the free space loss model. Thereby, an improvement of the median absolute error of up to 39.4% can be achieved.
- Showing the feasibility of our classification and mitigation solution by running them on an nRF52840 DK with only 1 MB of available ROM and 256 KB of available RAM. When run one after the other, our two models achieve a runtime of 138 ms, and occupy 88 KB of RAM and 270 kB of ROM.

1.3 Outline

This thesis is structured as follows. After providing an overview on UWB technology and on the employed machine learning algorithms in Chapter 2, we explore related literature and explain the methods used to tackle NLOS conditions (both classification and mitigation) in Chapter 3. The methods from some of these papers serve as the basis for the modifications found in this Master thesis. Chapter 4 then explains in detail our measurement setup deployed to collect the training data and the modifications we performed on the methods from the related literature. It further evaluates the results of different variants of NLOS classification and mitigation methods we used on our data set. Chapter 5 uses the most suitable classification and mitigation models found to derive an implementation capable of running on a constrained device. Moreover, we also perform a validation of the performance of a new test set. Finally, Chapter 6 concludes the findings in this Master thesis and describes possible future work.

2 Background

In this chapter, we provide some background information on the concepts used in this Master thesis. Section 2.1 explains the technology behind UWB and provides additional details on how UWB can create accurate distance measurements by overcoming some hurdles. In Section 2.2, we describe the hardware and software that is being used to achieve precise distance measurements. Section 2.3 then describes different machine learning techniques that are used in this paper to classify and mitigate erroneous distance measurements. Section 2.4 explains the procedure to calculate the principal component analysis, which we use to analyze our data set.

2.1 Ultra-Wideband Technology

This section describes the main features of UWB, the technology behind sending and receiving UWB packets, and how these packets can be used to accurately estimate the distance between two devices.

2.1.1 IEEE 802.15.4 Standard

IEEE 802.15.4 standard [31] compliant UWB radios use time-modulated impulse radio signals at a bandwidth greater than $500MHz$, which lead to a sharper pulse in the time domain compared to the bandwidth used in narrow-band technologies, for example, Wi-Fi with $20 - 80MHz$ bandwidth [20]. Compared to them, UWB pulses are easier to distinguish from reflected pulses, as can be seen in Fig. 2.1. The top left image shows the interference of a narrow-band pulse and its reflection in the time domain, while the bottom left image shows the same but with a UWB pulse. It is evident that UWB pulses are subject to less interference due to their high bandwidth. As is shown in the figure, the point of detection at the threshold is shifted and attenuated for the narrow-band pulse, due to the interference of the direct pulse and the reflection. For the UWB pulses, there is hardly any interference, which leads to a more accurate point of detection. The two images on the right of Fig. 2.1 show the narrow-band and UWB pulses with added noise. The top right image shows the narrow-band pulse with noise, which makes it hard to determine the exact point of detection. On the contrary, the detection of a UWB pulse is hardly affected by the noise, as can be seen in the bottom right image. During a UWB communication, the initiating device sends many of these low-energy radio pulses at a constant frequency that need to be accumulated by the receiver in order to be detected. Due to this method of signal detection, any signal pulse that does not repeat itself at the same frequency will not have constructive interference. UWB systems have noise-like signals, as each UWB pulse uses such a high bandwidth and is

only allowed to have a limited transmission power [32]. Even though UWB, by design, should inherently be robust against and should not interfere with narrow-band signals, a study by Chiani et al. [33] has shown that in some cases a narrow-band device and a UWB device can affect the signal to interference ratio. This can negatively impact the link between two narrow-band devices or two UWB devices. UWB uses frequencies from the ISM band from $3 - 10GHz$ and has 16 different available channels [31].

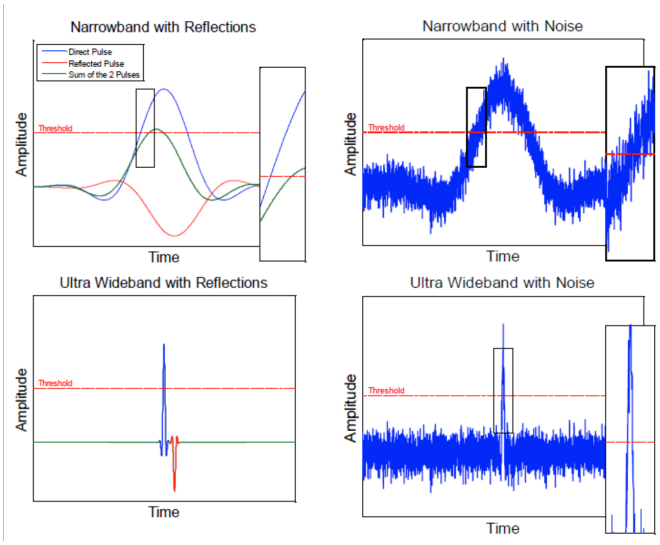


Figure 2.1: Comparison of an ultra wideband (UWB) and a narrowband (NB) pulse in the time domain [1].

2.1.2 UWB Physical Layer

The UWB physical layer, as described by the IEEE 802.15.4 standard [31], is comprised of two fields, the synchronization header (SHR) and the data portion. The SHR field is further comprised of the preamble and the start of frame delimiter (SFD) as is depicted in Fig. 2.2, and the data portion is comprised of the physical header (PHR) and the payload.
The preamble is used for packet detection and node synchronization. The SFD separates the preamble from the next fields and marks the start of the data portion with a timestamp, which is further used for the ranging procedures as described in Sect. 2.1.5. The timestamp during transmission is constructed from the raw clock counter marked at the beginning of the PHR transmission plus the added antenna delay that is calibrated to the device. The antenna delay needs to be accounted for, as the signal needs some time to leave the antenna that should not be counted towards the time of flight. The timestamp during reception is based on the start of reception of the SFD. The antenna delay is subtracted from the timestamp as the receiver antenna does not count towards the measured time of flight. Additionally, this timestamp is added with the correction factor determined by the leading edge detection algorithm. This needs to be done, as

small multipath component replicas can appear in the accumulator span, due to the clock offset between two devices. In order to not falsely identify one of these replicas as the first path, the detection threshold is artificially raised. If the indices of the detected first path and the strongest path are too far apart, the algorithm assumes that this is not a LOS measurement, and that the real first path might have been missed. If this is the case, the detection threshold is lowered and the leading edge detection algorithm iteratively searches for the real first detected amplitude within a time window. [3] [31] The PHR field contains information regarding the following data field, for example, the data length and the data rate. The data field contains the payload, which is constrained to 127 bytes by the standard. However, non-standard payload lengths do exist.

2.1.3 UWB Modulation

The PHR and data field are modulated via a burst position modulation (BPM) binary phase-shift keying (BPSK) scheme to convey information. This means that the information is determined by the positions of the bursts and a guard interval between the burst intervals stops potential interference of bursts as can be seen in the right half of the Fig. 2.2 [3]. Furthermore, the PHR and data field have a forward error correction (FEC) included in the form of a 6 bit single-error-correct double-error-detect (SECDED) code and a Reed Solomon (RS) code, respectively. The data field can be sent at a data rate of $110kbps$, $850kbps$, $6.8Mbps$, or $27Mbps$; however, a 0.87 factor of reduction on the actual symbol rate needs to be considered when using the RS code.

The SHR field does not use BPM/BPSK but instead sends a sequence of pulses at a pulse repetition frequency (PRF) of either $16MHz$ or $64MHz$. The bit can be a -1, 0, or 1 by setting the phase of the pulse, or by not sending a pulse at all as is seen in the left half of Fig. 2.2. The time it takes to send one symbol t_{ps} depends on the time needed to send one pulse t_p and on the number of time intervals needed for one symbol N, which is influenced by the chosen PRF and leads to the formula $t_{ps} = N * t_p$. The full time it takes to send the preamble is then defined by the number of preamble symbol repetitions (PSR), which defines the length of the preamble with each symbol being sent with time t_{ps}. This results in a preamble transmission time of $t_{psr} = PSR * t_{ps}$. The IEEE 802.15.4 UWB standard defines a PSR of 16, 64, 1024, and 4096. The DW1000 does not support a PSR of 16, however, other arbitrary PSR values may be used.

2.1.4 Channel Impulse Response (CIR)

The channel impulse response (CIR) depicts the magnitude and timing of the first path and each detected reflection as can be seen in Fig. 2.4, thus characterizing the surroundings of the receiver. A great advantage of UWB technology is that due to the precise time resolution, the direct path and all the reflections are distinguishable from one another in the CIR. An example of an attenuated direct path and a reflection that are present in the CIR is shown in Fig. 2.3. The first path denotes the first detection of the pulse by the receiver and is used to determine the time of arrival (ToA) that is needed

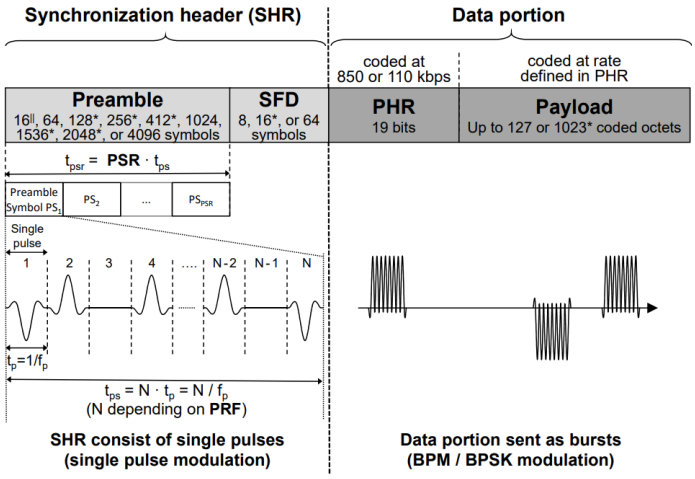


Figure 2.2: **UWB PHY frame structure** [2]. This diagram shows the different modulation schemes used for the SHR and the data portion, as well as their respective transmission rates. Lengths marked with $*$ are available on the popular DW1000 radio and || are not available on the DW1000.

to calculate the time of flight (ToF) of the signal and, ultimately, the distance between the devices. Since obstacles can attenuate or block the transmitted signal pulses, the CIR can have different characteristica as is shown in Fig. 2.4. The figure shows the absolute values of three CIRs in a LOS, WLOS, and NLOS scenario over a period of time. The CIR has been shifted to the left to place the detected first path index at

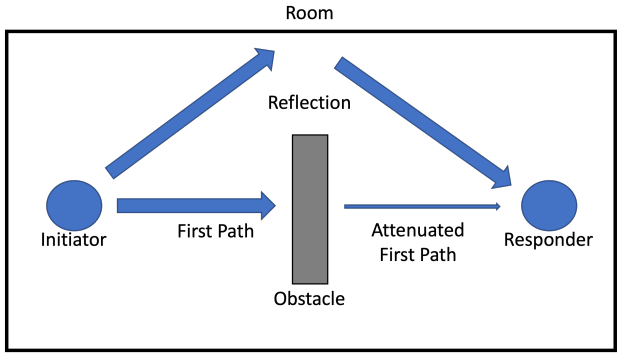


Figure 2.3: Example setup of an attenuated first path due to an obstacle and a reflection between two ranging devices in a room.

$20ns$ for all three cases, as the receiver does not detect the first path at the same point in time. Additionally, only 80 out of the 1016 samples of the CIR are plotted, as these contain the most useful information. Fig. 2.4 shows the differences between the three cases. For LOS conditions the by far highest peak is also the first path, which is easy to

detect. For WLOS conditions an obstacle can attenuate the first path, which can make detection harder. In the figure, the first path of the CIR captured in WLOS conditions is not the one with the highest amplitude, and the peak amplitudes are not too different. For NLOS conditions, the first path might be completely blocked, which means that the detected first path might be a reflection. As can be seen in the figure, the real first path might be at around 10 ns; however, the UWB receiver did not detect it. Fortunately, the multipath components (MPCs) can also be analyzed together with the first path to obtain insights into the reliability and quality of the measurement. These insights can be used to classify NLOS measurements or to correct the erroneous measurement to some degree with the help of SVMs and neural networks. Furthermore, as is shown by Großwindhager et al. [19], the CIR and a crude floor map can be used to localize a tag with only one anchor node with a worst case average of 34cm in LOS cases.

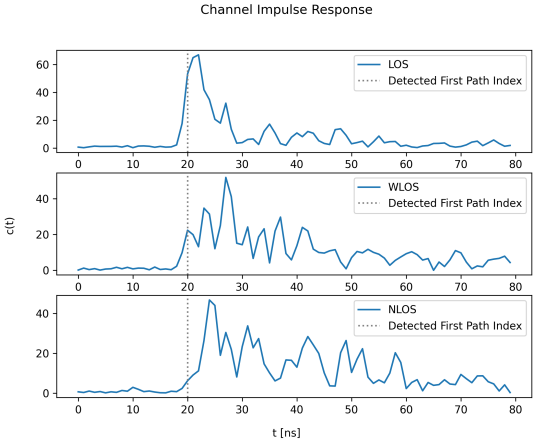


Figure 2.4: **CIRs for LOS, WLOS and NLOS cases.** The detected first path index is shifted to be at 20ns. Only 80 out of the 1016 samples of the CIRs are plotted in order to better compare the three cases.

2.1.5 ToF Estimation

The time of flight (ToF) is used to calculate the distance over which the radio pulses are travelling through the air. A simple method to do this is the single-sided two-way ranging (SS-TWR) algorithm, which measures the round-trip delay of a message from an initiator to a responder and back again, as can be seen in Fig. 2.5. Each node waits the same amount of time before replying, which results in a time T_{round} and T_{reply} with which the time of flight $\hat{T}_{prop}$ can be estimated. By sending two messages, the clock offset of the two devices can be overcome.

$$\hat{T}_{prop} = \frac{1}{2}(T_{round} - T_{reply}) \tag{2.1}$$

The double-sided two-way ranging (DS-TWR) algorithm is based on the SS-TWR

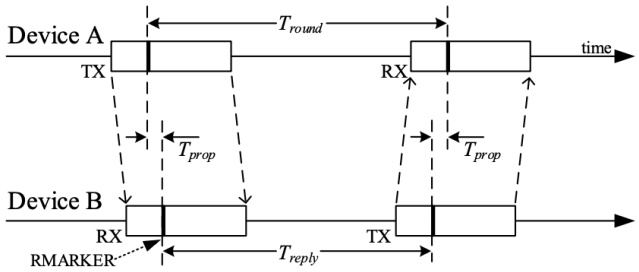


Figure 2.5: Single sided two-way ranging with two messages [3].

algorithm and results in a more accurate ranging estimation, as it also rules out the clock drift when calculating the ToF. The clock drift describes the difference between two device clocks as these do not run at the same speed. The higher the clock drift the bigger the ToF error. DS-TWR uses the same principle, but uses two round-trips, one initiated by each node, to estimate the time of flight. Additionally, the start of the second round trip can be combined with the reply of the first round trip, which results in a total of three messages between the initiator and the responder as is shown in Fig. 2.6. By doing this twice, the responder node can accurately estimate $\hat{T}_{prop}$ and rule out any effect of the clock drift [4] [34].

$$\hat{T}_{prop} = \frac{T_{round1} \cdot T_{round2} - T_{reply1} \cdot T_{reply2}}{T_{round1} + T_{round2} + T_{reply1} + T_{reply2}} \tag{2.2}$$

The result is then simply sent to the initiating node.

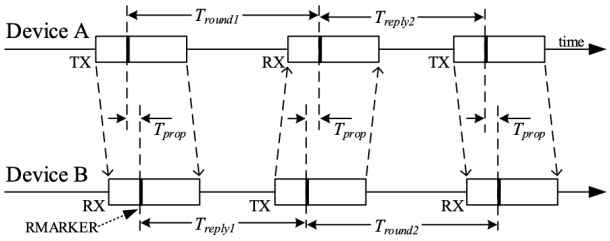


Figure 2.6: Double sided two-way ranging with three messages [3].

2.1.6 Bias Correction

An evaluation by Decawave [4] has shown that, due to the antenna gain, transmitter power, and all other gain or loss parameters in the system, the time of flight (ToF) is affected by the received signal level. The received signal level is calculated based on the free space loss model that describes how the signal level drops over distance in LOS conditions. This adds a positive or negative bias to the corresponding ranging measurement

as can be seen in Fig. 2.7. In order to compensate for this range bias, Decawave, one of the first manufacturers of UWB hardware, has taken range measurements with different pulse repetition frequencies (PRFs) at different distances and fitted a polynomial to these biases, one for each PRF. They supply the corresponding polynomial coefficients with their software and, by evaluating this polynomial at a given received signal level, we receive the range bias to subtract from or add to the measured range. The bias correction is only viable for LOS measurements, as obstacles influence the received signal level.

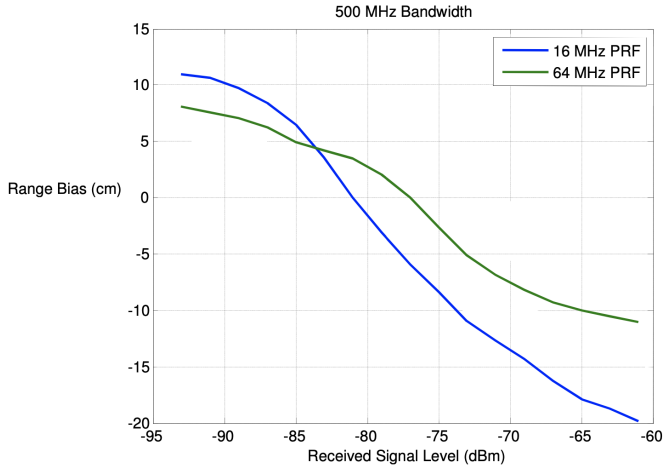


Figure 2.7: **Bias correction**. The range bias compared to the received signal level [4].

2.2 UWB Hardware and Software

This section briefly describes the hardware and the operating system used for the ranging measurements as well as an important calibration step to receive accurate results along with an analytical classification method.

2.2.1 DW1001 Module and DWM1001-DEV Board

The DW1001 module hosts the DW1000 UWB transceiver with the corresponding antenna, alongside a $64MHz$ Nordic NRF52832 ARM Cortex M4 and a BLE radio [35]. The DW1000 transceiver was designed for the IEEE 802.15.4-2011 UWB standard and is a fully integrated low power, single chip CMOS radio transceiver that achieves sub $10cm$ ranging accuracy [3]. The chip communicates with the DW1001 module via the SPI and it also comes with power management in mind. In fact, it offers a SLEEP and a DEEPSLEEP state, which both allow to switch off certain parts of the transceiver, whereas the SLEEP state also implements periodic checks to check if something needs to be sent or received.

The DW1001 module can be used with a variety of microcontrollers, but Decawave also offers it as a part of the DWM1001-DEV board as can be seen in Fig. 2.8. The board adds functionality by supporting both UWB and Bluetooth, as well as by introducing an on-board J-link for debugging and flashing via USB. Furthermore, it contains a position and networking stack (PANS) firmware to enable the creation of UWB based real-time location services (RTLS) applications, and includes battery connectors and USB ports as power sources [15].

Figure 2.8: Image of the DWM1001-DEV board and the inserted DW1001 module.

2.2.2 DWM1001-DEV Board Calibration

An important calibration of the DWM1001-DEV board needs to take place before using it for any ranging application, as the antenna delay creates a large offset during the calculation of the ToF (as the radio pulse travels slower in the transmitter and receiver antenna than in air). If not taken into consideration, this delay corresponds to a ranging estimate of about $160m$ at a set distance of $2m$ apart for our modules. For this reason, the antenna delay needs to be taken into account and subtracted from the ToF for the transmitter and the receiver antenna. The default value suggested for the DWM1001-DEV modules leaves our modules with an offset of about $30cm$. In order to reduce this offset even further, we calibrated the modules according to the application notes by Decawave [36]; however, this algorithm only estimates the antenna delay for each module and did not improve our offset. For this reason, we used a trial and error approach and set up the modules at a distance of 4 metres, and changed the antenna delay until the offset was close to $0cm$.

2.2.3 DWM1001-DEV Board Operating System

The Mynewt operating system [37] used in this Master thesis is a real-time cross-platform OS that is compatible with a variety of microcontrollers. It contains network stacks for BLE, Wi-Fi, IPv6, etc., and has its own build system with a packet manager. The

Mynewt OS is very modular, and the `mynewt-dw1000-apps` distribution supplies many demos for getting started. The `mynewt-dw1000-core` distribution contains the drivers and libraries needed to set up and configure the DW1000 impulse radio and to build a connection between two devices.

2.2.4 Decawave (DW) LOS Estimation

The `mynewt-dw1000-core` library provides a simple classifier using two metrics extracted from the CIR to classify measurement samples. This approach is also verified by the research of Gururaj et al. [3]. It compares the received signal strength indicator (RSSI) with the first path power level (FPPL) and provides an estimation of whether the sample belongs to a LOS or NLOS measurement. LOS measurements tend to have a power difference $< 6dB$, as the first path contains the most power compared to the rest of the multipaths. NLOS measurements, instead, tend to have a power difference $> 10dB$, as the first path is attenuated or blocked and does not contain the most power in relation to the other multipath components. The formula is $RSSI - FPPL$, where the RSSI is calculated with

$$RSSI = 10 \cdot log_{10}(\frac{C \cdot 2^{17}}{N^2}) - A \tag{2.3}$$

and the FPPL is calculated with

$$FPPL = 10 \cdot log_{10}(\frac{F_1^2 + F_2^2 + F_3^2}{N^2}) - A. \tag{2.4}$$

The value C is the power of the CIR reported by the DW1000, the value A is either 113.77 or 121.74 for a PRF of $16MHz$ or $64MHz$, respectively, and the value N is the preamble accumulation count. The values F1, F2, and F3 are reported magnitudes, measured during the search for the first path, by the leading edge detection algorithm.

This classification is defined as

LOS if:

$$(RSSI - FPPL) < 6 \tag{2.5}$$

NLOS if:

$$(RSSI - FPPL) > 10 \tag{2.6}$$

.

Anything in between LOS or NLOS is not defined by Decawave.

2.3 Machine Learning Techniques

Channel statistics provide a good insight into LOS and NLOS measurements. However, they cannot solve all problems and accurately classify or correct measurements. For this reason, machine learning is an interesting approach and some techniques are described in the following subsections.

2.3.1 Support Vector Machine

The support vector machine (SVM) is a machine learning model that tries to predict future values based on the data it has trained on, hence making it a supervised learning model. The SVM is popular due to its fast training compared to neural networks, good performance even with little training data, and small model size as it only needs its so-called support vectors to predict a value and not all of the training data points [38]. A support vector is a training data point that is used to predict a value. SVMs can be used to classify data using a support vector classifier (SVC) or perform regression (SVR) and receive a continuous-valued number. The data is described as $(\mathbf{x}_1, y_1), ..., (\mathbf{x}_m, y_m)$ where $\mathbf{x}_i \in R^N$is a feature vector. For the SVC $y_i \in \{-1, +1\}$ is the target classification and for the SVR $y_i \in R$ is a continuous target value with $i = 1...N$.

Support Vector Classification

SVCs are the most common application for SVMs and classify given input data according to what they have learnt from previous training data. An SVC tries to find a margin that maximizes the separation of two bounding hyperplanes, as this allows for a robust and general classification accuracy when seeing new data, as can be seen in Fig. 2.9. The data points that lie on a hyperplane are then called support vectors. The data to be trained on is generally categorized as either linearly separable or non-linear. In case of the latter, a kernel needs to be used that transforms the input data into a higher dimensional feature space in which the data is linearly separable as is shown in Fig. 2.10 [39].

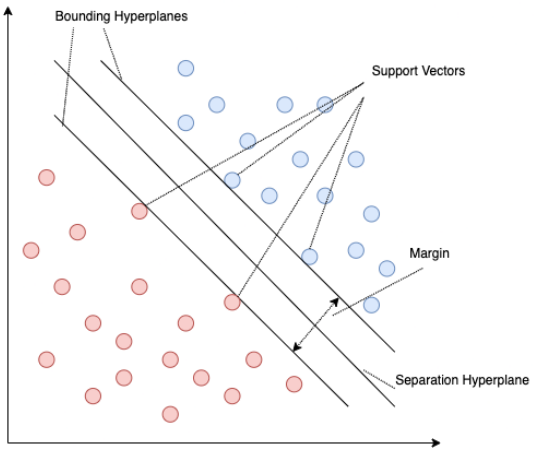


Figure 2.9: An example of a separation hyperplane and its bounding hyperplanes to form the maximized margin to separate the classes. The points on the bounding hyperplanes are called support vectors.

A popular kernel used for non-linear data is the radial basis function, which is defined as

$$\Phi(\mathbf{x}, \mathbf{x}') = exp(-\frac{1}{2}||\mathbf{x} - \mathbf{x}'||_2^2) \tag{2.7}$$

with $\mathbf{x}'$ being the feature vector of a new measurement.

The margin is a linear function $f(\mathbf{x}) = (\mathbf{x} \cdot \mathbf{w}) + b$, for $\mathbf{w} \in R^N, b \in R$ separating the data points and the maximum separation of the hyperplanes can be achieved by

$$Minimize : \Phi = \frac{1}{2}||\mathbf{w}||^2 + C\sum_{i=1}^{N}\xi_i \tag{2.8}$$

$$Subject\ to : y_i((\mathbf{w} \cdot \mathbf{x}_i) + b) \geq 1 - \xi_i, i = 1, ..., m \tag{2.9}$$

The newly introduced slack variable $\xi \geq 0$ allows the SVC to misclassify data, which allows the model to find a more general solution. The regularization variable C controls this trade-off between model complexity and misclassification. This optimization is solved with Lagrange multipliers $\alpha \geq 0$ and results in a decision function of

$$f(\mathbf{x}) = sign(\sum_{i=1}^{m} y_i\alpha_i\Phi(\mathbf{x}_i, \mathbf{x}) + b). \tag{2.10}$$

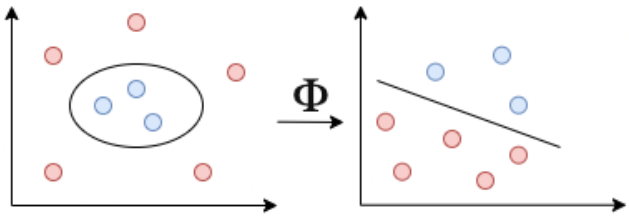


Figure 2.10: An example of finding a hyperplane to separate non-linear data by first performing a kernel transformation, where the data can be split with a straight line.

Support Vector Regression

The support vector regression (SVR) operates similarly to the SVC, however, the SVR tries to find the flattest tube (margin) that contains most of the training data points, while also minimizing the prediction error between a continuous prediction value and the continuous target value. The tube is formed with a hyperplane and the ϵ value. The ϵ describes the acceptable error a prediction can have in relation to the target value and spans around the hyperplane, as is shown in Fig. 2.11. The ϵ insensitive loss function penalizes testing data points that have an error greater than ϵ and ignores testing data points within the ϵ-tube. The choice of this hyperparamter determines the number of support vectors (and, consequently, the size of the model) as well as the accuracy of the predictions. In order for the SVR to be more robust against outliers, the parameter ξ,

seen in Fig. 2.11, defines the number of outliers that are allowed to be outside of the ϵ-tube during training [38] [40].

The SVR is constrained by the same optimization problem as the SVC and, after solving this problem, by introducing the Lagrange multiplier α (where only support vectors will have an $\alpha > 0$ and all other data points have an $\alpha = 0$).

The regression function is thereby described as [41]:

$$f(\mathbf{x}) = \sum_{i=0}^{m} (\alpha_i \Phi(\mathbf{x}, \mathbf{x}_i) + b). \tag{2.11}$$

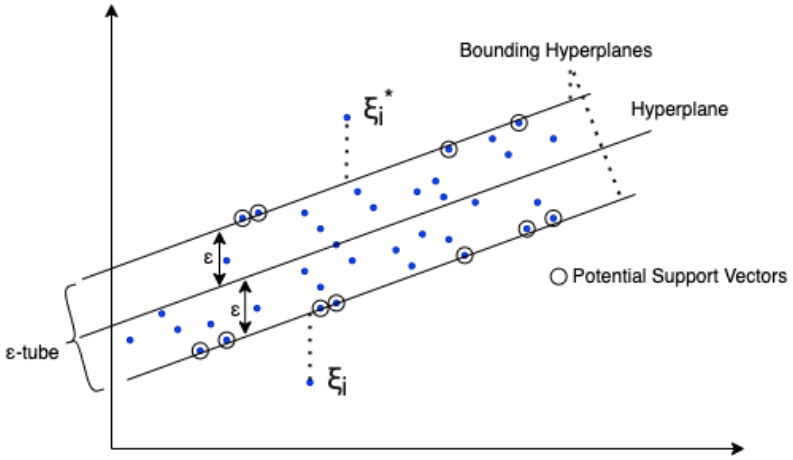


Figure 2.11: **Example of one dimensional SVR training points**. The ϵ-tube spans around the hyperplane that tries to minimize the distance of the predicted output to all the real output of the training data points. Data points close to the bounding hyperplanes are support vectors. The ξ parameter defines the two points as outliers.

2.3.2 Neural Networks

An artificial neural network (ANN) mimics the functions of the brain and similarly, it uses multiple connected and weighted neurons in order to perform complex computations. A single artificial neuron as shown in Fig. 2.12 is comprised of the number of inputs $\mathbf{x} \in R^N$ and each input's weight $\mathbf{w} \in R^N$. The inputs, their weights and an initial bias b ($x_0 = 1$, $w_0 = b$) are combined via $a = \sum_{i=0}^{N} w_i x_i = \mathbf{w}^T \mathbf{x}$ and fed into an activation function (f) which decides the output $z = f(a)$ of a neuron. In a biological neuron the inputs $\mathbf{x}$ correspond to the dendrites, the weights, bias and activation function belong to the cell body that houses the nucleus, and the outputs correspond to the axons [5].

A NN is comprised of different layers, which are made up of several connected and weighted neurons. The connections between neurons simulate synapses between biolog-

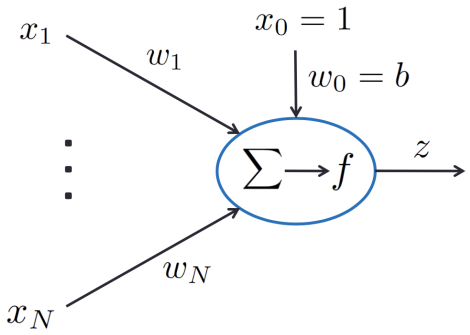


Figure 2.12: **Artificial Neuron** [5]. This figure shows the input vector $\mathbf{x}$, the weight vector $\mathbf{w}$, the biases $(x_0 = 1, w_0 = b)$, the activation function f, and the output $z = f(\mathbf{w}^T\mathbf{x})$.

ical neurons. The first layer is called the input layer, and the last layer is called the output layer. These layers are the only ones with which the user interacts. In between, there are different hidden layers that form the complexity of the NN and decide what happens with the data. Additionally, different methods can be used to improve the accuracy of the model and to avoid under- or overfitting, which will be addressed in this section. An advantage of neural networks is the ability to automatically extract features by themselves. This allows the NN to learn from complex structures without the need of defining features by hand. NNs can find patterns that the user might miss or cannot see due to the large amount of data. A NN is also a supervised learning system that trains on known data to predict the provided target values. The input data can either be handcrafted features like in the case of SVMs or it can be raw unprocessed data where the NN will find its own features. The only preprocessing that is often used and is beneficial to the learning of a NN is to have a 0-mean and unit variance in the data set. This ensures that all values have the same scale and that no data points have a more significant impact on the output simply because of their numeric values.

Activation Function

The activation function of each neuron determines the output and is responsible for the overall performance of the NN. Typically, a trade-off has to be made between the complexity a neuron should have and the performance the overall NN needs. Different activation functions perform better in different situations. For a regression model, the output activation function could be a rectifying linear unit (ReLU) [11], as is shown in Fig. 2.13, which is calculated with the formula

$$R(x) = max(0, x) \tag{2.12}$$

that defines inputs < 0 as 0 and keeps the other inputs the same.

For a classification model, the output activation function could be a softmax activation

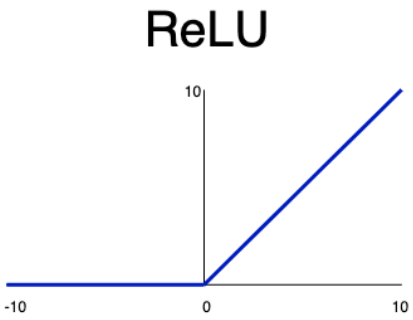


Figure 2.13: Rectified linear unit (ReLU) activation function.

function [24] that is usually used as the activation function for the output layer and calculates the individual probability for each class to form a sum equal to 1. The output neuron with the highest probability defines the predicted class of the total n classes and the formula is:

$$S(x_i) = \frac{exp(x_i)}{\sum_{j=i}^{n} exp(x_j)}. \tag{2.13}$$

Loss Function

In order to learn, the NN has to know what it is doing wrong and minimize the errors. By training the NN with target values, the loss function can calculate the error between the predicted and the target value for a given input. Each axis depicts an element of the input vector and creates a loss landscape, as can be seen in Fig. 2.14. The goal is to find the minimal error between the predicted and the target value, which means that we try to find the global minimum of the loss landscape. To find this global minimum we use gradient descent. Gradient descent is an iterative algorithm that calculates the positive gradient of the slope at a random starting point on the loss landscape and moves towards the next point in the negative, opposite direction of the gradient to find the smaller error, as is shown in Fig. 2.14. By iteratively moving towards smaller error points on the loss landscape, a local or global minimum can be found. The loss landscape can have more than one minima and depending on the starting point of the gradient descent algorithm, different minima might be reached in training.

An example of a popular loss function is the sum of squared errors where the error of a particular sample $E^{(i)}$ is a function of the predicted output $\hat{y}$ and the target value y for all possible number of outputs K [5].

$$E^{(i)} = \frac{1}{2} \sum_{n=0}^{K} (\hat{y}_k - y_k)^2 \tag{2.14}$$

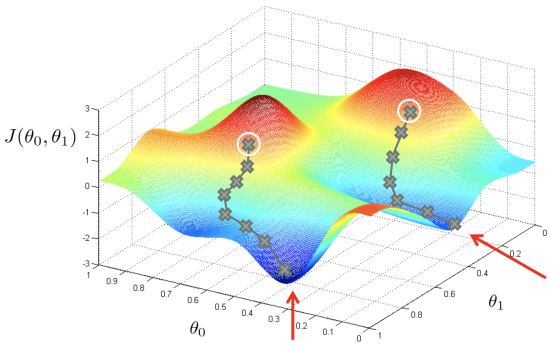


Figure 2.14: **Loss function**: Examples of a loss function of some arbitrary model $J(\theta_0, \theta_1)$ with input features θ_0 and θ_1 [5]. Depending on the initial start, different minima can be reached.

Batch Learning

The batch size determines how many input samples are analyzed with gradient descent before updating the weights of each neuron in the neural network. This affects how the model learns, as frequent updates of the weights take longer to process and, asthe gradient descent algorithm oscillates more as each training sample has a bigger impact [42]. There are three common batch sizes:

- Batch gradient descent (batch size = number of samples): Analyzes the full batch or all training samples before updating its weights. This method leads to higher memory consumption but also to faster processing speeds, as the weights are not updated as often. Additionally, it leads to a smoother trajectory of the gradient descent as outliers do not have a great impact.

- Mini-batch gradient descent (1 < batch size < number of samples): Takes a subset or mini-batches of the training samples and updates the weights after analyzing a mini-batch. This method reduces the memory consumption and often improves the accuracy of the neural network as each individual training sample has a bigger impact. The computation time rises as the weights need to be updated more often.

- Stochastic gradient descent (batch size = 1): Updates the weights after every training sample, which leads to the highest processing time. However, it needs the least amount of memory, as it only needs to hold one sample in memory at a time. The gradient descent algorithm oscillates more as each training sample affects the gradient, which can lead to slower convergence times. However, each training sample has a bigger impact on the weight updates.

When working with a GPU it is advisable to select a batch size of $2, 4, 8, 16, 32, ...$, as this maximizes the processing efficiency.

Optimizer

The optimizer updates the weights in a neural network through back-propagation and determines how fast and how well a network can learn. A popular optimizer is the adaptive moments (ADAM) optimizer [43], which is a modification of the stochastic gradient descent algorithm.

- Learning rate η: It determines the step size of the optimizer when searching for the minima of the loss function. Large learning rates correspond to faster learning, but can result in missing or not reaching minima as can be seen in Fig. 2.15 on the right. Instead, small learning rates are more accurate in finding minima,but they can get stuck in local minima or take excessive time to reach the minima as is shown in Fig. 2.15 on the left and in the center. The ADAM optimizer adjusts the learning rate during training.

- β_1: Is the exponential decay rate for the first moment estimate, which is the mean of the gradient in a selected window [43].

- β_2: Is the exponential decay rate for the second moment estimate, which is the uncentered variance of the gradient in a selected window [43].

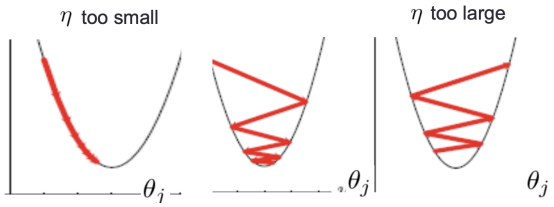


Figure 2.15: **Learning rates**: Examples of too small learning rates and too large learning rates that have a hard time finding the minima [5].

Back-Propagation

The weight update is propagated to each of the neuron's weights through back-propagation. The algorithm starts from the output layer and propagates the partial derivative of the error to the corresponding neurons in the prior layer. This way, each neuron receives its corresponding weight update to minimize the predicted error [5] [43] [44].

Under- and Over Fitting

When training a NN, the question arises when an optimal solution has been found and when to stop training a network. If the training loss reaches a minimum, the network might work exceptionally well on the training data but will perform terribly on new

unseen data. Fig. 2.16 shows a classification example of two classes. The green line perfectly separates the red and blue training data. However, new unseen training data close to the separation line might be predicted incorrectly or an outlier of the training data wrongly influences the prediction of new data. On the other hand, the brown line poorly classifies the two classes and the NN has not trained enough and is underfitting. The black line adequately separates the two classes and will probably perform better on new unseen data.

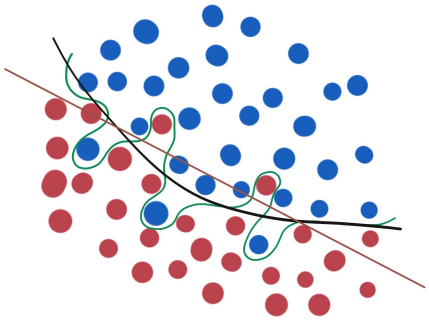


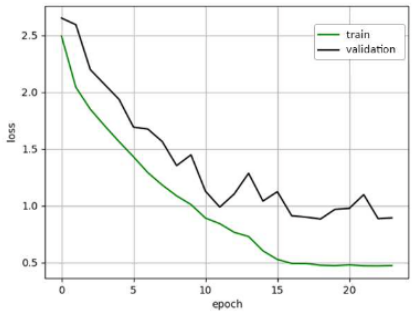
Figure 2.16: **Over and underfitting**. The image shows example data points that are fit adequatly (black line), overfit (green line) and underfit (brown line).

Early Stopping

One method to decrease the effect of under- and overfitting is early stopping, where the idea is to split the training data set into another data set called the validation set. While training the model on the training data set, the model is also validated on the validation set in terms of the validation loss. As soon as the validation loss does not decrease anymore, the model stops training even though the loss on the training data is still decreasing. An additional parameter called patience can help to overcome local minima, as can be seen in Fig. 2.17 by only stopping training when the validation loss has not decreased within several epochs. Therefore, by early stopping the NN based on the validation loss, the network can prevent overfitting and performs better on new unseen data.

Convolutional Neural Network

A popular NN is the convolutional neural network (CNN), which works particularly well with complex and non-spatially dependent data, for example, the position of an object in a picture. It reduces the number of parameters of a typical NN by using alternating convolutional and pooling layers that automatically extract relevant features or reduce the data size respectively. A convolution operation multiplies a region of the input with a defined kernel and the sum is taken as the output as is shown in Fig. 2.18. The pooling layer takes either the maximum or an average over an area of the input neurons as is shown in Fig. 2.19. This makes CNNs especially performant for image and audio



.

Figure 2.17: This example shows the training and the validation loss over time. While the training loss keeps decreasing, the validation loss can start to rise again [6]

recognition as the position of the object in the image does not contribute to the result. The same principle translates to UWB ranging as the CIR is spatially independent and contains a lot of information that is neglectable [45].

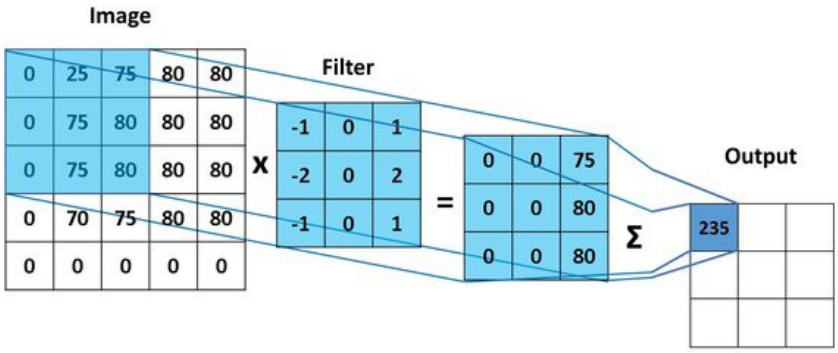


Figure 2.18: **Convolution**. Example operation of a convolution step with a filter of size 3 x 3 on an image [7].

The convolutional layer drastically reduces the complexity of the NN as the main idea is to not connect all neurons with each other but to only connect a few neurons to a neuron in the next layer. In terms of image processing, this means that a neuron in the next layer only receives a small area of the image as an input. The convolution of the neurons representing this small area filters out any dominant features and passes them on to the next layer. An average- or max- pooling layer then further filters the remaining data and reduces the complexity by taking the average or the maximum of an area of the previous layer. By repeating these two main ideas, the CNN manages to quickly process large input data and achieves good accuracy for classification and regression tasks [45].

The difference between classification and regression is determined by the output layer of the CNN and by which activation function is chosen.

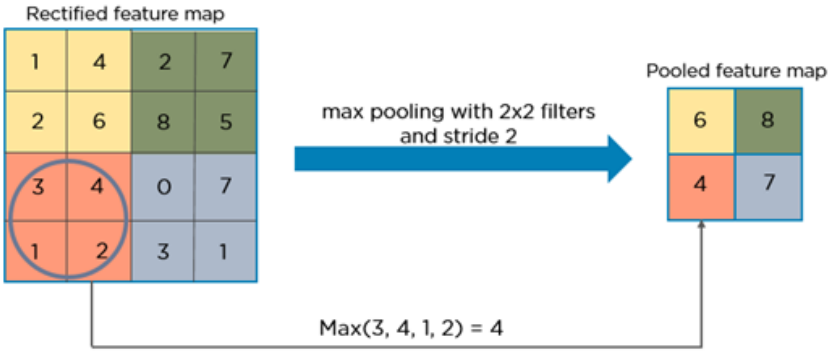


Figure 2.19: **Max Pooling Example**. Example operation of a max pooling step with a filter of size 2 x 2 and a stride 2 of an output of a ReLu activation function [7].

2.3.3 K-Fold

K-fold cross validation is used to test how general and consistent the predictions are while training and testing a ML technique with limited data.

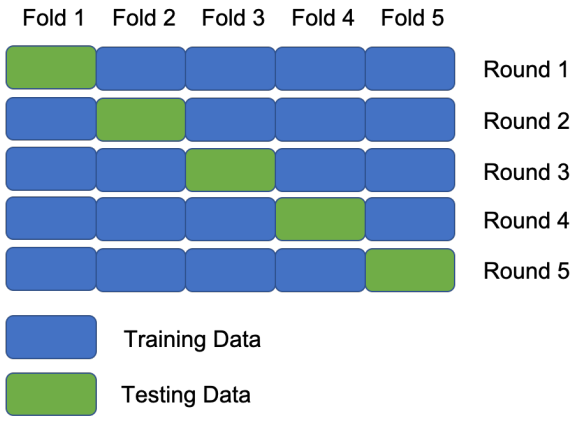


Figure 2.20: An example of a 5-fold split on a data set.

It is commonly used on analytical machine learning models, for example, for support vector machines but also for small neural networks. Too large neural networks can lead to infeasible run times. The idea is to separate a data set into equal folds where the last fold contains the remaining data that cannot be split evenly. For each run, one of these folds is used as a test set, while all the other folds are used as a training set. The test results are saved, the model discarded, and a new model with the same parameters is trained with the new training set and the new test set until all folds have been used as a test set. In the end, the results of all runs can show if the model performs consistently well given different training and testing data at each run. Fig. 2.20 depicts an example of a 5-fold split of a data set.

2.4 Principal Component Analysis

Principal component analysis (PCA) is a method to break down high dimensional data and to increase interpretability while minimizing information loss. Preferably, the high dimensional data is broken down into two principal components that define the data, and simplify a classification or regression task [46]. This reduces the dimensionality of the features that define the data and can make seemingly correlating data distinguishable. The input feature vector is a list of defining traits and the output is a list of the principal components. The flowchart of the algorithm is shown in Fig. 2.21, but it essentially reduces the feature vectors down to usually 2 principal components (PCs), where the first PC is the most weighted, the second PC the second most weighted, and so on.

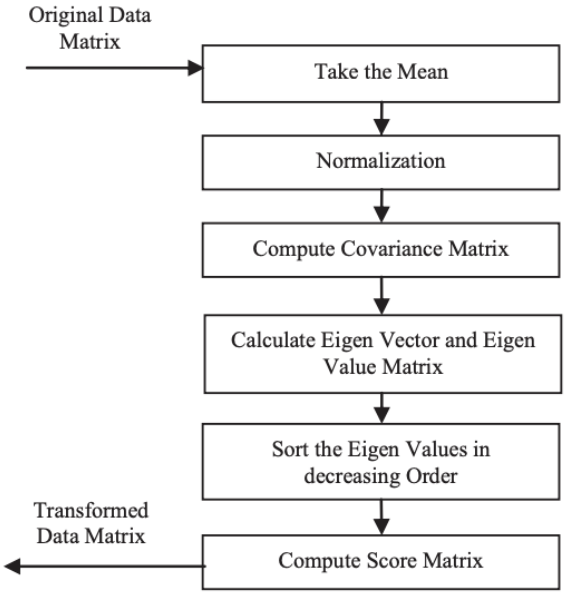


Figure 2.21: **PCA algorithm**. The input matrix is a series of feature vectors and the output matrix is a series of principal components [8].

The first step is to scale the values via normalization and to then compute the covariance matrix. In the covariance matrix, all diagonal elements represent variance and all the non-diagonal elements represent covariance. The goal is to have minimal covariance and maximal variance as small variance could indicate unimportant data. This is achieved by computing a diagonal matrix, where all values are zero except for the diagonal elements. We then compute the eigenvectors that characterize the matrix entries and sort them in descending order as they represent the relative importance of the principal components. This ranks the principal components and leaves us with an output matrix with the most important information [8]. In this Master thesis the PCA is used to analyse the correlation between LOS, WLOS, and NLOS data, and to check, whether a classification of all three conditions is meaningful.

3 Related Work

In this chapter, we review existing work and methods to tackle classification and mitigation problems of UWB distance measurements. In the first section, we review classification and mitigation methodologies without the help of machine learning models, and in the second section, we review classification and mitigation approaches with machine learning models. We further provide a more in-depth overview of the methodologies that we used in this Master thesis.

3.1 Survey of NLOS Classification and Mitigation Techniques

A survey by Khodjaev et al. [9] categorized NLOS classification and mitigation techniques without using machine learning. The authors created a taxonomy, shown in Fig. 3.1, which broadly separates existing approaches into an identification / detection or a mitigation category. When classifying NLOS data, this can be done through the variance of range estimates, by using channel statistics, or by using a map of the surroundings to estimate positions. To mitigate NLOS data, either the first path detection can be improved, or statistics algorithms can be used to reduce the ranging error.

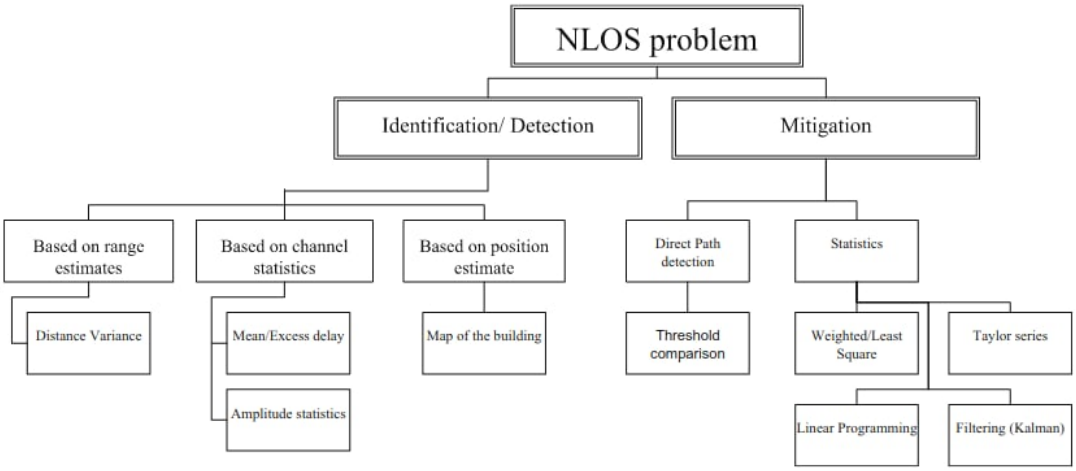


Figure 3.1: Taxonomy of solutions proposed to solve the NLOS problem without machine learning [9].

3.1.1 Mitigation Based on Position Estimates

The method proposed by Yung-Hoon et al. [47] uses a floor map and reference nodes to identify NLOS conditions and uses ray tracing to account for errors. These ray traces are used to calculate the excess time of flight that the signal needs to travel through different materials. They simulate a grid on a given floor map with the known materials and calculate all possible excess delay combinations between every grid point. They then use reference nodes and the floor map to identify NLOS situations and to correct the excess time of flight with their simulated ray traces.

Limitations: The methodology described in [47] needs a lot of preprocessing to simulate all the ray traces. It also needs a floor map with a list of the most present materials.

3.1.2 Mitigation Based on Direct Path Detection

Obstacles that block the LOS can attenuate the first path, which can lead to a false first path detection within the UWB receiver, which in turn, leads to a delayed ToA estimate. The mitigation method formulated by Wu et al. [48] attempts to correct the direct path of NLOS measurements, thereby improving the distance estimation. This is done by an iterative correlation peak search in the CIR before the detected strongest path. The search stops if no amplitude greater than a defined threshold is found within a given time window. This method is very similar to the leading edge detection deployed by Decawave.

Limitations: The algorithm to find the first path is an iterative one, which needs some time to process. Furthermore, an obstacle can fully block the first path, which means that it cannot be found with this method.

3.1.3 Mitigation Based on Statistics

Alavi and Pahlavan [49] take the measured distances and the used channel bandwidths as inputs and provide an estimated ranging error as output. They create a model that combines the knowledge of two types of errors. The first is the multi-path error that typically occurs in WLOS scenarios when the first path is attenuated. The second is the undetected first path error that typically occurs in NLOS scenarios when the first path is blocked and a multi-path is mistaken for a first path. Both of these error models try to describe the occurring ranging error based on empirical measurement results.

Limitations: The proposed methodology bases their models on only two features gathered from two measurement campaigns. This could lead to location specific results, as different materials and surroundings have a great effect on the range estimates.

3.1.4 Classification Based on Range Estimates

Borràs et al. [50] use the variance of the range estimates to decide if a LOS or a NLOS situation is present. This is based on the fact that NLOS measurements have a

higher variance, due to the influence of obstacles. Most of these methods use a gaussian probability model to estimate the likelihood ratio, which is compared to a threshold.

Limitations: The thresholds are determined by prior knowledge of the variances of LOS and NLOS ranging data in specific areas. Furthermore, these methods require several observations to determine the mean and standard deviation before classifying a measurement.

3.1.5 Classification Based on Channel Statistics

Gururaj et al. [51] classify LOS and NLOS data based on channel statistics. They compare two methods, one being the root mean squared (RMS) delay spread obtained by the CIR and the other being the received signal power and the first path power that they extract from the CIR. The RMS delay spread is calculated from the CIR amplitude $r(t)$ of each time step t with the formula

$$\tau_{RMS} = \int_T (t - \tau_{MED})^2 \frac{|r(t)|^2}{\int_T |r(t)|^2 dt} dt \tag{3.1}$$

where τ_{MED} is the mean excess delay that is defined as

$$\tau_{MED} = \int_T \frac{t|r(t)|^2}{\int_T |r(t)|^2 dt} dt. \tag{3.2}$$

The threshold to classify a measurement as LOS and NLOS is environment specific and is not specified by the authors, nevertheless, they achieve a classification accuracy of 96.3%.

The second approach involves using the received signal power performances and is the same methodology described in Sect. 2.2.4 that is used in the software provided by Decawave, where the LOS classification is defined as $RSSI - FFPL < 6$. The idea is that the received signal power and the first path power correlate differently depending on if it is LOS or NLOS. The authors claim to reach a classification accuracy of 96.69% and that the calculation only takes around $0.25ms$ compared to the extraction of the RMS delay spread, which takes $2 - 3s$.

Limitations: As the RMS delay spread requires an arbitrarily chosen threshold, it is not as suitable for off-the-shelf defices and needs readjustment when changing environments. This makes it more unpractical and less reliable for widespread use.

On the other hand, the power performance analysis with the RSSI and the FPPL is very promising and we do achieve an accuracy on our dataset of 78.5% (Tab. 4.3c).

There are multiple ways to extract relevant information needed to classify measurements. The simple ones which do not need any kind of machine learning analyze the distance variance, channel statistics like the mean excess delay, or need prior knowledge of the surroundings to detect if a measurement is useful or not.

3.2 Survey of NLOS Classification and Mitigation Techniques using Machine Learning

Nessa et al. [28] have conducted a survey of machine learning techniques to tackle indoor positioning problems. These problems are divided into classification and mitigation problems. In this section, we will provide a brief overview of a few of them and go into further detail into the ones used as a basis in this Master thesis.

3.2.1 Classification Based on Expectation Maximization - Gaussian Mixture Models

Fan and Awan [52] propose an unsupervised machine learning approach to classify NLOS measurements. This means that the training data is not labeled with a target value. The proposed method uses Gaussian mixture models and the expectation maximization algorithm to predict the class. The Gaussian mixture models combine the probability distributions of each extracted feature. The three extracted features used are the number of paths containing more than 85% of the received signal, the mean excess delay, and the root mean squared delay spread. The expectation maximization algorithm then tries to find the maximum likelihood for the LOS and NLOS gaussian mixture models. The authors claim to classify LOS and NLOS data with an accuracy of 86.5%.

Limitations: The proposed method has a hard time classifying outliers that loosely fit into one of the two clusters.

3.2.2 Classification Based on the Random Forest Algorithm

Ramadan et al. [53] use a random forest algorithm to classify LOS and NLOS data. A random forest is a collection of decision trees that each classify input data and where the majority predicted class is selected. A decision tree is constructed of decision nodes and leaf nodes. Every decision node looks at one of the features and determines which of the two following nodes should be taken. Leaf nodes contain the target class prediction, hence making it a supervised learning model. The training data is used to create the decision nodes, which guide the data towards a prediction leaf node. As one decision tree is very sensitive to input data, random forest uses multiple decision trees to reduce the prediction variance. The features used are the mean amplitude of a normalized CIR, the standard deviation, the skewness, and the kurtosis. The authors claim a classification accuracy for LOS of 95%, and for NLOS of 97.3%.

Limitations: The random forest algorithm seems to be very good for classification, however, it is not used for mitigation. Other machine learning algorithms can be used for classification and regression, which makes them more interesting for this Master thesis.

3.2.3 Classification and Mitigation Based on Convolutional Neural Networks

Bregar and Mohorčič [24] introduce a convolution neural network for NLOS identification and regression. The two models are both constructed out of alternating convolution

and pooling layers that are able to extract features directly from the channel impulse response. This means that they do not require prior knowledge of the environment, nor need to preprocess the channel impulse response to extract features by hand. The only difference between the classification and the regression model is the output layer. The classification model uses a softmax activation function on two output neurons to calculate the probability for each neuron. One neuron represents LOS, while the other one represents NLOS; together they sum up to 1. The higher probability neuron decides the classification. For the regression model, the output layer consists of one neuron using a linear activation function to predict a continuous value. The authours conducted a measurement campaign in two offices and used one of the rooms as the training and one as the test set. Their CNN classifier achieves an F1 score of 87.6%, and they claim to achieve better localization results by adding their error predictions. Additionally, the authors tested their CNN on devices with constrained resources, such as the Raspberry Pi 1 with 512 MB of RAM. The other devices used have more available RAM. The CNN models are developed with the TensorFlow library and the same models are used on their devices with constrained resources.

Limitations: The devices they used have the capability of running the full TensorFlow library, while we want to run our models on microcontrollers with equal to or less than $1MB$ of flash and RAM. For our goals, we need to convert the TensorFlow models to TensorFlow Lite Micro models that run on microcontrollers, as shown in Chapter 5.

3.2.4 Mitigation Based on Support Vector Machines

In 2010 Maranò, Gifford, and Wymeersh [54] conducted a measurement campaign and proposed a least squares SVM to classify and mitigate ranging measurements. Based on this work, Wymeersh, Maranò, and Gifford [25] in 2012 further developed two more regressors, one being the support vector machine and one being the Gaussian processes (GP). As a data set, they used the same data that was collected in 2010 with 1024 individual measurements, whereby 512 are LOS and 512 are NLOS. While conducting the measurement campaign, they analyzed the data and found some striking features. The features they used are the basis for the ones we use in this thesis and the formulas are described in Tab. 4.2 (1 to 7). They observed an overall lower energy in NLOS compared to LOS as the signal is obstructed. The LOS measurements have a shorter rise time of the first detected path than NLOS as nothing attenuates the signal. The root mean square (RMS) delay spread is higher in NLOS compared to LOS. The RMS delay spread describes the temporal dispersion of the energy detected in the CIR that is caused by reflections and in NLOS the first path and the reflections tend to have similar energy levels leading to a higher RMS delay spread.

Both the SVM and the GP approach have similar results and correct large error measurements by up to $3m$.

Limitations: This approach also introduces a large mitigation error of up to $2m$ for some measurements that only have an initial ranging error of $< 20cm$. For our data set, we did not encounter this problem. However, the model increases the residual ranging error of our unobstructed LOS measurements.

3.2.5 Mitigation Based on Convolutional Neural Networks

Angarano et al. [10] [11] have developed two deep neural networks (DNNs) to mitigate UWB ranging errors using regression models.

In [11], their focus was to create a good convolutional neural network that can directly mitigate the residual ranging error. Contrary to non-parametric regressors like the SVM and the GP, a CNN does not need handcrafted features but can learn features by itself. Therefore, the authors can directly feed the CIR as the input to the CNN and let the model extract its own features. They created a data set of 55000 samples from one small room ($5m$ x $3.5m$), one medium room ($5m$ x $5m$), one large room ($10m$ x $5m$), and an outdoor setting. The proposed CNN is a series of convolutional layers that extract features with batch normalization layers in between to normalize the values before each calculation step. At the end, they use a global average pooling layer to extract the features with the greatest impact and use two fully connected layers to receive a prediction on the residual ranging error. With this CNN model, they achieve an improvement of 54% for their whole data set with an initial mean absolute error of $12cm$.

Limitations: The proposed CNN reduces the mean absolute error by 45%. The model also works well for our WLOS and NLOS data; however, this CNN model increases our ranging error of the unobstructed LOS measurements in the process. For this reason, we also inspected the successor model of the same authors [10], which is also designed to have a smaller footprint, and to classify data prior to performing regression.

In [10], their focus was to construct an energy efficient and fast neural network to mitigate ranging errors, which they call Ranging Error Mitigation Network (REMNet). The deep neural network they came up with is a residual reduction module (RRM) based on a residual neural network and convolutional layers based on a convolutional neural network as can be seen in Fig. 4.7. This RRM allows for efficient and simplified learning by filtering out unimportant data and reducing the parameters in each run as they feed the channel impulse response (CIR) directly into the network and the features are learned by the network itself. The authors conducted a measurement campaign including three different sized rooms, measurements through the wall of the rooms, and an outdoor area. They conclude that the model learns similar features in all three rooms which creates a broad and general use-case for the model. The measurements which were conducted outdoors are found to contribute negatively, as there are very few multipaths that could contribute to the model. For their experiments, they only use LOS and NLOS scenarios to train and to test, and leave out through-wall measurements and outdoor measurements. From our perspective, their NLOS cases are similar to our WLOS cases and their through-wall cases are similar to our NLOS cases. This suggests that they are focusing on measurements that do not have as large ranging errors and are not suitable to mitigate measurements through a wall. The model does not explicitly classify CIR traces. Instead, the REMNet implicitly classifies data by design and predicts the ranging error.

Limitations: The data set used for their evaluation omits through-wall measurements, whereas our data set contains them.

4 Experimental Campaign

In the following sections, we discuss the setup that was used for the measurement campaign, the performance metrics used to evaluate the machine learning models, the methodologies used to build the models, and the results. Our goal is to mitigate erroneous ranging estimates by using a performant machine learning model which has a small footprint that can be used on low power microcontrollers with constrained resources, and that does not worsen standard LOS measurements. To achieve the best results we use a combination of classification and regression approaches.

4.1 Setup

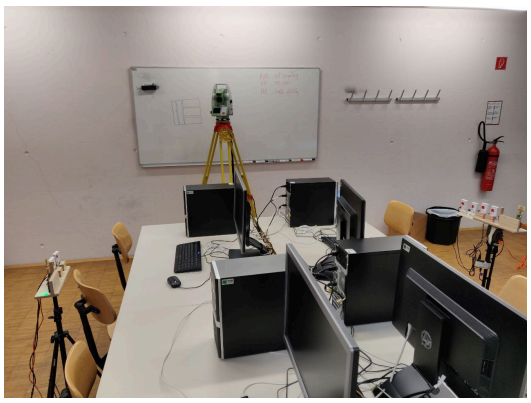
This section explains the exact setup and parameters used to perform the measurement campaign. We created a new data set including small error LOS measurements (median absolute error of $2.6cm$), where no obstacles block the transmitted signal pulses between two ranging devices, medium error WLOS measurements (median absolute error of $17cm$), where small obstacles attenuate the direct signal pulses but most reflections are not influenced, and high error NLOS measurements (median absolute error of $68cm$) where the ranging devices are separated by walls or other large objects which strongly attenuate or block the direct path. Therefore, our data set contains bigger errors than other papers, which use a smaller ranging error, for example, [11] with a mean ranging error of $12.4cm$.

4.1.1 Measurement Campaign

Our measurement campaign consists of 749 individual measurement points spread across 12 different locations inside the university building at distances between the initiators and responders of $0.5m - 13m$. The locations include five IT-laboratories, three hallways with intersections, a workshop, two small rooms, and one study centre as can be seen in Figs. 7.1, 7.2, and 7.3. The set of measurements in every scenario are split up evenly into LOS, WLOS, and NLOS conditions, and care is taken to limit any movement during the measurements. A measurement is classified as WLOS, when the direct path is blocked by a small object that typically only attenuates the signals. In NLOS conditions large obstacles (e.g., wall, corner of two walls) can potentially fully block the signals passing through.

Obstacles: As our measurement campaign aims to perform well in office environments, the obstacles are basic office materials. For the WLOS measurements, the obstacles are comprised of wooden office chairs, monitors (Fig. 4.1a), computers, plant pots, metal shelves, people, vending machines, wooden cabinets, wooden desk spacers, thin

glass walls, thin and small concrete railings with a height of $\sim 1.2m$ and open metal doors. In the case of the NLOS measurements, we measured through one or more thick concrete walls (Fig. 4.1b), solid glass walls, closed metal doors, and elevators alongside any obstacles used for WLOS which are also in the vicinity.



(a) Example setup of a WLOS measurement through monitors.

(b) Example setup of a NLOS measurement through the corner of concrete walls. Note that the modules have been slightly turned towards the camera for the picture. For all measurements the modules face each other.

Figure 4.1: Example setup for WLOS and NLOS.

4.1.2 Measurement Devices

We used six DWM1001-DEV modules by Decawave, shown in Fig. 2.8, to acquire the data set. A Leica TS11 tacheometer (Fig. 4.1a, 4.1b) provided the real distance as a reference value.

4.1.3 Measurement Parameters

Each measurement is performed on two sets of modules. One set consists of 3 Decawave DWM1001-DEV modules using channel 5 and the other set using channel 7. These two channels share the same centre frequency but use a different bandwidth, as can be seen in Table. 4.1. To prevent module dependent errors, we used two responders per channel, which alternatingly respond to a single initiator device. Only two modules were active during a measurement, while the other modules were silent. The evaluation results in this chapter only refer to the modules operating on channel 5 with a bandwidth of $500MHz$. The results of the modules operating on channel 7 follow the same trends.

Parameters	Module Set 1	Module Set 2
Channel	5	7
Centre Frequency (MHz)	6489.6	6489.6
Bandwidth (MHz)	499.2	1081.6
Preamble Codes (64 MHz PRF)	9	17
Antenna Gain (dB)	1	1
Tx Power (dBm)	-14.4	-14.4
Antenna Delay (ns)	515.3	513.6

Table 4.1: Parameters used for the Decawave DWM1001-DEV modules.

4.1.4 Data Analysis

Preamble Accumulation Count

UWB receivers correlate the preamble with a template signal and sum up the results, as explained in Section 2.1, and the DWM1001-DEV board saves them in the accumulator CIR memory [3]. Therefore, the final CIRs need to be normalized by the preamble accumulation count provided by the DWM1001-DEV board to calculate the original signal strength of the CIR. All CIRs used in this Master thesis have been normalized by the preamble accumulation count.

Uniform Data Set

For any further analysis, a uniform subset of 450 individual measurement points of the data set was used. This ensured that the same amount of LOS/WLOS/NLOS data at similar distances was used and that no bias was introduced. This means that our subset contains the same amount of LOS, WLOS, and NLOS measurements at around $1m, 2m...10m$. Fig. 4.2 shows the whole data set (bottom) compared to the uniform subset (top) of the data set. The histograms have a bin size of $1m$ except for the first and last bin, which have a size of $1.5m$.

Five recordings of each 50 sample measurement point are taken for training. The median ranging error of our LOS measurements for the whole data set and the uniform data set is $2.6cm$ and $2.8cm$, for the WLOS measurements $16.8cm$ and $17.9cm$, and for the NLOS measurements $67.8cm$ and $69.3cm$ respectively. Further analysis has been done with the uniform data set; however, the results correlate when using the whole data set.

Principle Component Analysis (PCA)

We also calculated the PCA twice with two different input matrices to find out if the feature dimensionality could be reduced, and to see if there is a correlation between LOS, WLOS, and NLOS data. Ideally, the three cases would be separable in 2D with a straight line, as this suggests three distinct cases. The first PCA (Fig. 4.3a) calculates

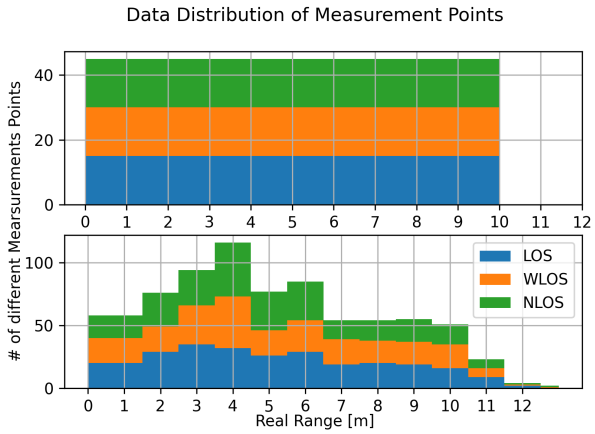


Figure 4.2: **Data distribution** of the whole measurement campaign (bottom) and the uniform subset (top) used for our analysis. This corresponds to a total of 749 measurement points and 450 uniformly distributed measurement points.

its principle components from the extracted features shown in Table 4.2 while the second PCA (Fig. 4.3b) is calculated from the CIRs.

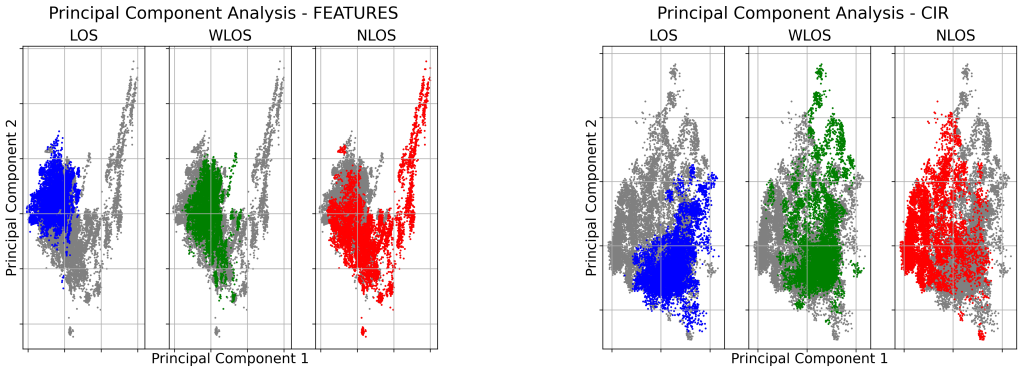
The PCA, as can be seen in Figs. 4.3a and 4.3b, does not make the three cases LOS, WLOS, and NLOS distinguishable in 2D with a straight line, as the points overlap with each other. However, there is a noticeable trend that groups the three cases. In Fig. 4.3a, the LOS cases are accumulated in the top left corner, the WLOS around the centre, and the NLOS cases are primarily at the bottom and on the right side. Similarly, but opposite in Fig. 4.3b, the LOS cases are accumulated in the bottom right, the WLOS is again centred around the middle, and the NLOS cases are primarily on the left side. This also correlates to the overlapping ranging errors that can be found in the data set of each case and suggests that a classification of all three cases is difficult. Therefore, we only classify LOS and NLOS measurements, as these are the easiest to distinguish and impact the ranging error the most. A further observation is that the PCA comprised of the extracted features is also more densely populated than the PCA comprised of the CIR.

4.2 Performance Metrics

In this section, we will briefly discuss the performance metrics we use to measure the quality of classification and mitigation methods.

4.2.1 Median Absolute Error (MAE)

We use the MAE to determine how much the residual ranging error has increased or decreased after our mitigation. Since the median is more resilient to outliers, we use it instead of the mean. The MAE is calculated by sorting the absolute residual ranging



(a) PCA comprised of extracted features.

(b) PCA comprised of the CIR.

Figure 4.3: PCA results with two principal components.

error ($|RRE|$) of each distance estimate and taking the value in the middle. For an even number of elements n, this is done by

$$MAE_{even} = \frac{1}{2}(\frac{|RRE_n|}{2} + \frac{|RRE_n|}{2} + 1) \tag{4.1}$$

and for an odd number of elements n

$$MAE_{odd} = \frac{|RRE_{n+1}|}{2} \tag{4.2}$$

.

4.2.2 90^{th} Percentile Absolute Residual Ranging Error ($|RRE|$)

The 90^{th} percentile $|RRE|$ is the value of the CDF plot at the 90% mark. It supplies a good benchmark, where 90 percent of all mitigated measurements have a smaller $|RRE|$. In order to calculate the 90^{th} percentile, the list of $|RRE|$s has to be sorted in ascending order. The percentile P of each index i of the total list length N can be calculated by

$$P = \frac{i}{N} \cdot 100 \tag{4.3}$$

and vice versa the index of a specific percentile can be found by

$$i = \frac{P}{100} * N \tag{4.4}$$

which is used to find the RRE at the given percentile.

4.2.3 F1-score

The F1-score measures the performance of the classifications and combines the two metrics precision and recall to form a harmonic mean within a value range from 0 to 1, where the latter is the optimal score.

$$F1 = \frac{2 \cdot (precision \cdot recall)}{(precision + recall)} \tag{4.5}$$

The precision is determined by the fraction of the true positives (T_p) over the sum of the true positives and the false positives (F_p). In terms of a classification, this represents the fraction of correct classifications regarding the overall classifications of a single class.

$$precision = \frac{T_p}{T_p + F_p} \tag{4.6}$$

The recall is determined by the fraction of the true positives over the sum of the true positives and the false negatives (F_n). In terms of a classification, this represents the fraction of correct classifications regarding the total occurrence of a single class.

$$recall = \frac{T_p}{T_p + F_n} \tag{4.7}$$

4.2.4 R^2-score

The R^2-score measures the prediction accuracy, i.e., determines how well the prediction matches the true error. The best score is a R^2-score of one. However, the score can also be arbitrarily worse. The symbol u is the residual sum of squares and v is the total sum of squares and they are made up of the true residual ranging error y_{true} and the predicted residual ranging error y_{pred}.

$$R^2 = 1 - \frac{u}{v} \tag{4.8}$$

$$u = \sum (y_{true} - y_{pred})^2 \tag{4.9}$$

$$v = \sum (y_{true} - \mu_{y_{true}})^2 \tag{4.10}$$

4.2.5 K-Fold

In our case, we use a 10-fold cross validation and stack all the prediction results to receive a smooth CDF plot. As we use 5 samples of each measurement between two points, we take care not to have samples of the same measurement in the training and in the testing set. The training features for the SVR are always comprised of all measurement cases (LOS/WLOS/NLOS), while the evaluation is performed separately, either only on one case or on all cases. The F1- and R^2-score are calculated individually in each run, and the mean and variance are taken as the end result.

4.3 Decawave (DW) LOS Estimation

Since Decawave does not define conditions in between the LOS and NLOS thresholds, we have modified the DW LOS estimation explained in Sect. 2.2.4. We changed the two thresholds (6 and 10) for LOS, undefined, and NLOS to a single threshold of 8.5. The threshold of 8.5 achieves a 7.4% better F1-score than only using the threshold of 6, and it achieves a 1.2% better F1-score than only using a threshold of 10. Therefore, the DW LOS estimation can be used to classify LOS and NLOS measurements to perform different mitigation methods depending on the classification.

LOS:

$$(RSSI - FPPL) < 8.5 \tag{4.11}$$

NLOS:

$$(RSSI - FPPL) >= 8.5 \tag{4.12}$$

4.4 Support Vector Machine

This section describes the structure of the SVM used for classification and regression, explains the features being used, and shows the evaluation results.

4.4.1 SVM Structure

For the support vector machines, we use the Python library *sklearn* version 0.24.0 with the default values, if not specified otherwise in this section. For both the SVR and the SVC, we use a radial basis function (RBF) as the kernel, which transforms the features into a higher dimensional feature space. For the kernel coefficient γ we used `scale`. The kernel is the same one as defined in Eq. 2.7, but with the added kernel coefficient. The `scale` parameter defines the coefficient as the fraction of 1 over the product of the number of features ($n_{features}$) and the variance of the features (σ_{x_m}). It further defines how far the influence of a single training data reaches, i.e., the distance of the training samples to the new sample in the hyperdimensional space. Low γ values have a far influence and high values have a close influence [55].

$$\gamma = \frac{1}{(n_{features} \cdot \sigma_{x_m})} \tag{4.13}$$

$$\Phi(\mathbf{x}, \mathbf{x}') = exp(-\gamma ||\mathbf{x} - \mathbf{x}'||_2^2). \tag{4.14}$$

For the SVR, we also set $\epsilon = 0.1$ in all calculations if not specified differently. For the SVR and SVC, we keep the regularization parameter $C = 1$ as the default.

4.4.2 Features

In Tab. 4.2, we depict the handcrafted features used when training the models. The features taken from [25] and [56] are: energy, maximum amplitude, rise time, mean excess delay, root mean squared delay spread, kurtosis, and the estimated distance. The

energy denotes the total received signal strength of the received CIR. The maximum amplitude describes the part of the CIR with the highest received signal strength. Both the energy and the maximum amplitude are higher for LOS measurements than for WLOS and NLOS measurements at the same distance, as the signal is not attenuated due to obstacles. The rise time measures the time the amplitude takes to rise from a threshold close to the standard deviation of the CIR to a threshold close to the maximum amplitude of the CIR. For LOS cases, this value should be small, however, for WLOS and especially NLOS, the maximum amplitude is not necessarily the first path, which leads to a longer rise time. The mean excess delay describes the first moment of the power delay profile, which is the temporal mean of the signal energies detected. The closer together the maximum amplitude and the reflection's amplitudes are, the higher the mean excess delay is. Typically, this occurs when the first path is attenuated, or blocked. The root mean squared delay spread denotes the second moment of the power

Name	**Equation**
Energy	$\varepsilon_r = \int_T \lvert r(t)\rvert^2 dt$
Maximum Amplitude	$r_{max} = \max_t \lvert r(t)\rvert$
Rise Time[a]	$t_{rise} = t_H - t_L$
Mean Excess Delay[b]	$\tau_{MED} = \int_T t\psi(t)dt$
RMS Delay Spread[c]	$\tau_{RMS} = \int_T (t - \tau_{MED})^2 \psi(t)dt$
Kurtosis[d]	$\kappa = \frac{1}{\sigma^4_{\lvert r\rvert} T} \int_T (\lvert r(t)\rvert - \mu_{\lvert r\rvert})^4 dt$
Estimated Distance	d
Pre FP-index Variance	$\sigma_{fp_{idx}}$
DW LOS Estimation	RSSI - FPPL

[a] $t_L = min\{t : |r(t)| \geq \alpha\sigma_n\}$ and
$t_H = min\{t : |r(t)| \geq \beta r_{max}\}$,
with σ being the standard deviation of the thermal noise and
$\alpha = 6$ and $\beta = 0.6$ according to [25].
[b,c] $\psi(t) = |r(t)|^2/\varepsilon_r$.
[d] $\mu_{|r|} = \frac{1}{T}\int_T |r(t)|dt$ and
$\sigma^2_{|r|} = \frac{1}{T}\int_T (|r(t)| - \mu_{|r|})^2 dt$.

Table 4.2: Extracted features from a CIR.

delay profile, showing the temporal spread of the energy from reflections in relation to the mean excess delay. The higher the temporal spread of the energy, the more similar the amplitudes are for the first path and the multi-paths. The kurtosis describes the distribution of energy in the CIR. The distribution for LOS measurements is shifted to the left, as the first path tends to contain the most energy. The last feature is the estimated distance that puts the other features into context, for example, the energy and maximum amplitude are dependent on the distance between the devices and could be similar on large range LOS measurements and short range NLOS measurements. We also added two more features that we developed. One of them is the variance of the small window before the first path index (pre FP-index variance), as this region in the CIR differs between LOS, WLOS, and NLOS situations, and is crucial in the detection of the first path or in finding missed first paths. The window size is $10ns$ since we shift our CIR values until the first path index is at $10ns$. As can be seen in Fig. 2.4, in LOS cases the first path is easily detectable, as no obstacles interfere with the UWB pulses and there is not a lot of variance from the CIR's amplitudes before the first path. For WLOS, the first path does not necessarily have the highest amplitude and, in extreme cases, the first path is not detected at all while the reflections are more or less unaffected. Similarly, in NLOS cases, the first path is strongly attenuated. At the same time, the reflections are attenuated or blocked, which makes the first path detection harder and can lead to a high variance in amplitude around the detected first path. The second added feature is the continuous value of the adaptation of the Decawave (DW) LOS estimation as described in Sect. 4.3.

The calculated features are then normalized to aid the SVM to weight all features equally. We have tried the common standard scaler, however, we achieved best results with the min-max scaling method even though this method is more prone to outliers. The min-max scaling is calculated by taking the maximum and the minimum of a feature vector $\mathbf{x}_i$.

$$\mathbf{x}_i' = \frac{\mathbf{x}_i - min(\mathbf{x}_i)}{max(\mathbf{x}_i) - min(\mathbf{x}_i)} \tag{4.15}$$

4.4.3 SVM Results

Fig. 4.4 shows the results of three mitigation techniques based on the SVR model compared to the uncorrected residual ranging error. The raw SVR mitigation performs the best in NLOS situations, decently on WLOS, but it worsens the LOS ranging error, which is not acceptable. Therefore, we use the DW LOS estimation and the SVC to classify the traces prior to mitigating them. If the classification predicts a LOS case, we only perform a bias correction as explained in Sect. 2.1.6, if not, then we mitigate the measurements with the regression prediction. By doing so, we still achieve good results on WLOS and NLOS data, however, we also manage to increase the LOS performance. As can be seen in Tab. 4.3c, the SVC outperforms the DW LOS estimation and achieves a higher F1 score (85.1% vs 73.6%), which means the SVC detects more accurately LOS conditions than the DW LOS estimation. However, the DW LOS estimation needs fewer resources and computation time, which makes them both have their legitimacy

depending on the available resources of the used platform. The SVR with the SVC (`SVM with Classifier` line) achieves an overall improvement for all cases of the 90^{th} percentile $|RRE|$ of 22% from $1.331m$ to $1.039m$ and an improvement of the MAE of 7.4% from $0.188m$ to 0.174. A list of all results is shown in Tab. 4.3a.

4.5 Neural Networks

We use two different neural networks developed by Angarano *et al.* [10], [11]. The REMNet is the second one, it is the revised version that has been optimized for devices with constrained resources. Both CNNs receive the first 152 samples of the CIR after the

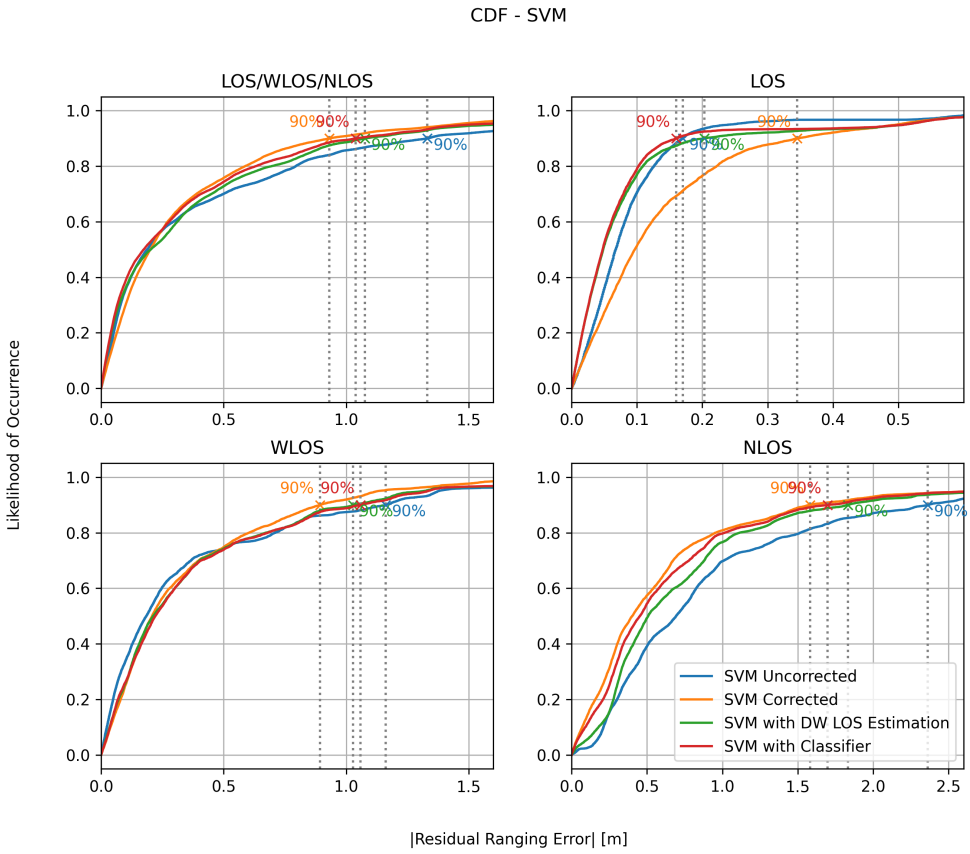


Figure 4.4: **CDF plot of the residual ranging errors using the SVR**. The `SVM Corrected` line describes the raw regression mitigation of the SVR. The `SVM with DW LOS Estimation` and `SVM with Classifier` classify the data with the DW LOS estimation and the SVC, respectively, to either mitigate LOS with the bias correction and NLOS with the SVR.

first path index has been shifted to the 10^{th} position. Since the CIR is accumulated until

a SFD is detected [3], the CIR values are only normalized by the preamble accumulation count to receive the original CIR amplitudes. The neural networks were implemented with the TensorFlow library.

4.5.1 CNN Structure

The CNN proposed in [10] is created to mitigate the ranging error. The structure comprises 16 layers that try to extract features from the CIR and use them to predict a ranging error based on the supplied target data. The first and last layers are the input and output layers with 14 hidden layers in between, as is shown in Fig. 4.5. The first 12 hidden layers are alternating batch normalization (BN) and convolutional layers. The BN layer normalizes the data by creating a zero mean and standard deviation of

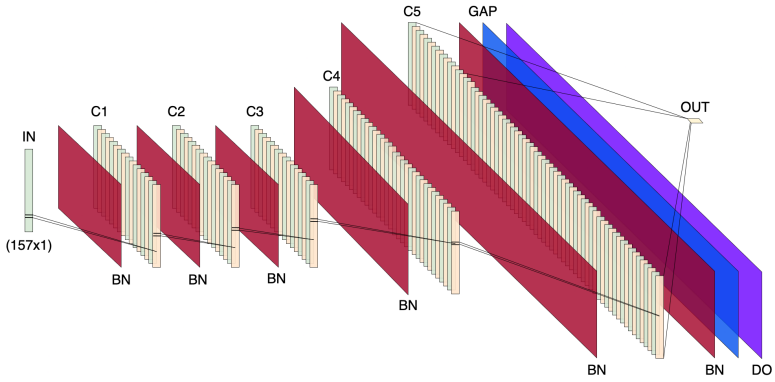


Figure 4.5: **CNN architecture** based on [10]. The layers are batch normalization (BN), convolutional (C1-C5), global average pooling (GAP), dropout (DO) and at the output is a fully connected (FC) layer.

1, while the convolutional layers extract features that correlate to the target training data. Before returning a prediction, a global average pooling (GAP) layer smooths out the data and removes noisy information from the neurons. Then, a dropout (DO) layer randomly drops some neurons, inhibiting them from contributing to the next layer and thereby stopping the CNN from relying too heavily on single neurons and overfitting to the training targets. The DO layer is only used during training and not used for testing. The first fully connected (FC) layer then combines all neurons of one dimension and the second FC layer combines these neurons even further to one output neuron, which results in the predicted ranging error. For our training data, the CNN performed better by skipping the first FC layer and connecting the DO layer's output to the single output neuron. All our calculations were done with this slight modification. As can be seen in Fig. 4.5, the number of neurons used increases with the number of layers, which makes the model less suitable for devices with constrained resources, as each added neuron increases the total number of calculations. Since this CNN model is the basis for the more size optimized REMNet model, we did not modify this CNN to perform classification but only used the DW LOS estimation. For early stopping, we

use a patience of 10 and monitor the validation loss. As batch size, we use 32, which is mini-batch gradient descent.

4.5.2 CNN Results

The raw CNN mitigation performs very well on NLOS data, well on WLOS data, and bad on LOS data (Fig. 4.6). As mentioned before, we should not worsen the LOS ranging errors, therefore, we used the DW LOS estimation to nearly prevent all negative effects on LOS measurements, while improving the WLOS and NLOS measurements. The CNN performs the best but also needs the most resources. The CNN achieves an overall improvement for all cases of the 90^{th} percentile $|RRE|$ of 33% from $1.331m$ to $0.895m$ and an improvement of the MAE of 17% from $0.188m$ to $0.156m$ as can be seen in more detail in Tab. 4.3a.

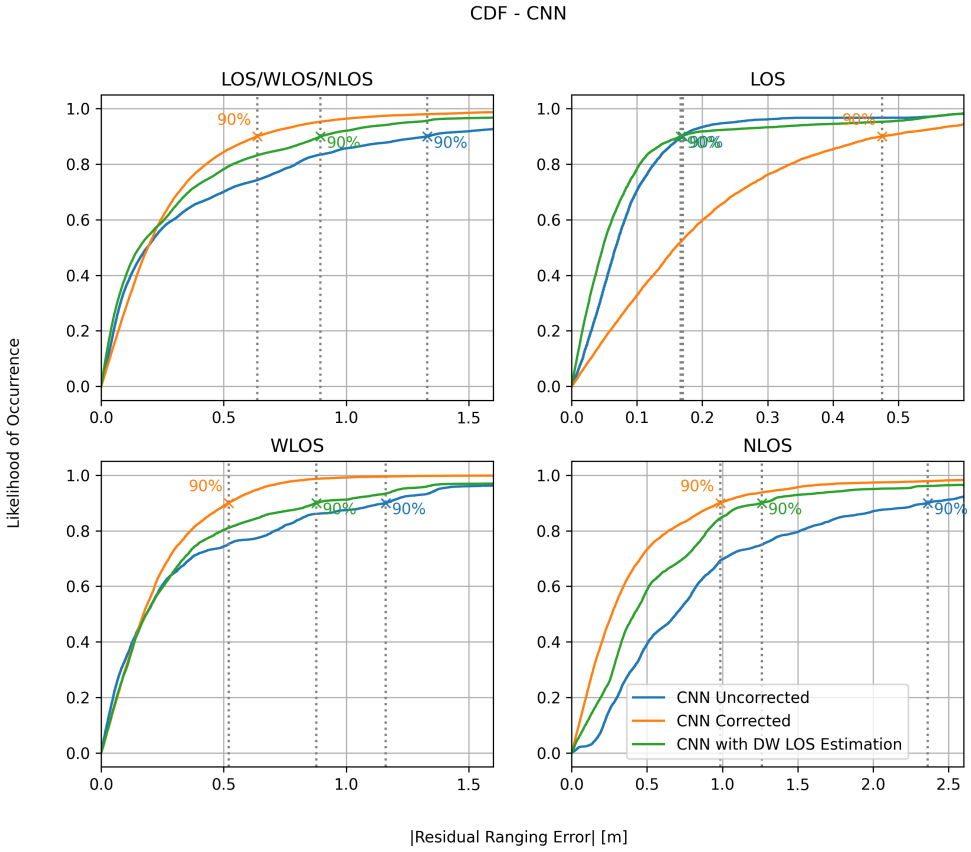


Figure 4.6: **CDF plot of the residual ranging errors**. The `CNN Corrected` line describes the raw regression mitigation of the CNN. The `CNN with DW LOS Estimation` classifies the data with the DW LOS estimation to either mitigate LOS with the bias correction, or the NLOS with the CNN regression.

4.5.3 REMNet Structure

Fig. 4.7 shows the REMNet structure based on the work of [11] and is used in this Master thesis for mitigation. It is developed by the same authors of [10] and the model has been redesigned and optimized in terms of size. It comprises an input layer, a residual reduction module (RRM), and an output layer. The RRM is responsible for learning new features while also reducing the dimensionality, which allows the model to be more suitable for constrained devices. Inside the RRM, the squeeze and excitation (SE) block enhances found features by multiplying itself with a scaling factor derived from itself. In order for the REMNet to run on a microcontroller, we needed to modify the structure. The TensorFlow Lite Micro library is a slimmed down version of the TensorFlow library optimized for microcontrollers. It enables TensorFlow models to be run on a device with constrained resources. However, this has its limitations and the architecture is adjusted. Instead of using a 1D convolution for our 1D CIR arrays, we need to use a 2D convolution. This implies that the layers need to be extended by another dimension. This does not change how the network behaves, it simply allows us to use the 2D functions of the TensorFlow Lite Micro library. Another hurdle is that the TensorFlow library automatically rescales the inputs and outputs using the *Expand Dims* layer, which is unavailable in the TensorFlow Lite Micro library. For this reason, we manually reshaped some inputs and outputs, which forces the TensorFlow library to use the *Reshape* and the *Strided Slice* layers instead. The RRM is repeated for N

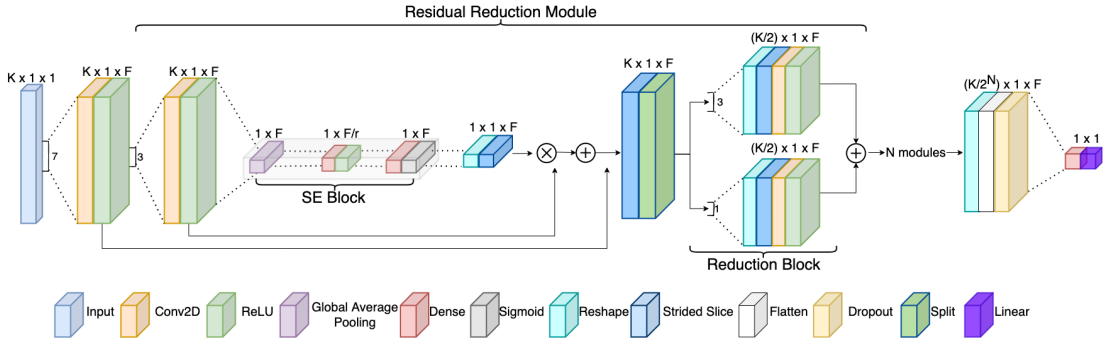


Figure 4.7: **REMNet regression architecture** based on [11]. K are the number of samples of our CIR, F is the number of filters, and N is the number of residual reduction modules.

times before returning the mitigation prediction. The parameters for the REMNet were found using a random search approach proposed by [11]. The number of samples in one CIR $K = 152$ is used as the input, the number of filters $F = 16$, the number of residual reduction modules (RRM) $N = 3$, and the number of filters is reduced by the factor of $r = 8$ in the SE block. Apart from the input layer, which has a kernel size of 7, all convolutional layers use a kernel size of 3. For early stopping, we use a patience of 10 and monitor the validation loss, and as the batch size we use 1, which is stochastic gradient descent.

REMNet classification: In order to use the REMNet model for classification, we combine the methodology of [11] and [24] and replace the activation function of the last dense layer with a `softmax`. Instead of having one output node, we now also have two nodes, which represent the LOS and the NLOS classes, as can be seen in Fig. 4.8. These nodes calculate the probabilities for LOS and NLOS and the node with the higher probability will determine the classification.

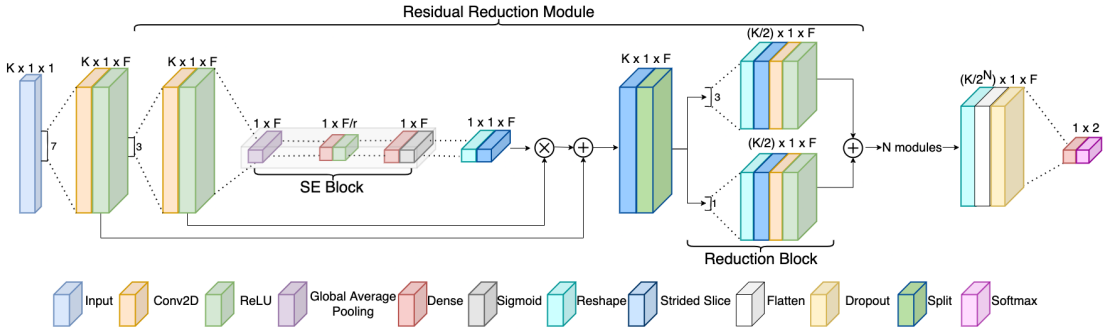


Figure 4.8: **REMNet classifier architecture** based on [10] and [11]. K are the number of samples of our CIR, F is the number of filters, and N is the number of residual reduction modules.

4.5.4 REMNet Results

The REMNet performs well on LOS, WLOS, and NLOS data as can be seen in Fig. 4.9. The raw REMNet mitigation only introduces a slight error in LOS measurements that have a larger error; otherwise, it can even improve the LOS for small errors. The REMNet classifier has the highest overall F1-score of 92.2% as can be seen in Tab. 4.3c and, together with the REMNet regression, perform better than the DW LOS estimation with the REMNet regression. The REMNet regressor with the REMNet classifier achieves an overall improvement for all cases of the 90^{th} percentile $|RRE|$ of 50.7% from $1.331m$ to $0.658m$, and an improvement of the MAE of 39.4% from $0.188m$ to $0.114m$, as can be seen in more detail in Tab. 4.3a. The REMNet regressor together with the DW LOS estimation performs nearly identically in LOS and achieves good results in WLOS and NLOS. In total, it achieves an improvement of the 90^{th} $|RRE|$ of 31% from $1.331m$ to $0.919m$, and an improvement of the MAE of 28.7% from $0.188m$ to $0.134m$.

4.6 Results Summary and Discussion

This section summarizes the best performing classification and mitigation combinations. In Tables 4.3a, 4.3b, and 4.3c all of the median absolute errors (MAE), the 90^{th} percentile absolute residual ranging errors ($|RRE|$), the R^2-scores, and the F1-scores are shown. Fig. 4.10 shows the SVC together with the SVR as the `SVM with Classifier`

with a yellow line, the CNN with the DW LOS estimation as the `CNN with DW LOS Estimation` with a green line, and the REMNet regressor with the REMNet classifier as the `REMNet with Classifier` with a red line. All three correction methods per-

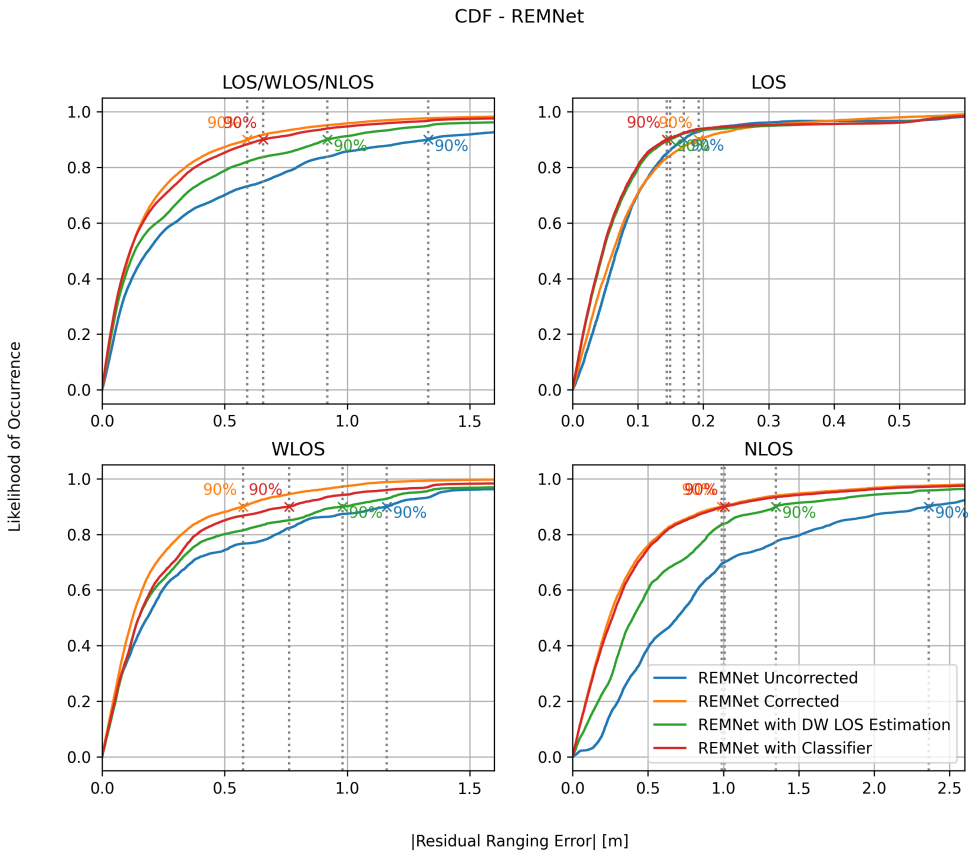


Figure 4.9: **CDF plot of the residual ranging errors**. The `REMNet` Corrected line describes the raw regression mitigation of the REMNet. The `REMNet with DW LOS Estimation` and `REMNet with Classifier` classify the data with the DW LOS estimation and the REMNet classifier, respectively, to either mitigate LOS with the bias correction and NLOS with the REMNet regression.

form nearly identically for the LOS cases with a 90^{th} percentile $|RRE|$ difference within only 9.4% and a MAE difference within 1.5%. For the WLOS cases, the SVR with the SVC and the CNN with the DW LOS estimation perform very similarly; however, the REMNet regressor with the REMNet classifier outperform them both, the SVMs by 25.5%, and the CNN with the DW LOS estimation by 10%, regarding the 90^{th} percentile $|RRE|$. The REMNet regressor with the REMNet classifier also outperform the SVMs by 39.1% and the CNN with the DW LOS estimation by 19.6%, regarding the MAE. In NLOS conditions, the SVMs and NNs all manage to reduce the $|RRE|$ well. Again, the

REMNet regressor and the REMNet classifier, in terms of the 90^{th} percentile $|RRE|$, perform 29.3% better than the SVMs and 10.8% better than the CNN with the DW LOS estimation. Regarding the MAE, they also perform 27.4% better than the SVMs and 21.9% better than the CNN with the DW LOS estimation. Overall for LOS, WLOS, and NLOS, the REMNet regressor and classifier are 28.6% better than the SVMs and 17.8% better than the CNN with the DW LOS Estimation in terms of the 90^{th} percentile $|RRE|$, and 32% better than the SVMs and 22.3% better than the CNN with the DW LOS estimation in terms of the MAE.

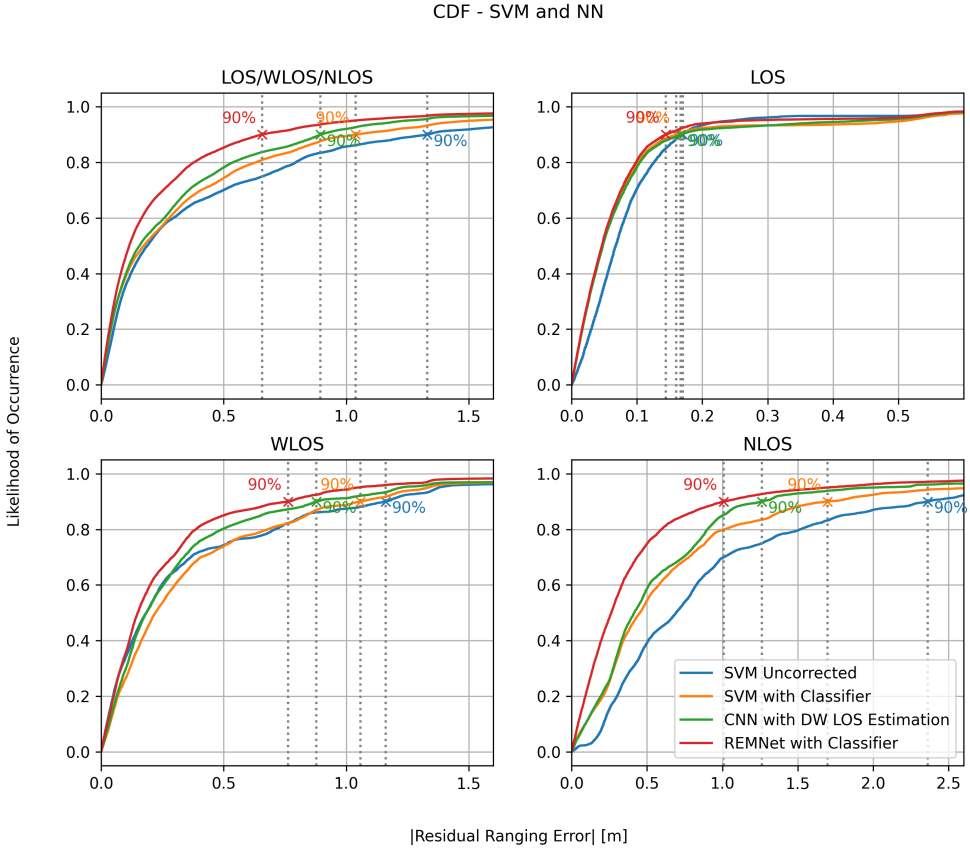


Figure 4.10: **CDF plot of the residual ranging errors**. The figures show the best performing classification and mitigation combinations of the SVMs and NNs to reduce the RRE.

The mean R^2-score shown in Tab. 4.3b is calculated from the results of each individual run of the k-fold. The same trend can be seen with the MAE and the 90^{th} percentile absolute RRE. The LOS measurements have a better R^2-score when a classifier is applied, while the WLOS and NLOS measurements only have a slight decrease in the R^2-score. In terms of the standard deviation of the R^2-score, the classifier decreases the σ for LOS measurements while not really changing it for WLOS and NLOS measurements. This

shows that the classifier adds the benefits we wanted, i.e., it nearly prevents the negative regression effects on the LOS while still improving the WLOS and NLOS measurements.

The mean F1-scores in Table 4.3c are also calculated with the results of each individual run of the k-fold algorithm. It shows how similar the DW LOS estimation, the SVC, and the REMNet classifier correctly classify LOS situations with more than 95% accuracy. However, only the REMNet classifier achieves an accuracy above 90% when classifying NLOS data, while the DW LOS estimation drops below 70% and the SVC drops below 85%. The standard deviation also suggests that all three classifiers are consistent with their prediction accuracy for LOS. For NLOS, the standard deviation is a bit higher for the SVC and the DW LOS estimation. Therefore, the REMNet classifier is the best classifier in terms of the F1-score in accurately predicting LOS and NLOS measurements. However, the DW LOS estimation is not bad for its simplicity and if the used microcontroller does not have enough memory or computing power, it is a good alternative to correctly identify LOS situations.

The evaluation of the experimental campaign has shown that REMNet regression performs well on LOS, WLOS, and NLOS data, while the SVM and CNN regressors can mitigate the residual ranging error for WLOS and NLOS measurements, however, they worsen LOS measurements. For this reason, we implemented three different classifiers that decide which measurements need to be corrected. LOS measurements are corrected with the bias correction, while NLOS measurements are corrected with the regression output. The WLOS cases lie somewhere in between LOS and NLOS cases and cannot easily be distinguished and are, therefore, classified as either LOS or NLOS. The DW LOS estimation performs worse on predicting a LOS measurement compared to the SVC and REMNet classifier, while also needing fewer resources. This makes it a viable option if the microcontroller does not have enough memory or processing power for machine learning models. Otherwise, the REMNet classifier outperforms both the SVM and the DW LOS estimation and is the recommended classifier model.

The REMNet regressor and the REMNet classifier perform the best compared to the SVR and SVC, and to the CNN and DW LOS estimation in terms of the MAE, 90^{th} percentile absolute RRE, R^2-score, and F1-score. If the CNN is used with a better classifier, it would perform similarly to the REMNet; however, the model size of the REMNet is smaller and fits onto a microcontroller. There is an available TensorFlow Lite library for microcontrollers for easy portability of the NN models to a microcontroller. This makes the REMNet regressor and classifier the best choice to implement on a microcontroller. Therefore the next chapter describes the implementation of the REMNet regression and classification models.

	MAE [m]				90^{th} Percentile $\|RRE\|$ [m]			
	ALL	LOS	WLOS	NLOS	ALL	LOS	WLOS	NLOS
Uncorrected	0.188	0.067	0.184	0.693	1.331	0.170	1.161	2.361
SVM Corrected	0.201	0.097	0.208	0.402	0.931	0.345	0.893	1.581
SVM with DW LOS Estimation	0.202	0.048	0.212	0.508	1.077	0.203	1.028	1.833
SVM with Classifier	0.174	0.047	0.221	0.453	1.039	0.160	1.058	1.698
CNN Corrected	0.191	0.159	0.172	0.267	0.637	0.475	0.521	0.987
CNN withDW LOS Estimation	0.156	0.048	0.185	0.415	0.895	0.167	0.877	1.261
REMNet Corrected	0.116	0.062	0.124	0.243	0.592	0.193	0.575	0.989
REMNet with DW LOS Estimation	0.134	0.047	0.149	0.400	0.919	0.149	0.981	1.348
REMNet with Classifier	0.114	0.046	0.149	0.263	0.658	0.144	0.762	1.007

(a) Performance of the mitigation in all cases regarding the median absolute error (MAE), and the residual ranging error (RRE) of the 90^{th} percentile. The results for each of the 10-fold runs are stacked and evaluated together.

	μ R^2 score				σR^2 score			
	ALL	LOS	WLOS	NLOS	ALL	LOS	WLOS	NLOS
SVM Corrected	0.204	-2.668	-0.593	-0.199	0.140	2.336	0.680	0.577
SVM with DW LOS Estimation	-0.045	-0.962	-0.796	-0.670	0.253	1.412	0.568	0.722
SVM with Classifier	0.126	-0.859	-0.789	-0.332	0.204	0.954	0.647	0.669
CNN Corrected	0.638	-5.036	-0.023	0.538	0.168	5.308	1.351	0.274
CNN with DW LOS Estimation	0.235	-0.340	-0.585	-0.187	0.320	0.972	1.125	0.579
REMNet Corrected	0.636	-0.614	0.153	0.492	0.195	1.665	0.623	0.288
REMNet with DW LOS Estimation	0.206	-0.104	-0.435	-0.221	0.310	0.605	0.702	0.570
REMNet with Classifier	0.527	-0.188	-0.316	0.391	0.217	0.836	0.768	0.375

(b) Mean R^2-score of the classifications calculated in each of the 10-fold runs.

	μ F1 score			σ F1 score		
	ALL	LOS	NLOS	ALL	LOS	NLOS
DW LOS Estimation	0.736	0.952	0.660	0.122	0.025	0.123
SVC	0.851	0.981	0.841	0.107	0.012	0.096
REMNet Classifier	0.922	0.982	0.965	0.130	0.026	0.035

(c) Mean F1-score of the classifications calculated in each of the 10-fold runs.

Table 4.3: Regression and classification results for SVM, CNN and REMNet.

5 Embedded Implementation

The primary goal of the embedded implementation is to validate that the REMNet model can correctly mitigate measurements on a microcontroller that are recorded on the DWM1001-DEV boards. This chapter explains the procedure of implementing the REMNet regression and classification models onto a nRF52840 DK microcontroller. The reason for implementing the models to a new device instead of the DWM1001-Dev board is that Arduino offers a ready-to-use TensorFlow Lite library for microcontrollers which is supported by the nRF52840 DK, and not by the DWM1001-DEV board. Note that our goal is not to provide a full-fledged implementation integrated into a microcontroller, but rather to prove the feasibility of running our ML models on a microcontroller.

5.1 Hardware

The nRF52840 DK is an ARM Cortex-M4 with floating point support and features an IEEE 802.15.4 narrow-band radio. It uses a Segger J-Link OB, or a USB interface for programming and debugging. The kit is Arduino Uno Rev3 compatible, which enables it to be used with a variety of third party shields and libraries. The stand-alone nRF52840 DK [57] does not support UWB. However, due to its compatibility with a variety of shields, one can add on top a UWB shield made by TU Graz that enables a DW1000 radio. We do not perform the ranging on the nRF52840 DK, as this would require a new calibration of the device and could introduce new problems. Nevertheless, this could be addressed in future work. The nRF52840 DK has $1MB$ of flash and $256KB$ of RAM available [58].

5.2 Software

5.2.1 TensorFlow

TensorFlow is an open source machine learning platform that offers libraries and tools to easily implement different machine learning algorithms. It manages everything from the inputs, the network structure, the processing unit, and the outputs, to saving the model or deploying it on different architectures, as is shown in Fig. 5.1. The TPU is a tensor processing unit that was specifically designed to handle large amounts of data. For this Master thesis, we deployed our model as a TensorFlow Lite model for microcontrollers designed to be lightweight.

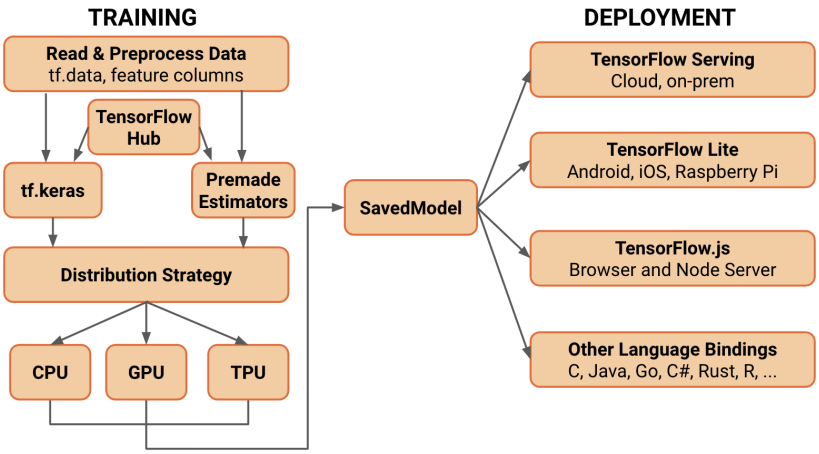


Figure 5.1: TensorFlow architecture.

5.2.2 TensorFlow Lite Micro Model Conversion

The TensorFlow Lite Micro library is a slimmed down TensorFlow library specifically designed to run on microcontrollers with limited resources. This is done by converting the TensorFlow model to a TensorFlow Lite (TFLite) model and then creating a hex dump of that TFLite model to create a C array of the model. This C array can then be used by the TensorFlow Lite Micro library to run the model. To ensure a small footprint, only the needed layers for the models are flashed onto the device. As not all layers are available, we had to convert the input dimensions to fit a 2D data space as the library only supports 2D functions. The 1D layers are easily transformed into the 2D space by expanding the inputs and kernels with an extra dimension. This means that instead of having an input dimension of $(152, 1)$, we now have an input dimension of $(152, 1, 1)$. This is comprised of the CIR length, CIR layers, and the number of features per sample. Similarly the kernel size is now transformed from (3) to $(3, 1)$. Another change had to be made to the models. As the model architecture shrinks and expands the model dimensions during the RRM block, TensorFlow automatically used the layer *Expand Dimensions*. This layer, however, is not supported by the TensorFlow Lite Micro library. Therefore, we manually resized the input data when needed before each layer to ensure that the layers *Resize* and *Strided Slice* are used instead. The commented code for the REMNet regressor and classifier layers can be seen in Listing 7.1 and 7.2.

5.2.3 Implementation Details: TensorFlow Lite Micro Library

The TensorFlow Lite Micro C++ library allows TensorFlow Lite models to run on constrained devices. The most important files of the library are the `all_ops_resolver.h`, `micro_mutable_op_resolver.h`, `micro_error_reporter.h`, and `micro_interpreter.h`. The `all_ops_resolver.h` contains all operations available to a microcontroller and it

needs the most amount of memory. For this reason, the `micro_mutable_op_resolver.h` should be used instead, as this only loads the specified operations into memory. For the REMNet models, we only need to load 13 operations. These include the layers shown in Figs. 4.7 and 4.8 (except for the input layer), the multiplication, and addition layer. Not all operations of the standard TensorFlow library are available in the TensorFlow Lite Micro library, so care has to be taken to only use compatible operations for the REMNet model. The `micro_error_reporter.h` handles the debug information, and the `micro_interpreter.h` handles and runs the models.

After the operations have been specified, the C array of the TensorFlow Lite conversion can be loaded. The array is thereby mapped to a usable data structure by the library. In order to run inference on a model, some memory needs to be allocated for the input, output, and intermediate arrays (these arrays are called tensors). The allocation of a tensor arena (maximum size needed to store all tensors) is model specific and is usually determined by experimentation. Our regression and classification models each receive a tensor arena size of $39 * 1024$. The interpreter can then be instantiated with all the created variables and the tensors can be allocated. If the tensor arena size is too small, this step will through an error to indicate the memory mismatch. The interpreter provides pointers to its input and output tensors, from where the model receives its input data and the output can be read. At this point, the interpreter is ready to run inference.

5.2.4 Quantization

A full integer quantization was done to minimize the size of the model and to improve the processing speed. This quantization method is also compatible with the TensorFlow Lite Micro library. Additionally, a quantization layer and a dequantization layer are added. The full integer quantization ensures that all math operations are integer quantized except for the input and output. The reason for this is that some input CIR values are smaller than 1, and the output is the error prediction in metres; however, the magnitude is often in the range of decimetres. In order for the full integer quantization to work, we need to calibrate the minimum and maximum of the floating-point tensors. This is done by supplying a representative data set of a few hundred sample CIRs out of the training data set [59] [60].

5.3 Feasibility of REMNet on an Embedded Device

5.3.1 Workflow

Figure 5.2 demonstrates the workflow followed in this work. The data acquisition for the measurement campaign, as described in Sect. 4.1.1, and the independent final test data was performed on the DWM1001-DEV modules. The final test data was recorded in a private apartment with different obstacles to see if the models still perform well in a new environment. The obstacles in the final test consisted of a mattress, blankets, a human, shelves, walls, an oven, a table, a laptop screen, and clothes. All of the measured data

was sent to a PC via a JLINK connection and stored locally for further processing. The training CIRs from the measurement campaign and the final test CIRs are preprocessed as explained in Sect. 4.1.4. The training CIRs are then used to train the REMNet regressor and classifier and the finished trained model is converted into a TFLite model. The C array describing the TFLite model is then flashed onto the nRF52840 DK, where it can be used to perform classification and regression on the microcontroller. One of the final test CIRs is then sent to the nRF52840 DK via a serial connection, where it is fed into the regressor and classifier. Each prediction is then sent back to the PC via the serial connection and the next CIR is requested. In the end, the final test ranges are mitigated according to the classification and regression results.

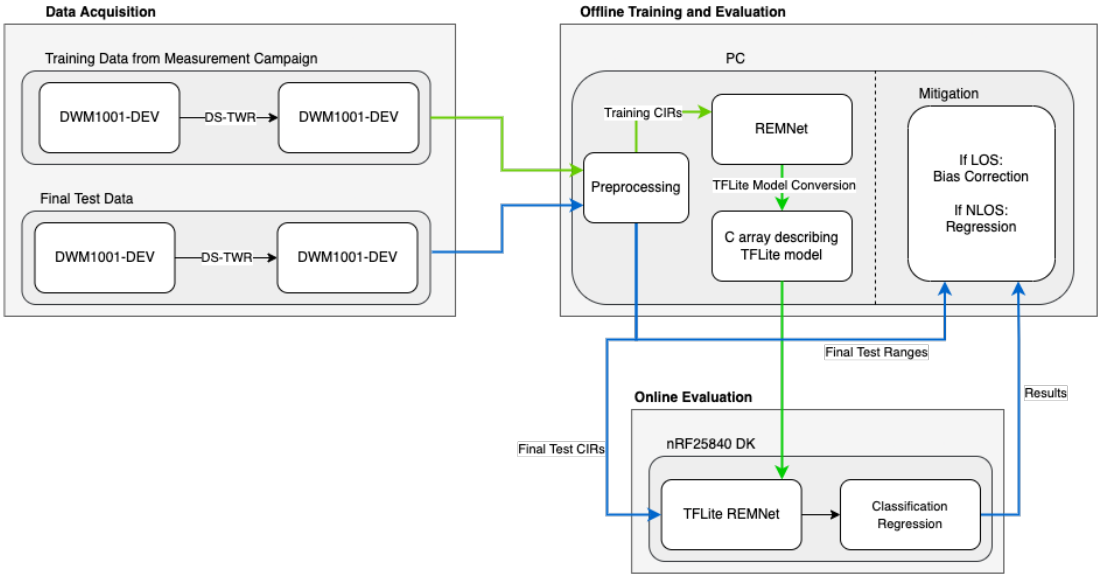


Figure 5.2: Workflow followed in this work, starting from the data collection with which the REMNet models are trained on. The trained model is converted to a TFLite model and is flashed onto the nRF52840 DK as a C array. The final test classifications and regressions are done on the microcontroller and sent back for the evaluation.

5.3.2 Results and Discussion

The REMNet lite model was tested using the non-quantized and the full integer quantized version. For both examples, we measured the time it takes the nRF52840 DK to run the classification model and, afterwards, the regression model on the same example CIR. As can be seen in Table 5.1, the full integer quantized model is 43% faster than the non-quantized model, needing only $138ms$ to perform classification and regression compared to the non-quantized model, which needs 242ms. The quantized model also reduces the model's size in the program storage space by 19%, while still needing the

same amount of dynamic memory to run the model. All in all the full integer quantization reduces the model size and increases the execution speed, while not lowering the accuracy.

TensorFlow Lite Micro	Non-Quantized	Full Integer
Execution time	242 ms	138 ms
RAM on nRF52840 DK	88 kB	88 kB
ROM on nRF52840 DK	334 kB	270 kB

Table 5.1: Regression and classification runtime and file size comparison of the non-quantized and the full integer quantized TensorFlow Lite Micro models running on the nRF52840 DK.

The results of the final test data are presented in Fig. 5.3 as box plots. Each measurement point of the final test data is plotted in the x-axis as the uncorrected measurement (LOS, WLOS, and NLOS) in blue, together with the correction of that measurement in orange, and the correction with the full integer quantized model in green. A box plot shows a few significant statistics, in this case regarding the residual ranging errors. The box contains 50% of the residual ranging errors, where the middle line inside the box denotes the median. The whiskers above and below this box each hold 25% of the remaining residual ranging errors. The single points that are shown are considered outliers. Each of the three plots in Fig. 5.3 refers to one of the cases (LOS, WLOS, and NLOS). There are 6 test measurements per case, resulting in 18 test points. As can be seen in Fig. 5.3a, the LOS test measurements are improved by a few centimetres with exception of the first measurement point. The REMNet TensorFlow Lite model and the full integer quantized TensorFlow Lite model both correctly classify most of the measurements and only perform the bias correction. For the WLOS cases in Fig. 5.3, the mitigation methods have a harder time, since they are either classified as NLOS (which induces a larger correction), or as LOS (where the bias correction hardly has an effect). In any case, the errors lie within $10cm$ accuracy, which is acceptable as UWB only allows for sub $10cm$ accuracy [15]. Some of the NLOS cases shown in Fig. 5.3.c are greatly improved. In three cases, the residual ranging error is reduced by $50 - 100cm$. When a NLOS measurement is classified as LOS, then the bias correction has a negative mitigation effect, as the bias correction is calibrated towards real LOS cases without obstacles.

The box plot diagram visualizes the results, presented in Tabs. 5.2a, 5.2b, and 5.2c. Tab. 5.2a shows the exact values of the median, mean, and standard deviation of the $|RRE|$. Note that these errors are absolute errors, in order to be compliant with the results section of the experimental campaign. It can be seen in Fig. 5.3.a and in Tab. 5.2c that the non-quantized and the full integer quantized model perform identically for LOS conditions. In WLOS conditions, there is a slight offset between the two models, where the quantized model has a slightly higher mean $|RRE|$ by 2 cm, but a slightly lower standard deviation of the $|RRE|$ by 6 cm. This correlates to Fig. 5.3.b, where measurement point 6 has a higher standard deviation towards a lower residual ranging

error. For NLOS conditions, the situation is different, as the median, mean, and standard deviation of the $|RRE|$ decreases, when using the quantized model. This can also be seen in Fig. 5.3.c, where the standard deviation is higher for measurement points 4, 13, and 14. The decreased median and mean values are probably due to measurement point 4, where the non-quantized model has a large deviation of the $|RRE|$ compared to the quantized model. Otherwise, Fig. 5.3.c suggests that the non-quantized model, in general, achieves more accurate results in NLOS conditions. Nevertheless, the quantized model only deviates by 12.6% regarding the median $|RRE|$, and 7% regarding the mean $|RRE|$.

In terms of the R^2-score, shown in Tab. 5.2b, the two models have similar scores. In LOS conditions the non-quantized model slightly achieves a better score, however, in WLOS and NLOS the quantized model performs slightly better.

In terms of the F1-score, shown in Tab. 5.2c, the non-quantized model performs slightly better than the quantized model in LOS and NLOS conditions. However, the accuracy deviation is less than 0.3% for LOS and NLOS conditions.

TensorFlow Lite Micro	Non-Quantized			Full Integer		
	LOS	WLOS	NLOS	LOS	WLOS	NLOS
Median $\|RRE\|$ (m)	0.046	0.286	0.555	0.046	0.286	0.485
Mean $\|RRE\|$(m)	0.060	0.334	0.500	0.060	0.336	0.465
Standard Deviation $\|RRE\|$ (m)	0.050	0.189	0.318	0.050	0.183	0.302

(a) Absolute residual ranging errors after the bias correction (classified as LOS) and the regression correction (classified as NLOS).

TensorFlow Lite Micro	Non-Quantized			Full Integer		
	LOS	WLOS	NLOS	LOS	WLOS	NLOS
R^2-Score	-0.080	-3.502	-1.026	-0.083	-3.463	-0.771

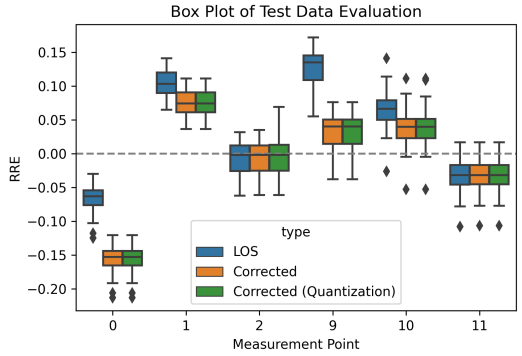
(b) R^2-score of the non-quantized and full integer quantized models.

TensorFlow Lite Micro	Non-Quantized		Full Integer	
	LOS	NLOS	LOS	NLOS
F1-Score	0.996	0.818	0.994	0.815

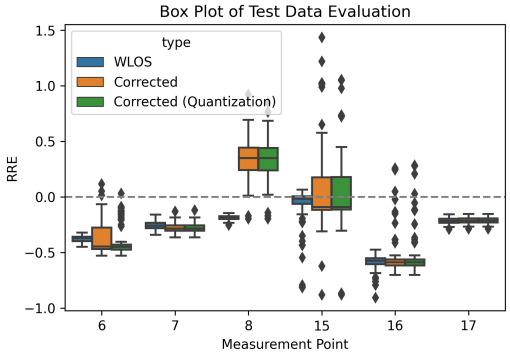
(c) F1-score of the non-quantized and full integer quantized models. The WLOS conditions are classified either as LOS or NLOS.

Table 5.2: Regression and classification results for the non-quantized and quantized TensorFlow Lite Micro models.

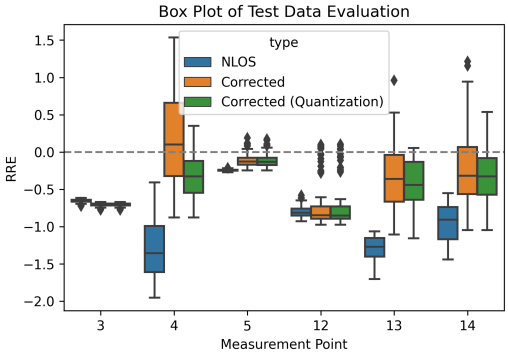
In summary, the full integer quantized model decreases the runtime, as well as the amount of memory needed by sacrificing float accuracy during the model's calculations. Nevertheless, the results show that the quantized model tends to have a smaller standard deviation and achieves very similar results as the non-quantized model does.



(a) LOS



(b) WLOS



(c) NLOS

Figure 5.3: Box plots showing the corrected residual ranging errors. The measurements are classified with the REMNet classifier and either mitigated with the bias correction or the REMNet regression output.

6 Conclusion and Future Work

In this chapter, we will recap the contributions of this Master thesis and provide an outlook on possible further work.

6.1 Conclusion

The goal of this Master thesis was to directly mitigate ranging errors on devices with constrained resources using UWB-technology. To achieve this goal, we used three promising machine learning models found in related literature (an SVM [25], a CNN [10] and a refined version of the CNN named REMNet [11]), and carried out our own measurement campaign resulting in 749 distinct measurement setups. The measurement campaign was conducted on DWM1001-DEV modules, whereas the feasibility of running inference of the ML models, was conducted on an nRF52840 DK. The evaluation of the three machine learning models on our data set showed that WLOS and NLOS measurements could be improved, however, the error of LOS measurements where worsened. For this reason, we wanted to classify LOS and NLOS measurements to not perform a mitigation on LOS but to only use the bias correction described in Sect. 2.1.6 to compensate for any gain and loss parameters in the system. This compensation is only valid for the LOS measurements, as the reference values are obtained in clear LOS and do not apply to WLOS or NLOS measurements. We used a statistical approach, an SVC, and a modified version of the REMNet to classify the measurement prior to any mitigation. By classifying into LOS and NLOS and mitigating accordingly, we achieve good mitigation results in LOS, WLOS, and NLOS cases. The SVR together with the SVC achieves an overall improvement of the MAE of 7.4% and of the 90^{th} percentile of 22%. The CNN together with the DW LOS estimation achieve an overall improvement of the MAE of 17% and of the 90^{th} percentile of 33%. The REMNet together with the REMNet classifier achieve an overall improvement of the MAE of 39.4% and of the 90^{th} percentile of 50.7%.

Our experimental campaign concludes that the REMNet performs the best and is also easily transportable onto the nRF52840 DK. We converted the REMNet TensorFlow model into a TensorFlow Lite model for microcontrollers and tested on new unseen testing data. The TensorFlow Lite model and the full integer TensorFlow Lite model both easily run on the nRF52840 DK and take 242 ms and 138 ms respectively, to run the classification and the regression model. Aside from being faster, the quantized model also reduces the size of the model by up to 54%, while not significantly reducing the performance. Furthermore, the results of this Master thesis have led to a scientific publication by Stocker et al. [30] at an international workshop.

6.2 Future Work

This Master thesis has shown that a device with constrained resources can run a classification model and a regression model to mitigate ranging errors in $138ms$. This thesis will be used to further develop classification and mitigation methods running on microcontrollers. Some concrete steps are:

- The full-fledged implementation integrated on a microcontroller. This incorporates the integration of the collection of UWB traces, as well as the inference of the classification and the mitigation models on a proper OS / toolchain.
- The SVR and the SVC are yet to be run directly on a microcontroller to investigate the model size and performance.
- The CNN can be tested with a better classifier and implemented on a device with more memory and processing power.
- At the moment, the REMNet regressor and classifier are two separate models needing their own memory. However, they only differ in their output layer. A multi-output neural network could be made to use the same trained model, and output a regression and a classification prediction at the same time. This could save memory, as only one model needs to be stored. The performance of the regressor and classifier would need to be re-evaluated.
- The REMNet regressor and classifier can be evaluated with a localization task to see if they improve the localization estimates.
- In this thesis, the regression and classification models are trained on CIRs from one responder. However, a method proposed by Großwindhager et al. [61] uses quasi simultaneous CIR responses from multiple devices to quickly localize itself. The effect of our regression and classification model on a CIR that contains more than one response can be evaluated.

Bibliography

[1] Decawave, *APS006: The Effect of Channel Characteristics on Time-stamp Accuracy in DW1000 Based Systems*, 2014.

[2] B. Großwindhager, C. A. Boano, M. Rath, and K. Romer, "Enabling Runtime Adaptation of Physical Layer Settings for Dependable UWB Communications," in *2018 IEEE 19th International Symposium on "A World of Wireless, Mobile and Multimedia Networks" (WoWMoM)*, pp. 1–11, 2018.

[3] Decawave, *DW1000 User Manual*, 2017.

[4] Decawave, *APS011 Application Note: Sources of Error in DW1000 Based Two-Way Ranging (TWR) Schemes*, 2014.

[5] G. E. F. Bellec, "Computational Intelligence (Introduction to Machine Learning) SS16." Technische Universität Graz, 2016.

[6] Q. Li, M. Yan, and J. Xu, "Optimizing Convolutional Neural Network Performance by Mitigating Underfitting and Overfitting," in *2021 IEEE/ACIS 19th International Conference on Computer and Information Science (ICIS)*, pp. 126–131, 2021.

[7] S. Kini, "Getting Started With CNN." Accessed Nov. 23, 2021 [Online] https://medium.com/@kinisanketh/getting-started-with-cnn-18c03efc7d06/, 2020.

[8] S. Sehgal, H. Singh, M. Agarwal, V. Bhasker, and Shantanu, "Data Analysis using Principal Component Analysis," in *2014 International Conference on Medical Imaging, m-Health and Emerging Communication Systems (MedCom)*, pp. 45–48, 2014.

[9] J. Khodjaev, Y. Park, and A. S. Malik, "Survey of NLOS Identification and Error Mitigation Problems in UWB-Based Positioning Algorithms for Dense Environments," *Annales des Télécommunications*, vol. 65, pp. 301–311, 2010.

[10] S. Angarano and M. Chiaberge, "Deep Learning Methodologies for UWB Ranging Error Compensation," Master's thesis, Politecnico di Torino, 2020.

[11] S. Angarano, V. Mazzia, F. Salvetti, G. Fantin, and M. Chiaberge, "Robust Ultra-wideband Range Error Mitigation with Deep Learning at the Edge," *CORR – arXiv preprint 2011.14684*, 2020.

[12] R. S. Rosli, M. H. Habaebi, and M. R. Islam, "Characteristic Analysis of Received Signal Strength Indicator from ESP8266 WiFi Transceiver Module," in *2018 7th International Conference on Computer and Communication Engineering (ICCCE)*, pp. 504–507, 2018.

[13] D. Dardari, A. Conti, U. Ferner, A. Giorgetti, and M. Z. Win, "Ranging With Ultrawide Bandwidth Signals in Multipath Environments," *Proceedings of the IEEE*, vol. 97, no. 2, pp. 404–426, 2009.

[14] M. Ibrahim, H. Liu, M. Jawahar, V. Nguyen, M. Gruteser, R. Howard, B. Yu, and F. Bai, "Verification: Accuracy Evaluation of WiFi Fine Time Measurements on an Open Platform," *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking*, pp. 417–427, 2018.

[15] Decawave, "Product Brief: DWM1001-DEV." Brochure, 2017.

[16] A. F. Molisch, "Ultra-Wideband Communications: An Overview," *URSI Radio Science Bulletin*, vol. 2009, no. 329, pp. 31–42, 2009.

[17] A. F. Molisch, D. Cassioli, C.-C. Chong, S. Emami, A. Fort, B. Kannan, J. Karedal, J. Kunisch, H. G. Schantz, K. Siwiak, and M. Z. Win, "A Comprehensive Standardized Model for Ultrawideband Propagation Channels," *IEEE Transactions on Antennas and Propagation*, vol. 54, no. 11, pp. 3151–3166, 2006.

[18] S. Sharma, H. Mohammadmoradi, M. Heydariaan, and O. Gnawali, "Device-Free Activity Recognition Using Ultra-Wideband Radios," in *2019 International Conference on Computing, Networking and Communications (ICNC)*, pp. 1029–1033, 2019.

[19] B. Großwindhager, M. Rath, J. Kulmer, M. S. Bakr, C. A. Boano, K. Witrisal, and K. Römer, "SALMA: UWB-based Single-Anchor Localization System using Multipath Assistance," in *Proceedings of the 16th ACM Conference on Embedded Networked Sensor Systems*, SenSys '18, pp. 132–144, 2018.

[20] A. Alarifi, A. Al-Salman, M. Alsaleh, A. Alnafessah, S. Al-Hadhrami, M. A. Al-Ammar, and H. S. Al-Khalifa, "Ultra Wideband Indoor Positioning Technologies: Analysis and Recent Advances," *Sensors*, vol. 16, pp. 1–36, 2016.

[21] M. Heydariaan, H. Mohammadmoradi, and O. Gnawali, "Toward Standard Non-line-of-sight Benchmarking of Ultra-wideband Radio-based Localization," in *2018 IEEE Workshop on Benchmarking Cyber-Physical Networks and Systems (CPS-Bench)*, pp. 19–24, 2018.

[22] Decawave, *APS006 Application Note Channel Effects on Communications Range and Time Stamp Accuracy in DW1000 Based Systems*, 2014.

[23] V. Barral, C. J. Escudero, and J. A. García-Naya, "NLOS Classification Based on RSS and Ranging Statistics Obtained from Low-Cost UWB Devices," in *2019 27th European Signal Processing Conference (EUSIPCO)*, pp. 1–5, 2019.

[24] K. Bregar and M. Mohorčič, "Improving Indoor Localization Using Convolutional Neural Networks on Computationally Restricted Devices," *IEEE Access*, vol. 6, pp. 17429–17441, 2018.

[25] H. Wymeersch, S. Maranò, W. M. Gifford, and M. Z. Win, "A Machine Learning Approach to Ranging Error Mitigation for UWB Localization," *IEEE Transactions on Communications*, vol. 60, no. 6, pp. 1719–1728, 2012.

[26] J. A. Afif, L. K. Seong, and S. Krishnan, "A SVM Approach to UWB-IR based Positioning under NLOS Conditions," in *2010 IEEE International Conference on Communication Systems*, pp. 233–237, 2010.

[27] C. Mao, K. Lin, T. Yu, and Y. Shen, "A Probabilistic Learning Approach to UWB Ranging Error Mitigation," in *2018 IEEE Global Communications Conference (GLOBECOM)*, pp. 1–6, 2018.

[28] A. Nessa, B. Adhikari, F. Hussain, and X. N. Fernando, "A Survey of Machine Learning for Indoor Positioning," *IEEE Access*, vol. 8, pp. 214945–214965, 2020.

[29] C. L. Sang, B. Steinhagen, J. D. Homburg, M. Adams, M. Hesse, and U. Rückert, "Identification of NLOS and Multi-path Conditions in UWB Localization using Machine Learning Methods," *Applied Sciences*, vol. 10, no. 11, 2020.

[30] M. Stocker, M. Gallacher, C. A. Boano, and K. Römer, "Performance of Support Vector Regression in Correcting UWB Ranging Measurements under LOS/N-LOS Conditions," in *CPS-IoTBench 2021 - Proceedings of the 2021 Benchmarking Cyber-Physical Systems and Internet of Things*, pp. 6–11, Association of Computing Machinery, 2021.

[31] "IEEE Standard for Low-Rate Wireless Networks," *IEEE Std 802.15.4-2015 (Revision of IEEE Std 802.15.4-2011)*, pp. 1–709, 2016.

[32] I. Oppermann, M. Hámáláinen, and J. Iinatti, *UWB Theory and Applications*. John Wiley & Sons, Ltd, 2004.

[33] M. Chiani and A. Giorgetti, "Coexistence Between UWB and Narrow-Band Wireless Communication Systems," *Proceedings of the IEEE*, vol. 97, no. 2, pp. 231–254, 2009.

[34] Decawave, *APS013 Application Note The Implementation of Two- Way Ranging with the DW1000*, 2015.

[35] Decawave, *DWM1001 Module Development Board Datasheet*, 2017.

[36] Decawave, *APS014: DW1000 Antenna Delay Calibration*, 2018.

[37] mynewt.apache.org, "Apache Mynewt." Accessed Nov. 18, 2021 [Online] https://mynewt.apache.org/.

[38] M. Awad and R. Khanna, *Efficient Learning Machines*. Springer Nature, 2015.

[39] C. Junli and J. Licheng, "Classification Mechanism of Support Vector Machines," in *WCC 2000 - ICSP 2000. 2000 5th International Conference on Signal Processing Proceedings. 16th World Computer Congress 2000*, vol. 3, pp. 1556–1559, 2000.

[40] A. J. Smola and B. Schölkopf, "A Tutorial on Support Vector Regression," *Statistics and Computing*, vol. 14, pp. 199–222, 2004.

[41] T. Pock, "Sparse Kernel Machines." Technische Universität Graz, 2020.

[42] D. Masters and C. Luschi, "Revisiting Small Batch Training For Deep Neural Networks," *CORR – arXiv:1804.07612*, 2018.

[43] D. P. Kingma and J. L. Ba, "ADAM: A Method for Stochastic Optimization," *CORR – arXiv:1412.6980*, 2017.

[44] R. Hecht-Nielsen, "Theory of the Backpropagation Neural Network," in *International 1989 Joint Conference on Neural Networks*, vol. 1, pp. 593–605, 1989.

[45] S. Albawi, T. A. Mohammed, and S. Al-Zawi, "Understanding of a Convolutional Neural Network," in *2017 International Conference on Engineering and Technology (ICET)*, pp. 1–6, 2017.

[46] F. Song, Z. Guo, and D. Mei, "Feature Selection using Principal Component Analysis," in *2010 International Conference on System Science, Engineering Design and Manufacturing Informatization*, vol. 1, pp. 27–30, 2010.

[47] Y.-H. Jo, J.-Y. Lee, D.-H. Ha, and S.-H. Kang, "Accuracy Enhancement for UWB Indoor Positioning Using Ray Tracing," in *2006 IEEE/ION Position, Location, And Navigation Symposium*, pp. 565–568, 2006.

[48] S. Wu, Y. Ma, Q. Zhang, and N. Zhang, "NLOS Error Mitigation for UWB Ranging in Dense Multipath Environments," in *2007 IEEE Wireless Communications and Networking Conference*, pp. 1565–1570, 2007.

[49] B. Alavi and K. Pahlavan, "Modeling of the TOA-based Distance Measurement Error Using UWB Indoor Radio Measurements," *IEEE Communications Letters*, vol. 10, no. 4, 2006.

[50] J. Borrás, P. Hatrack, and N. B. Manclayam, "Decision Theoretic Framework for NLOS Identification," in *VTC '98. 48th IEEE Vehicular Technology Conference. Pathway to Global Wireless Revolution (Cat. No.98CH36151)*, vol. 2, pp. 1583–1587, 1998.

[51] K. Gururaj, A. K. Rajendra, Y. Song, C. L. LAW, and G. Cai, "Real-Time Identification of NLOS Range Measurements for Enhanced UWB Localization," in *2017 International Conference on Indoor Positioning and Indoor Navigation (IPIN))*, pp. 1–7, 2017.

[52] J. Fan and A. S. Awan, "Non-Line-of-Sight Identification Based on Unsupervised Machine Learning in Ultra Wideband Systems," *IEEE Access*, vol. 7, pp. 32464–32471, 2019.

[53] M. Ramadan, V. Sark, J. Gutiérrez, and E. Grass, "NLOS Identification for Indoor Localization using Random Forest Algorithm," in *WSA 2018; 22nd International ITG Workshop on Smart Antennas*, pp. 1–5, 2018.

[54] S. Maranò, W. M. Gifford, H. Wymeersch, and M. Z. Win, "NLOS Identification and Mitigation for Localization Based on UWB Experimental Data," *IEEE Journal on Selected Areas in Communications*, vol. 28, no. 7, pp. 1026–1035, 2010.

[55] scikit, "RBF SVM parameters." Accessed Dez. 3, 2021 [Online] https://scikit-learn.org/stable/auto_examples/svm/plot_rbf_parameters.html, 2021.

[56] Z. Zeng, S. Liu, and L. Wang, "NLOS Identification for UWB Based on Channel Impulse Response," in *2018 12th International Conference on Signal Processing and Communication Systems (ICSPCS)*, pp. 1–6, 2018.

[57] N. Semiconductor, "nRF52840 DK Product Brief." Brochure.

[58] N. Semiconductor, *nRF52840 Product Specification*, 2021.

[59] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference," *CORR – arXiv:1712.05877*, 2017.

[60] tensorflow.org, "Model Optimization." Accessed Nov. 19, 2021 [Online] https://www.tensorflow.org/lite/performance/model_optimization, 2021.

[61] B. Großwindhager, M. Stocker, M. Rath, C. A. Boano, and K. Römer, "SnapLoc: An Ultra-Fast UWB-Based Indoor Localization System for an Unlimited Number of Tags," in *2019 18th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*, pp. 61–72, 2019.

7 Appendix

7.1 Measurement Campaign

The following Figs. 7.1, 7.2, and 7.3 depict our measurement campaign consisting of 749 individual measurement points spread across 12 different locations inside the university building. The locations include five IT-laboratories, three hallways with intersections, a workshop, two small rooms, and one study centre.

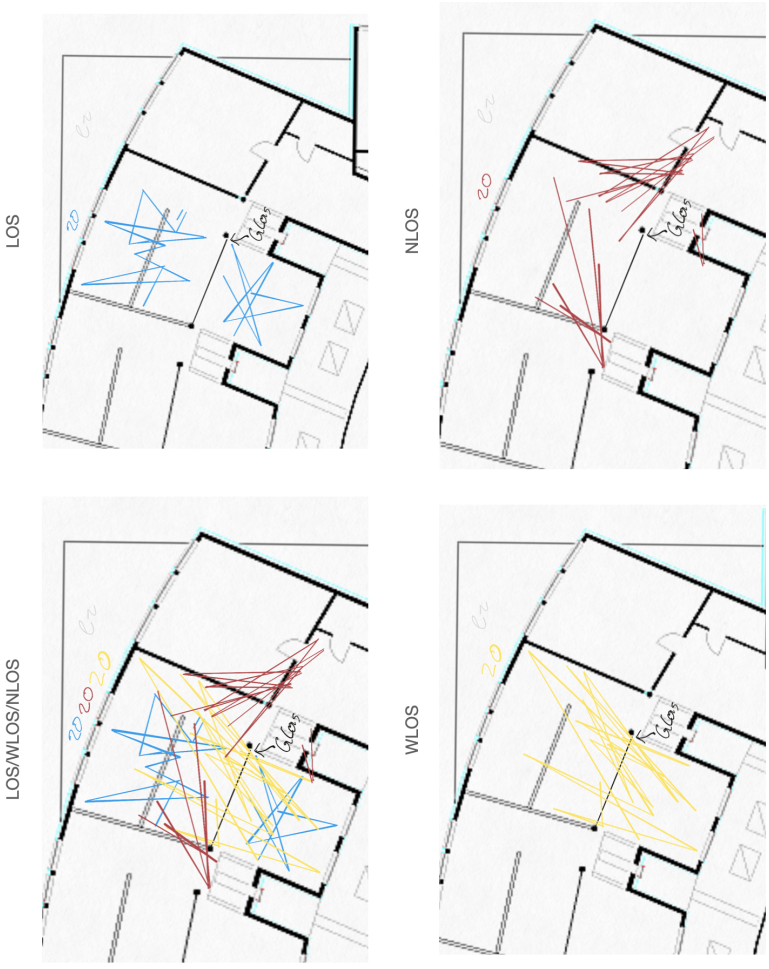


Figure 7.1: First part of the measurement campaign at Inffeldgasse 10 on the 3^{rd} floor.

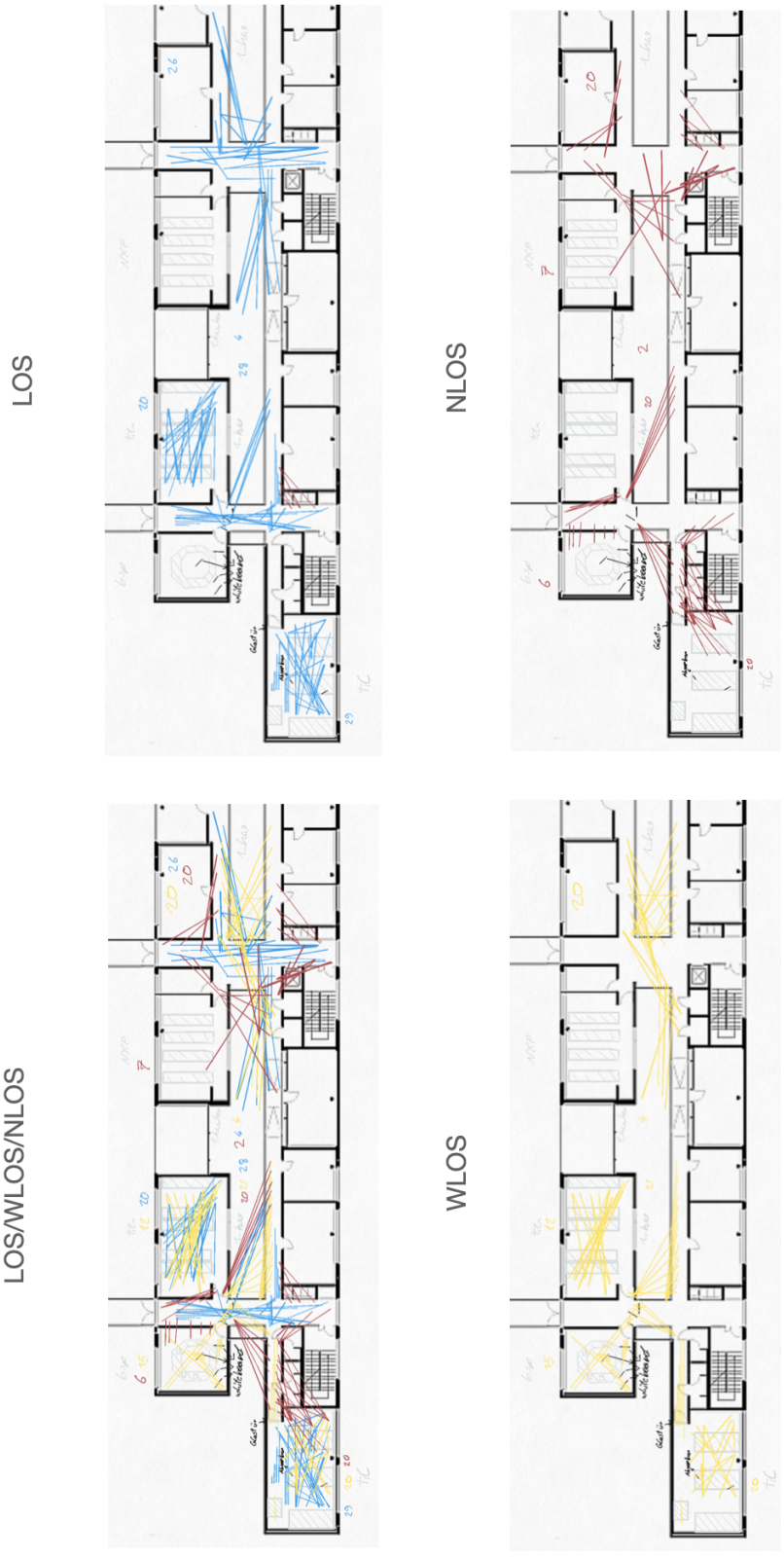


Figure 7.2: Second part of the measurement campaign at Inffeldgasse 16 on the 1^{st} floor.

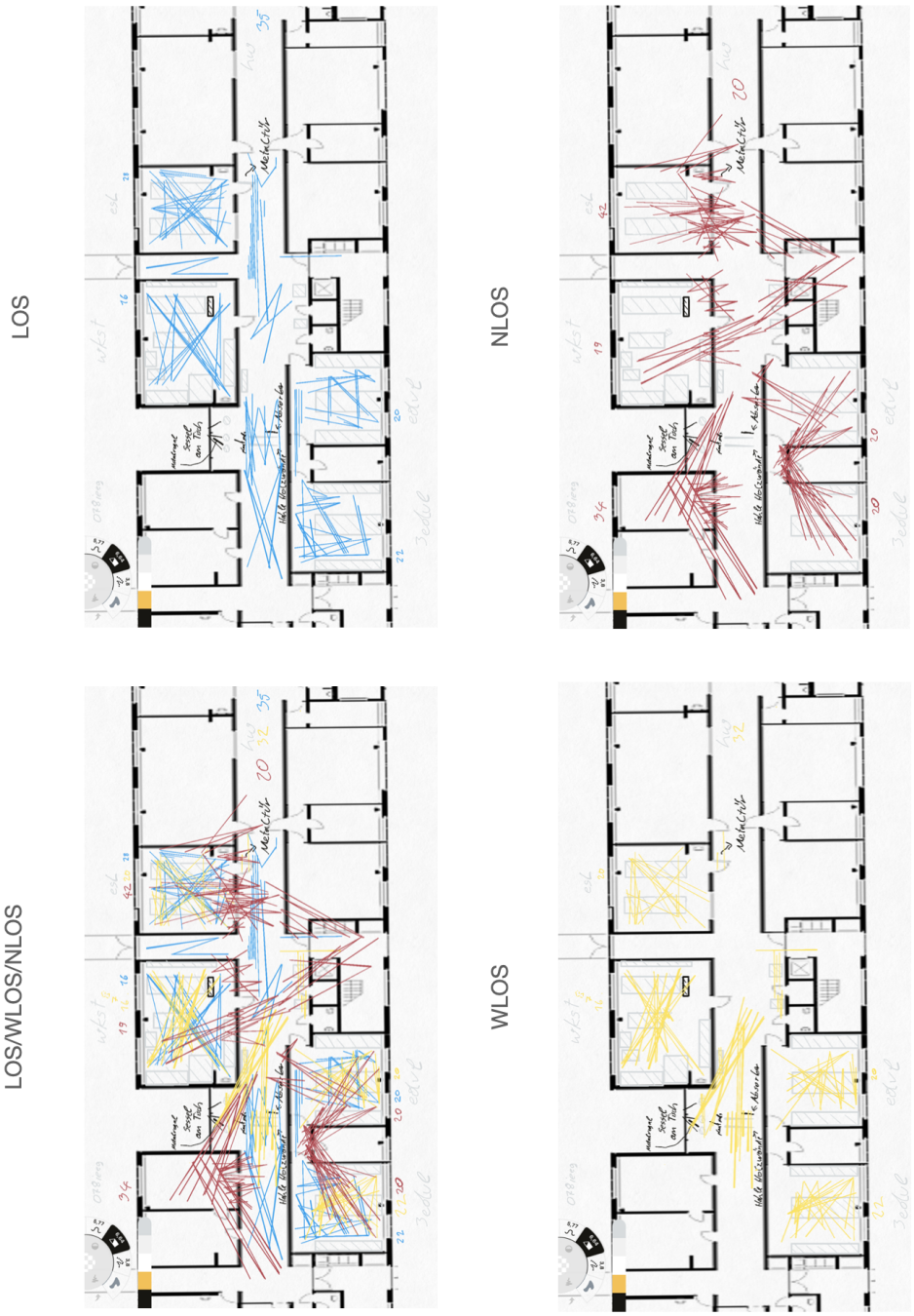


Figure 7.3: Third part of the measurement campaign at Inffeldgasse 16 on the ground floor.

7.2 Code Snippets

The following Listings 7.2 and 7.1 show the code snippets used to create the REMNet classification and the REMNet regression models that run on the nRF52840 DK.

```
def REMNet_classification_model(input_length):
    F = 16
    N = 3
    r = 8
    # define inputs
    input_shape = ((input_length, 1, 1))
    inputs = layers.Input(shape=input_shape, batch_size=None)
    identity = layers.Conv2D(filters=F, kernel_size=(7, 1), activation='relu',
        padding="same")(inputs)
    # Residual Reduction Module (RRM)
    for n in range(N):
        conv_2 = layers.Conv2D(filters=F, kernel_size=(3, 1), activation='relu',
            padding="same")(identity)
        # SE-block (Squeeze-Excitation)
        gap_se = layers.GlobalAveragePooling2D()(conv_2)
        dense_1_se = layers.Dense(units=F/r, activation="relu")(gap_se)
        dense_2_se = layers.Dense(units=F, activation="sigmoid")(dense_1_se)
        dense_2_se = dense_2_se[:, None, None,:]
        multiply_se = layers.multiply([conv_2, dense_2_se])
        addition_se = layers.add([multiply_se, identity])
        # Reduction Block
        addition_se_split = tf.split(addition_se[0], 2, axis=0)
        addition_se_split[0] = addition_se_split[0][None,:,:,:]
        addition_se_split[1] = addition_se_split[1][None,:,:,:]
        reduction_1 = layers.Conv2D(filters=F, kernel_size=(3, 1), strides=(2, 1),
            padding="same", activation="relu")(addition_se_split[0])
        reduction_2 = layers.Conv2D(filters=F, kernel_size=(3, 1), strides=(2, 1),
            padding="same",activation="relu")(addition_se_split[1])
        identity = layers.add([reduction_1, reduction_2])
    flatten_final = layers.Flatten()(identity)
    droupout_final = layers.Dropout(0.5)(flatten_final)
    # Output layer for Classification
    out = layers.Dense(units=2, activation="softmax")(droupout_final)
    classification_model = models.Model(inputs, out)
    return classification_model
```

Listing 7.1: Snippet of REMNet classification code.

```
def REMNet_regression_model(input_length):
    F = 16
    N = 3
    r = 8
    # define inputs
    input_shape = ((input_length, 1, 1))
    inputs = layers.Input(shape=input_shape)
    identity = layers.Conv2D(filters=F, kernel_size=(7, 1), activation='relu',
        padding="same")(inputs)
    # Residual Reduction Module (RRM)
    for n in range(N):
        conv_2 = layers.Conv2D(filters=F, kernel_size=(3, 1), activation='relu',
            padding="same")(identity)
        # SE-block (Squeeze-Excitation)
        gap_se = layers.GlobalAveragePooling2D()(conv_2)
        dense_1_se = layers.Dense(units=F/r, activation="relu")(gap_se)
        dense_2_se = layers.Dense(units=F, activation="sigmoid")(dense_1_se)
        dense_2_se = dense_2_se[:, None, None,:]
        multiply_se = layers.multiply([conv_2, dense_2_se])
        addition_se = layers.add([multiply_se, identity])
        # Reduction Block
        addition_se_split = tf.split(addition_se[0], 2, axis=0)
        addition_se_split[0] = addition_se_split[0][None,:,:,:]
        addition_se_split[1] = addition_se_split[1][None,:,:,:]
        reduction_1 = layers.Conv2D(filters=F, kernel_size=(3, 1), strides=(2, 1),
            padding="same", activation="relu")(addition_se_split[0])
        reduction_2 = layers.Conv2D(filters=F, kernel_size=(3, 1), strides=(2, 1),
            padding="same",activation="relu")(addition_se_split[1])
        identity = layers.add([reduction_1, reduction_2])
    flatten_final = layers.Flatten()(identity)
    droupout_final = layers.Dropout(0.5)(flatten_final)
    # Output layer for Mitigation
    out = layers.Dense(units=1, activation="linear")(droupout_final)
    regression_model = models.Model(inputs, out)
    return regression_model
```

Listing 7.2: Snippet of REMNet regression code.